Universidad Autónoma de Madrid
Escuela politécnica superior

Proyecto fin de carrera

USO DE UNA NARIZ ELECTRÓNICA ULTRA-PORTÁTIL EN ROBOTS PARA LA DETECCIÓN DE FUENTES DE ODORANTES

Ingeniería de Telecomunicación

Alejandro Pequeño Zurro
Mayo 2015

USO DE UNA NARIZ ELECTRÓNICA ULTRA-PORTÁTIL EN ROBOTS PARA LA DETECCIÓN DE FUENTES DE ODORANTES

AUTOR: Alejandro Pequeño Zurro
TUTOR: Francisco de Borja Rodríguez Ortiz

Grupo de Neurocomputación Biológica
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo 2015

Resumen

Resumen

Este proyecto desarrolla el uso de estrategias de búsqueda aplicadas en plataformas robóticas para la localización de fuentes de odorante. La investigación en el campo de las narices electrónicas y sus posibles aplicaciones es actualmente un campo activo de investigación.

Para la consecución de este objetivo utilizamos sensores de tipo MOS y estudiamos como está vinculada la percepción del odorante y el efecto de histéresis de los sensores con respecto el control de la temperatura de calefacción de los mismos. En el GNB se han utilizado anteriormente este tipo de estrategias aplicadas en el campo de la discriminación de odorante obteniendo resultados satisfactorios [1, 2]. Demostramos a lo largo de este proyecto como el uso de determinadas estrategias de modificación de temperatura de calefacción en ciclo cerrado generan comportamientos de adaptabilidad al medio en el que se sitúan.

El problema de la localización es muy complejo, para su resolución es necesario extraer información de la dispersión de odorantes y de los factores que la componen para ser explotada en los sistemas olfativos artificiales. La integración de narices artificiales en una plataforma robótica junto con los protocolos diseñados para el uso de los mismos son los componentes que conformarán este tipo de sistema. Para la creación de una plataforma integrada se ha diseñado un modelo de nariz electrónica que permite la utilización de las estrategias desarrolladas en sistemas autónomos con microcontroladores.

Los sistemas diseñados explotarán la información del entorno a través de los algoritmos de localización desarrollados. Este tipo de algoritmos se caracterizan por la utilización de señales diferenciales en sistemas multisensor de narices electrónicas en un único robot. Estos algoritmos corroboran el correcto funcionamiento de las estrategias de control de los sensores y obtiene resultados satisfactorios para ambos sistemas implementados.

Palabras Clave

Narices electrónicas, localización de odorantes, sistema multisensor, estrategia mono-robot, robótica sensorial, modulación de temperatura adaptativa, control en ciclo cerrado.

Abstract

This project develops the use of searching strategies applied on robotic platforms for odorant sources localization. Nowadays research on electronic noses and its possible usages are very active in this field.

For achieving this goal we have used MOS sensors and we have studied how the heating control of the sensor influences the odorant perception and the sensor's hysteresis. Previously in the GNB, we had used this kind of strategies applied on odour's discrimination field obtaining satisfactory results [1, 2]. Throughout this project we demonstrate how particular heat modification strategies, generate behaviors of adaptability to the environment in which they are located.

The problem of localization is a complex task, it is necessary to exploit the information from odorant dispersion and its components in order to design artificial olfactory systems. The artificial noses integration on a robotic platform along with the protocols designed for use thereof are the components that make up this kind of system. To create an integrated system we have designed an electronic nose model which allows the use of the heating strategies developed on autonomous systems with microcontrollers.

These designed systems exploits the environment information throughout the localization algorihtms that were developed. These algorithms are characterized by the use of differential signals in multisensor systems composed of artifical noses in a single robot. The algorithms verify the enviroment adaptability and obtain sucessfully results for both implemented systems.

Key words

Electronic nose, odorant localization, multisensor system, single robot strategy, sensory robotics, adaptative temperature modulation, closed loop control.

Agradecimientos

En primer lugar agradecer todo el apoyo recibido de mi tutor, Francisco de Borja, sin cuya guía, dedicación y compromiso no hubiera sido posible este proyecto. De igual manera agradecer tanto a Francisco como a Pablo Varona, por abrirme las puertas del GNB y ayudarme a desarrollar mi vocación investigadora.

A todos los miembros del GNB porque todos han aportado algo en este proyecto. A David y a Fernando por su paciencia y el valioso legado que dejan. A Carlos, Victor, Angel, Irene y Aarón porque con ellos siempre se pueden pasar buenos momentos y en general, a todos los miembros que han compartido y compartirán su tiempo en estas cuatro paredes.

Gracias a todos los compañeros de la eps sin los que estos años habrían sido mucho mas duros. Charlie, Dani, Escat y todos los demás con los que comparto aficiones y buenos ratos.

A Aura porque también es parte de este proyecto y a todos los amigos de siempre que quedan para toda la vida.

Y finalmente a mi familia, por permitirme siempre elegir mi camino.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

1.1. Motivación del proyecto

El estudio de la robótica sensorial y en particular centrado en el sentido del olfato es un campo emergente que desde el inicio del siglo XXI se ha situado en el punto de mira de numerosos investigadores de diferentes campos. El desarrollo de sensores electrónicos y la aplicación de los mismos en sistemas inteligentes puede resolver desafíos tales como la localización de fuentes de odorante en el espacio, el estudio de la distribución de gases en un entorno, la detección de plumas de odorante o la discriminación de los mismos tal y como se observa en últimos estudios realizados sobre el tema [4, 5, 2, 6, 7]. Pero la explotación en sistemas comerciales no es el único destino de la investigación pues el estudio del sentido del olfato nos abre una ventana al funcionamiento del mismo, ya que éste, es uno de los sentidos del que tenemos menos información y de los que más directamente se encuentra conectado con nuestro cerebro.

El Grupo de Neurocomputación Biológica(GNB) ha dedicado un área de investigación a los sistemas olfativos artificiales, el trabajo final de máster realizado por David Yáñez [8] otorgó las herramientas necesarias para comenzar la experimentación con los estímulos olfativos captados por los sensores TGS [9] que posteriormente ha dado como resultado publicaciones en este campo o relacionadas con él [1, 2, 10].

El anterior trabajo de Tomas Vázquez [11] sentó el inicio de los estudios de las aplicaciones de este tipo de sensores integrados en robótica. A diferencia de este trabajo, para nuestro problema diseñaremos un algoritmo de localización y utilizaremos una plataforma robótica más adecuada para la navegación en superficies sin obstáculos. Desde entonces, en el GNB se han planteado dos enfoques para afrontar el problema de la localización, la división de la tarea con varios robots utilizando lógicas de localización para una única nariz electrónica [12], y la vinculada a las estrategias multisensor de narices electrónicas integradas en una única plataforma que es el trabajo que presentamos a lo largo de esta memoria de proyecto.

En el campo de la localización de fuentes de odorante se proponen diversas soluciones bioinspiradas, algunos ejemplos son el vuelo de la polilla macho guiada por feromonas o el seguimiento de alimento de la langosta azul, ambos con resultados experimentales ya contrastados [13, 14]. Por otro lado las estrategias de explotación mediante los sistemas multisensor han resultado en algoritmos que otorgan mayor información acerca del entorno [15, 6].

La complejidad de la dinamica de fluidos hace que las variables del entorno en movimiento o la sensibilidad de los sensores sean variables a tener en cuenta y puedan caracterizar estrategias para el desarrollo de algoritmos. En este aspecto se implementarán metodologías de control para maximizar la información obtenida de los sensores aprovechando las propiedades dinámicas de los mismos.

A lo largo de este proyecto utilizaremos la teoría de control y el análisis y procesado de señales que junto a la integración de las narices artificiales en una plataforma robótica automatizada, crean un entorno multidisciplinario utilizando variedad de tecnologías y técnicas de distinto ámbito al servicio de la explotación de información obtenida del entorno.
Por lo tanto, este proyecto tiene una gran componente inicial de aprendizaje debido a este entorno multidisciplinar en el que se desarrolla así como la originalidad del campo de investigación que explora. Estos dos son factores que hacen de este proyecto la base para numerosos proyectos futuros con relación al uso de las narices electrónicas y su integración en plataformas robóticas.

1.2. Objetivos

En esta sección se plantearán los objetivos globales y las tareas concretas que guiaron el desarrollo de este proyecto.

1. Estudio y análisis del estado del arte respecto algoritmos de búsqueda de odorantes, estrategias bioinspiradas y entornos de dispersión de fluidos.
2. Utilización de narices olfativas Olus, proporcionadas por el GNB, para la aplicación de estrategias de captación, modulación y parametrización de los odorantes según percepción sensorial de la nariz.
    a) Estudio de los diferentes protocolos de comunicación con las narices electrónicas realizados en [8, 11].
    b) Instalación y uso de las librerías desarrolladas por Fernando Herrero [2] para la comunicación con el encapsulado Olus.
    c) Diseño de software de muestreo de las narices electrónicas.
3. Estudio de la dispersión de odorantes y parámetros a tener en cuenta en el diseño de los algoritmos de búsqueda.
    a) Modificación de calefacción de los sensores en ciclo cerrado mediante la aplicación de técnicas de la teoría de control.
    b) Caracterización de los sensores en el entorno de trabajo.
    c) Utilización de narices electrónicas para el estudio de los factores de dispersión. Influencia de distancias, odorantes y objetos externos en la dispersión de la fuente de odorante.
4. Integración en un plataforma robótica apropiada para la creación de un sistema completo de localización.
    a) Estudio previo de anteriores trabajos del GNB en integración [11].
    b) Selección de plataforma.
    c) Diseño e impresión de piezas físicas en tecnología 3D.
    d) Estudio de posibilidad de integración de estrategias multisensor.
    e) Algoritmos de localización en una y dos dimensiones.

5. Acondicionamiento de sensores comerciales modelo TGS [9] para aplicación de técnicas modulatorias de temperatura de calefacción en placas microcontroladoras y diseño de algoritmos de localización basados en el mismo.

   a) Acondicionamiento de calefacción de los sensores para la utilización de protocolos no definidos por el fabricante.

   b) Integración de los mismos en plataforma robótica para conformar un sistema sensorial completo.

6. Diseño de algoritmos de localización basados en técnicas de control y utilización de varias narices electrónicas simultaneamente en una única plataforma robótica.

7. Experimentos de localización.

   a) Demostración experimental de la efectividad del algoritmo desarrollado respecto el problema base en la localización de fuentes de odorante.

   b) Funcionamiento del sistema completo para la localización de fuentes de odorante.

1.3. Organización de la memoria

La memoria aquí presentada ha sido desarrollada partiendo de dos principios básicos:
Carácter evolutivo, partiendo de los modelos más sencillos hasta las estructuras completas del sistema. Las pruebas expuestas a menudo se realizan de forma incremental de forma que las mejoras del sistema se tienen en cuenta en la integración de nuevos componentes.
Modularidad, los sistemas desarrollados han sido divididos en partes para la asociación en bloques con características en común. Los sistemas finales son explicados en su totalidad en el capítulo 6, en el cuál se hace referencia a los demás componentes.

Los capítulos que conforman esta memoria serán los siguientes:

1. **Introducción.** Introducción al proyecto, motivación, objetivos y organización de la memoria. Es el capítulo en el que estamos actualmente.
2. **Estado del arte.** Estudio de las narices electrónicas, estado actual de los algoritmos de búsqueda y localización de fuentes de odorante e introducción a los técnicas de la teoría de control que hemos utilizados en diferentes aspectos a lo largo de este proyecto.
3. **Placa microcontroladora y plataforma robótica.** Se explicarán las características del microcontrolador utilizado en este proyecto para la integración del sistema olfativo artificial desarrollado y la plataforma robótica utilizada en la consecución del objetivo de localización de fuente. Resaltamos las características necesarias para la integración con nuestros dispositivos frente a las características que ofrecen. Finalmente se proporciona una guía de conexión para la integración final del sistema integrado.
4. **Caracterización de sensores y entorno de pruebas.** Debido a la complejidad de la dispersión de odorantes se iniciaron las estrategias desde el punto de vista del control de las narices electrónicas. En este capítulo se expondrán los experimentos previos con las técnicas de modificación de la temperatura que han dado lugar al ajuste final de los parámetros de muestreo en los sensores. En este capítulo también se detallarán algunos de los elementos que componen el entorno de pruebas sobre el que se desarrollan las estrategias de localización.
5. **Narices electrónicas: acondicionamiento para modulación de temperatura.** El acondicionamiento externo de los sensores para el particular control de temperatura de las técnicas de explotación de los sensores era uno de los objetivos principales en el desarrollo de este proyecto, en este capítulo se explicará el desarrollo de las narices electrónicas desarrolladas desde el punto de vista electrónico, se especificarán los circuitos finales de la nariz y las características principales que ofrece la nariz al sistema completo de localización.
6. **Sistemas olfativos artificiales desarrollados.** Que es un sistema olfativo artificial y los elementos que los componen. Dados los dos tipos de narices que se han utilizado en este proyecto se han desarrollado dos tipos de sistemas, aquí se detallaran los componentes de los mismos así como las principales diferencias y las motivaciones principales para la evolución hacia el sistema integrado.
7. **Integración, pruebas y resultados.** Explicación detallada de la evolución del algoritmo desarrollado para la localización de fuentes de odorante. Tipos de experimentos desarrollados en ambos sistemas y comparativa final de visualización de resultados estadísticos en los experimentos realizados de localización.
8. **Conclusiones y trabajo futuro.** Último capítulo de la memoria de este proyecto, conclusiones del trabajo realizado, consecución de objetivos y perspectivas de trabajo futuro para el desarrollo de este proyecto y otros derivados del mismo.

2

Estado del arte

2.1. Introducción

La localización de odorantes es el acto de encontrar la fuente de un determinado químico que se volatiliza en el medio. Este acto es y ha supuesto el medio de vida para muchos de los organismos que actualmente habitamos en la tierra. El estudio de las estrategias de localización aplicadas en plataformas robóticas tiene muchas aplicaciones posibles: aplicaciones industriales, como localización de fugas de químicos o eliminación de productos defectuosos en procesos de calidad; aplicaciones humanas tales como rescate de supervivientes en desastres naturales, localización de bombas, diagnóstico clínico, monitorización no intrusiva o estudios medioambientales de la calidad del aire.

Desde el punto de vista de la ingeniería, la herramienta artificial utilizada para la localización de odorantes es la denominada nariz electrónica, denominado de esta manera porque este artefacto es el encargado de la captación, análisis y explotación de los estímulos olfativos del entorno análogamente a las funciones sensoriales realizadas por las narices de los seres vivos. Es por esto que la implementación de una nariz electrónica artificial es un trabajo multidisciplinar que engloba los campos de la electrónica, teoría de control, computación e inteligencia artificial.

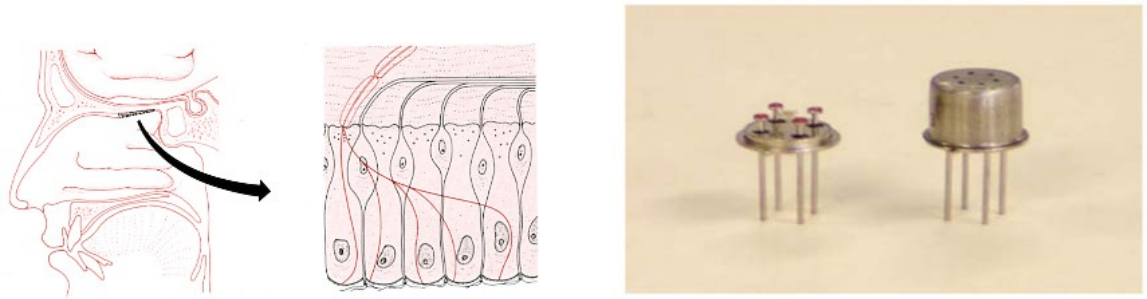
Las estrategias que mejor han funcionado en un campo de estudio incipiente son las creadas por la evolución de los seres vivos. El componente bioinspirado tiene una gran importancia a la hora de desarrollar algoritmos de búsqueda. La localización de la polilla macho buscando rastros de feromonas o las estrategias de búsqueda de nutrientes de la bacteria *Escherichia coli* como ejemplos, han inspirado multitud de estudios y algoritmos. Todos estos algoritmos y comportamientos biológicos pueden observarse con detalle en [16], cuya intención es aportar un amplio estudio del estado actual de la localización de odorantes.

El estudio del problema completo de localización se compone de tres fases principales: reconocimiento de la presencia de odorante, seguimiento de odorante y verificación de fuente. Para la creación de sistemas completos de localización es necesario contar con plataformas robóticas capaces de trasladarse por los distintos entornos en los que situamos el problema, en [4] podemos encontrar un amplio estudio de la robótica aplicada a este problema.

El trabajo del GNB del que parte este proyecto integró una nariz electrónica portátil en un robot modular [11]. Este tipo de nariz electrónica se ha utilizado en otros trabajos enfocados en la discriminación de fuentes [2, 1].
Exploraremos las técnicas utilizadas en los anteriores trabajos para aplicarlas en el estudio de la localización, entre ellas destacamos la aplicación de la teoría de control de la que hablaremos en la sección 2.4.1.

2.2. Narices Electrónicas

Para la explotación de los algoritmos de búsqueda es necesario tener un dispositivo olfativo artificial denominado comúnmente como nariz electrónica. El termino nariz electrónica es aplicado al conjunto de sensores y al hardware de acondicionamiento de los mismos que en conjunto con un software apropiado son capaces de imitar parcialmente las funciones y capacidades asociadas a nuestro sentido del olfato.
Este sentido en sistemas biológicos se articula mediante receptores sensoriales que responden ante sustancias químicas que llegan a través del aire. En humanos, los receptores se encuentran en el epitelio olfativo, figura 2.1a. Análogamente los sensores artificialmente diseñados para tal propósito poseen una superficie sobre la que las sustancias químicas excitan al sistema como observamos en la figura 2.1b.



(a) (b)

Figura 2.1: Representación del sentido olfativo en seres humanos en contraposición con los sensores que conforman las narices electrónicas. Sensor modelo TGS de la casa Fígaro [9].

Las tecnologías más comunes capaces de mimetizar los receptores olfativos de los seres vivos pueden clasificarse globalmente en: tecnologías ópticas, acústicas, espectrometría de masa, tensión superficial, nanotecnología y tecnología eléctrica. En [17] se desarrolla con gran detalle cada una de las tecnologías y los métodos que las componen.
Concretando el estudio nos centraremos en los sensores de tecnología eléctrica que por sus cualidades poseen una mejor integración en plataformas robóticas. Principalmente encontramos los siguientes tipo de sensores:

- **Quimiorresistores.** Sensores que varían sus propiedades de conductancia eléctrica ante la presencia de sustancias químicas. Tienen una baja selectividad por lo que la identificación de un único químico es un problema añadido pero son económicos y de tamaño reducido. Este es el tipo de sensores que utilizaremos [9].
- **Quimiotransistores.** Transistores con un recubrimiento que lo hacen sensibles a sustancias químicas. Pueden usarse en entornos líquidos y gaseosos.
- **Quimiocapacitores.** Sensores que varían su capacidad ante la absorción de partículas en el dieléctrico que lo compone.

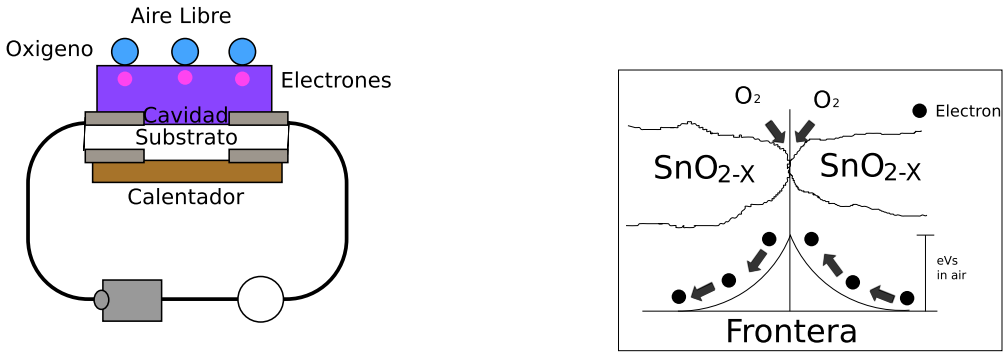
- **Quimiosensores amperimétricos.** Miden la corriente que pasa entre dos electrodos situados en una celda electroquímica. Son muy sensibles pero de coste elevado.

Se puede consultar bibliografía básica del funcionamiento del sentido del olfato, las técnicas utilizadas y los sensores que las componen en la referencia [18].

2.2.1. Sensores Quimiorresistivos

Es el sensor más utilizado en los estudios de la localización de odorantes debido a su versatilidad y su bajo tamaño y coste. Los sensores quimiorresistivos para narices artificiales son comúnmente de tipo MOS 8.2.3, son denominados de esta manera dadas las reacciones de oxido-reducción acontecidas en su interior. El principio de acción de los sensores será explicado de manera intuitiva a continuación:

1. Los sensores al aire libre de contaminantes (21 % de O_2) acumulan el oxigeno en la superficie del sensor. Ante la presencia del oxigeno los electrones libres de la superficie del sensor son atraídos creando un potencial de barrera (eVs para el aire). Por lo que el flujo de electrones se detiene y por lo tanto su resistividad es muy elevada. La abstracción de la figura 2.2 muestra este comportamiento.



(a) (b)

Figura 2.2: Sensores MOS expuestos al aire libre. Las partículas de oxígeno en el aire atraen los electrones de la cavidad y el potencial de barrear aumenta impidiendo el paso de corrientes pequeñas. Reproducido de [19].

2. Ante la presencia del gas contaminante las cargas del gas se oxidan eliminando el oxigeno de la superficie del sensor. Como resultado la densidad de oxigeno en la superficie se reduce y el potencial de barrera disminuye, por lo tanto, el flujo de electrones aumenta y la resistividad del sensor disminuye. Por lo tanto, la medición directa de la resistencia del sensor nos otorga información tanto del gas expuesto como de la concentración de partículas del mismo. Las reacciones químicas producidas pueden ser modificadas tanto por la temperatura de calentamiento de la placa interna transferida a la reacción como por los distintos componentes reactivos que se acumulen en la superficie del mismo. En la figura 2.3 observamos este comportamiento característico en comparación con la figura anterior del sistema.

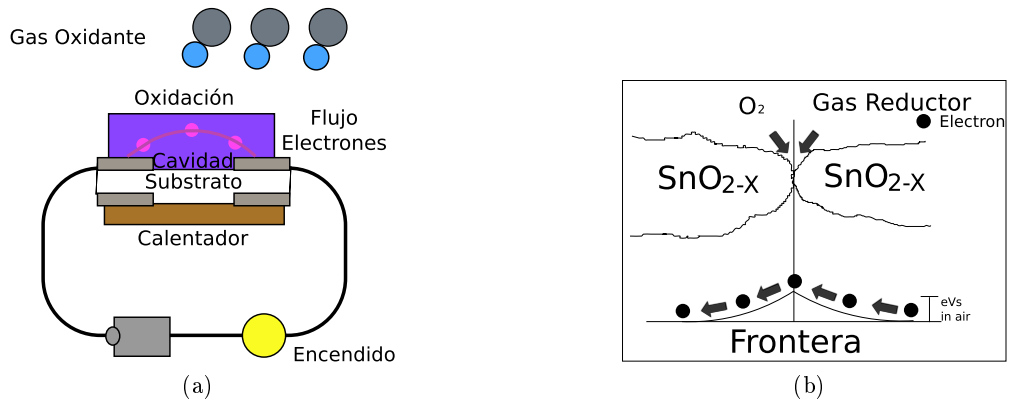


Figura 2.3: Sensores MOS expuestos a odorantes. Ante la presencia de un gas contaminante la superficie se oxida eliminando las partículas de oxigeno de la cavidad y aumentando el flujo de electrones. El flujo de electrones provoca una disminución del potencial de barrera permitiendo el paso de corriente. El comportamiento del sistema permite obtener en relación directa la concentración de partículas del medio de sustancias contaminantes a través de la medición de la conductividad del sensor. Reproducido de [19].

2.2.2. Dispersión de fluidos

Puesto que el objetivo es la localización de sustancias volátiles en el medio, el estudio de la dispersión de fluidos en entornos podrá otorgarnos del conocimiento previo necesario para entender algunos de los complejos mecanismos que suceden en la dispersión de odorantes.

Denominamos difusión como el proceso molecular mediante el cual se mezclan las partículas de dos orígenes diferentes. Una de ellas la podemos denominar fuente difusiva mientras que a la otra denominaremos medio en el que se propaga asignados mediante el volumen que representan del total. Denominamos dispersión al proceso difusivo de la fuente dispersiva cuando entran en juego otras variables externas que modifican el comportamiento propio tales como el viento, las turbulencias o las corrientes de convección.
La dispersión de fluidos es un tema complejo debido a la cantidad de factores externos que definen un medio y que modifican el comportamiento final del sistema. Pero además las variables de entorno como la densidad de partículas, la temperatura, la humedad o las turbulencias del sistema hacen que el proceso dispersivo no sea único dado un par fuente/medio. El estudio de tales procesos así como los modelos matemáticos que los rigen pueden conocerse en detalle en la bibliografía [20, 21].

Centrándonos en la aparición de los fenómenos naturales asociados a la dispersión podemos clasificar los entornos de explotación en tres según la bibliografía básica [16]:

1. Entornos dominados por la difusión, medios subterráneos en los cuales la densidad de partículas es muy alta.

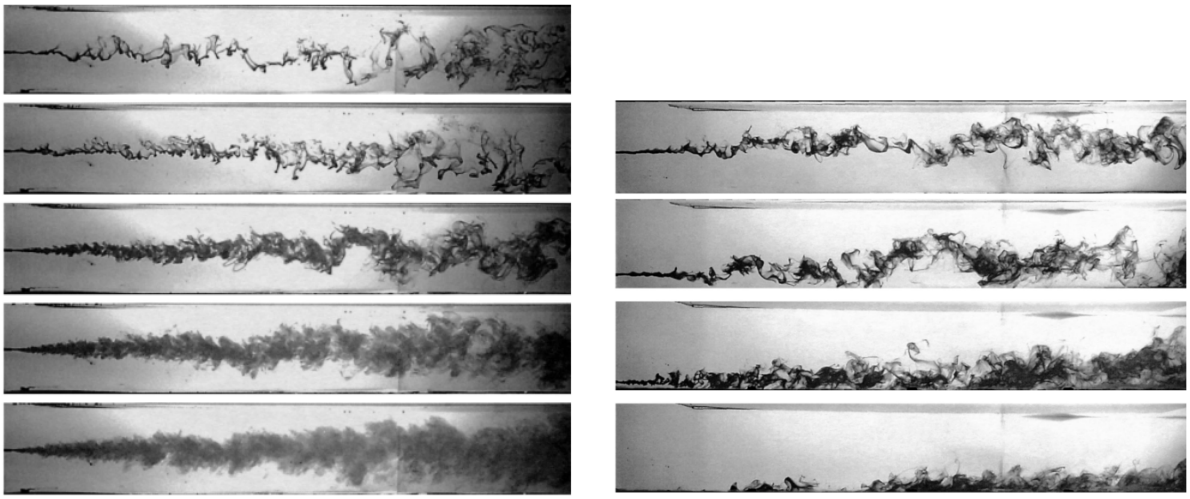
2. Entornos dominados por las turbulencias, medios cuyos parámetros externos como viento o cambios bruscos de temperatura caracterizan la dispersión. Estos últimos son los localizados al aire libre.

3. Entornos dominados por la turbulencia con velocidad de dispersión baja, podrían denominarse de esta forma los entornos en los que situamos nuestro problema. Son entornos cerrados en los que existen turbulencias de tipo remolino, denominadas *eddies*, que no tienen un comportamiento predominante con respecto la dispersión de la fuente de odorante.

Concretando en nuestro problema, obtendremos las conclusiones básicas que otros investigadores han podido demostrar [22, 23, 24, 25, 26] y que son aplicables a nuestro medio dispersivo.

- **Velocidad de escape.** Denominamos velocidad de escape como la velocidad en la que la fuente de odorante libera moléculas del fluido al sistema. Este factor es inicialmente intrínseco del odorante pero a su vez puede modificarse introduciendo mecanismos externos como un pulverizador o un ventilador.
  Se ha demostrado en [23] que a mayor velocidad de escape de un gas las perturbaciones del sistema tienden a diluir en mayor medida la pluma de odorante, o rastro de odorante, a lo largo de su trayectoria. Este comportamiento lo observamos en la figura 2.4a.

- **Altura de dispersión.** La altura a la que se coloca la fuente de odorante puede determinar el comportamiento general de la dispersión. La comparativa expuesta en [23] determina que en regiones de dispersión cercanas al suelo el odorante diluye de forma más homogénea. En casos extremos de distancias muy bajas la pluma de odorante tiende a diluirse en menor medida debido a la mayor concentración de odorante que queda atrapada en la capa más próxima al suelo. La imagen de este comportamiento es la denominada figura 2.4b.



(a) Velocidad de dispersión.

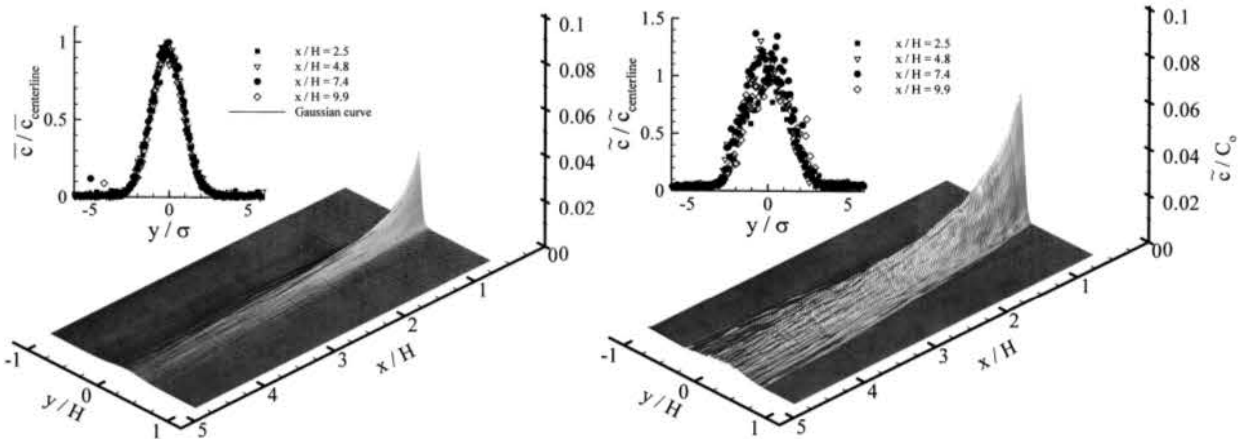
(b) Altura de fuente.

Figura 2.4: Figuras de dispersión de fuentes de odorante en fluidos. La figura a representa la dispersión de un odorante a diferentes velocidades, la velocidad se incrementa de arriba hacia abajo. Se puede observar como velocidades mayores crean mayores perturbaciones en el sistema y por lo tanto dispersión más homogénea. La figura b representa figuras de dispersión con distancia de la fuente al suelo como variable, de arriba a abajo se decrementa la distancia al suelo. En esta figura se puede observar como afecta la distancia al suelo en la dispersión y como distancias muy pequeñas pueden crear un pluma más estrecha al quedarse atrapada en la lamina de fluido más cercana al suelo. Imágenes tomadas de [23].

- **Distancia a la fuente.** El estudio de la figura 2.5 muestra el comportamiento complejo de la dispersión de la fuente a lo largo del eje bidimensional de la dispersión expuesta en la tercera ventana empezando por arriba de la figura 2.4b.

  En la figura 2.5a se analizaron los datos de distancia y concentración media normalizados. En el eje x, el coincidente con la dirección de dispersión de la fuente, los valores promedios de concentración decaen exponencialmente. En el eje y de la dispersión, los valores medidos configuran una función de gauss con el centro en el punto de dispersión de la fuente. Este comportamiento es el esperado para la dispersión de odorante con un comportamiento de dispersión unidimensional muy marcado como el utilizado tipo de fuente lineal de la sección 4.4.3.

  En la figura 2.5b se analizaron los valores de desviación de los promedios obtenidos en la ventana anterior, la función de gauss es más difusa para este caso. La aparición de picos de concentración es por lo tanto más probable cuanto más cerca de la fuente nos situemos. En este sentido puede verse remarcado como el entorno tiende a homogeneizar el odorante a largas distancias disminuyendo la aparición de diferencias de odorante dependientes de la distancia a la que nos situemos con respecto la fuente.



(a) (b)

Figura 2.5: Análisis estadístico de la dispersión con respecto la distancia a la fuente aproximada a una curva gaussiana. Aumento probabilístico de encuentros de mayor concentración promedio en distancias más cercanas a la fuente. Mayor variabilidad cuanto más cerca a la fuente nos encontremos lo que otorga información acerca de la homogeneidad de la dispersión a grandes distancias. Imágenes tomadas de [23].

2.3. Algoritmos de Búsqueda

Dada la complejidad de la dispersión de odorantes en los diferentes medios, las estrategias de búsqueda han evolucionado en diversas modalidades. La falta de capacidad humana para esta tarea hace que el entendimiento la misma se enfoque en el entorno bioinspirado de la naturaleza. En [27] podemos encontrar algunos de los retos y aplicaciones que actualmente se están considerando con relación a la explotación de algoritmos en el entorno del sentido olfativo artificial.

Hiroshi Ishida nos ofrece en [4] una revisión de las estrategias aplicadas en robótica y las tendencias actuales para la consecución del objetivo en diferentes medios de propagación.
En ocasiones, el objetivo de la localización unívoca no puede ser alcanzado debido a que el rastro es muy difuso como en una fuga subterránea o en una situación de rescate, es por esto que parte de la investigación se centra en el trazado de mapas de representación de odorantes. Respecto a este problema encontramos artículos con buenos resultados en estrategias de tipo cooperativo [28, 5], este tipo de estrategias nos permiten ser más eficientes al dividir la tarea principal coordinando varios robots móviles.

Otro tipo de problema es la simulación de las estrategias de localización, aunque la caracterización matemática de la dispersión hace que los modelos de simulación del entorno sean complejos se han obtenido resultados satisfactorios con resultados concluyentes en [29, 30, 31].

Estrategias bioinspiradas

En el esfuerzo conjunto de categorizar el comportamiento animal asociado a la búsqueda de objetivos mediante información sensorial olfativa, los investigadores han podido observar que es mayor la búsqueda sin estimulo olfativo que la búsqueda guiada.

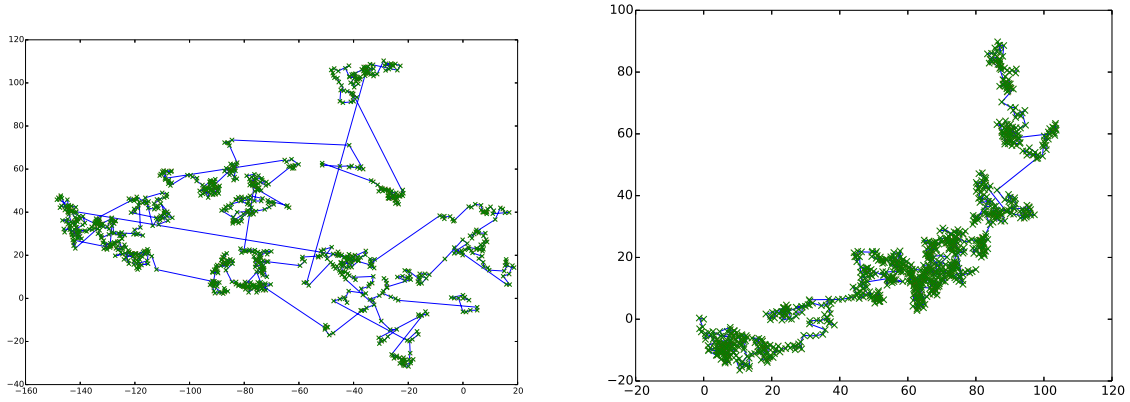
La falta de información sensorial no resulta en comportamientos repetitivos ya que los buscadores pronto abandonan las áreas sin rastros [32]. Las áreas de búsqueda en gran variedad de animales son mucho mayores a la distancia alcanzable sensorialmente. Memorizar las regiones exploradas se vuelve en un comportamiento poco eficiente e improbable. Estas son las premisas que conforman el paradigma de la búsqueda sin información en los seres vivos.
El estudio de estos modelos de comportamiento han concluido en modelos de búsqueda aleatoria denominamos comúnmente como *random walks*.

El modelo que presenta mejores resultados demostrados en [33, 34] es el denominado *levy flight* que ha sido observado en diversidad de especies en el reino animal. Está considerado el modelo más eficiente en la búsqueda sin información y por lo tanto una referencia para la localización de fuentes de odorante en la subtarea inicial de búsqueda de información.
Los sucesivos pasos de la estrategia de Levy son independientes y pueden calcularse con una función de distribución uniforme para la selección del angulo de rotación y un caso particular de la distribución de Pareto en la función de probabilidad que define la longitud del paso.

$$\phi \sim unif(0, 2\pi) \qquad \Theta_L(l) = (\alpha - 1) l_m^{\alpha-1} l^{-\alpha} \tag{2.1}$$

De la ecuación 2.1 el factor α es el parámetro de estabilidad mientras que el factor l_m representa el menor salto posible de la distribución. En la figura 2.6 se puede observar un recorrido realizado con este tipo de distribución.

Otros trabajos del GNB han demostrado la aplicación de los *levy flight* en [10] utilizando el enfoque cooperativo del enjambre de robots. Este tipo de estrategia podría resultar útil para un único robot como demuestra el comportamiento del depredador en el reino animal.



(a) (b)

Figura 2.6: Ejemplo de recorrido realizado mediante aproximación a la distribución 2.1 asociada con *levy flight*. Parámetro de estabilidad α modifica el comportamiento de los trayectos. Parámetro $\alpha = 1{,}5$ en figura a, $\alpha = 2{,}5$ en figura b. Densidad de color en representación representa mayor número de saltos en la región.

Chemotaxis

La estrategia denominada *chemotaxis* ha sido históricamente uno de los algoritmos más estudiados. Se basa en la medición sensorial en dos sitios separados espacialmente, las medidas tomadas representaran una señal gradiente que marcara la dirección del aumento de concentración y por consiguiente del movimiento en la búsqueda de la fuente de odorante. Lugar de mayor concentración obtenida.
Este tipo de estrategia ha sido observado en diferentes organismos como la langosta azul, el escarabajo pelotero o el microorganismo *E.coli*. En [35] se puede encontrar una comparativa de diferentes aproximaciones de este tipo de algoritmo con los sistemas biológicos de los que se inspira.

Infotaxis

Inspirado en la búsqueda de la polilla macho guiada por el rastro de feromonas de la hembra, existe una estrategia que actualmente demuestra resultados muy positivos en la localización [36, 37].
Este algoritmo se denomina *infotaxis* [14] y tiene como principio la creación de gradientes basados en la dirección que maximiza el número de encuentros con la pluma. A diferencia de *chemotaxis*, este tipo de algoritmos están preparados para ser explotados en entornos donde existen grandes turbulencias y donde la disponibilidad de información olfativa no es continua.
Actualmente se han contrastado los resultados procedentes de las simulaciones y experimentación robótica con las señales recibidas de las antenas de las polillas en un sistema biomecánico que demuestra la similitud entre el tipo de trayectorias del modelo y el mecanismo biológico real dirigiendo la plataforma robótica [38].

Estrategias Multisensor

Al igual que la naturaleza provee a los seres vivos de distintos sentidos sensoriales, las estrategias basadas en la integración de varios tipos de sensores para la explotación conjunta de los mismos es una evolución natural de los algoritmos.

En [39] queda demostrado la mejora en eficiencia de los algoritmos en entornos abiertos cuando se incluyen estrategias multisensor que integran un anemómetro dirigiendo el robot en contra del viento cuando se obtiene señal de las narices electrónicas.
Son muchos los estudios realizados con la integración tanto de anemómetros, o instrumentos de medición de variación en el flujo de partículas, como de sensores infrarrojos o dispositivos visuales que permiten el reconocimiento conjunto de la fuente en diferentes circunstancias [40, 41, 42].

Este tipo de estrategias también se han llevado acabo en menor número con la integración de varios sensores de odorantes. En [43] podemos observar un sistema con dos sensores olfativos independientes realizando una tarea de seguimiento de pluma con eficacia. Si nos centramos en la tarea propia de localización, [44] ha desarrollado un sistema en el cual se integran varios sensores para dirigir la plataforma robótica hacia la fuente.

Otro tipo de estrategias vinculadas a algoritmos evolutivos y genéticos han obtenido resultados muy positivos en este campo, la diferencia principal es que son capaces de adaptarse a las situaciones cambiantes del entorno debido al aprendizaje adquirido anteriormente. En [45] y [6] se pueden observar este tipo de modelos y la efectividad de los mismos.
En [46] se integran este tipo de algoritmos con estrategias bioinspiradas que modelan un sistema olfativo con dos sensores imitando las fosas nasales de la mayoría de mamíferos terrestres. Con esta integración son capaces de medir las respuestas diferenciales en diversos entornos para localizar la fuente de odorante con éxito.

2.3.1. Conclusión

Hemos podido analizar como los entornos en los que se explotan los algoritmos caracterizan el modelo asociado de dispersión. Las estrategias bioinspiradas son una gran herramienta gracias a la cual podemos realizar búsquedas más efectivas tanto con información sensorial como sin ella. Los algoritmos con esta componente tan marcada como *infotaxis* han demostrado gran versatilidad en entornos con grandes perturbaciones y una gran similitud con los sistemas biológicos.

Por otro lado las estrategias multisensor nos permiten la obtención de la máxima información sensorial del entorno. Son una estrategia a tener en cuenta en la tarea completa de localización de fuente en todas sus subtareas, desde la percepción de un químico en el ambiente hasta el reconocimiento de la fuente pasando por los modos de seguimiento de la pluma de odorante.

Los algoritmos evolutivos han demostrado que son capaces de explotar más información del sistema obtenida por las estrategias multisensor y adaptarla junto con componentes bioinspiradas para la consecución del objetivo de localización en gran variedad de entornos.

Los algoritmos que hemos desarrollado están enfocados en el control de los sensores para la explotación del entorno, esta perspectiva desarrolla una adaptabilidad que permite su uso como herramienta básica sobre la que se construye un sistema más complejo.
En nuestro caso utilizaremos una estrategia multisensor integrando varias narices electrónicas en un sistema simultaneo para obtener gradientes que dirigirán el sistema hacia la fuente como la estrategia en la que se basa *chemotaxis*.

La información obtenida de los sensores aplicando nuestros algoritmos podrá ser utilizada en conjunto con las estrategias estudiadas en esta sección para obtener decisiones más robustas en otro tipo de entornos. Esto queda referenciado en la sección 8.2 de trabajo futuro.

2.4. Teoría de control

La teoría de control es un campo interdisciplinar que se encarga del comportamiento de los sistemas dinámicos. La teoría de control y más específicamente el control automático es una parte fundamental de nuestra vida, la encontramos aplicada tanto en vehículos aeroespaciales como en los termostatos de nuestras casas entre otras muchas aplicaciones.
En particular los controladores PID son muy útiles para el control de un determinado parámetro en un sistema cambiante. Estos sistemas por lo general están expuestos a multitud de factores externos que la complejidad del análisis matemático se vuelve ineficiente o inexacto en su aplicación. Esta característica tan particular hace que se hayan convertido en la técnica de control más utilizada en la industria actual [47].

2.4.1. Controlador PID

El controlador PID es uno de los sistemas de control más utilizados, se define como un controlador en ciclo cerrado que modifica la variable de entrada al sistema a través de la obtención de una variable de error vinculada con la salida del mismo. Su funcionamiento se basa en el cálculo de una variable de ajuste del sistema que puede estar compuesta por tres componentes normalizados según las variables denominadas K_p, K_i y K_d respectivamente, la figura representantiva de estos factores es la 2.7:

- **Variable Proporcional.** Variable de error calculada como la diferencia entre la señal obtenida de la salida del sistema y el valor objetivo deseado para el cual estamos ejerciendo el control. Valores proporcionales muy bajos pueden resultar en falta de sensibilidad mientras que valores muy altos en situaciones de inestabilidad.
- **Variable Integrativa.** Integración temporal de los parámetros proporcionales del sistema a lo largo del tiempo. Este factor aumenta la variación en etapas iniciales en las que el error es muy grande pero a su vez puede llegar a producir oscilaciones en el sistema.
- **Variable Derivativa.** Derivación temporal inmediata del parámetro de error en el tiempo. Es una medida de la variación instantánea del sistema y por lo tanto su utilidad reside en la absorción de ruido aleatorio en el sistema así como la eliminación de la oscilación.

Existen tantas modificaciones del control PID para un determinado sistema como subconjunto de valores posibles aplicados que cumplen la funcionalidad del mismo. Además, dependiendo del comportamiento buscado puede prescindirse de alguno de las componentes.

El ajuste de parámetros K_p, K_i y K_d es experimental aunque existen diferentes configuraciones base sobre las que poder realizar un ajuste fino de los parámetros.

Ajuste de parámetros Ziegler-Nichols

El ajuste Ziegler-Nichols es un método heurístico para el ajuste de un controlador PID. Basa sus estudios en el comportamiento cruzado de los parámetros otorgando valores proporcionales entre si en función del comportamiento deseado.
La metodología es la siguiente: únicamente con el parámetro K_p, aumentamos el valor hasta la obtención de un valor K_u en el cual el sistema comienza a oscilar con una señal periódica de periodo T_u.
Estos dos parámetros son los que se utilizan en la tabla 2.1 para obtener los valores finales del controlador en función del tipo de control que queremos ejercer. Cabe destacar que esta metodología es de caracter orientativo. En nuestro problema se han aplicado inicialmente estos parámetros para posteriormente ser modificados realizando un ajuste fino en función del comportamiento esperado del sistema.

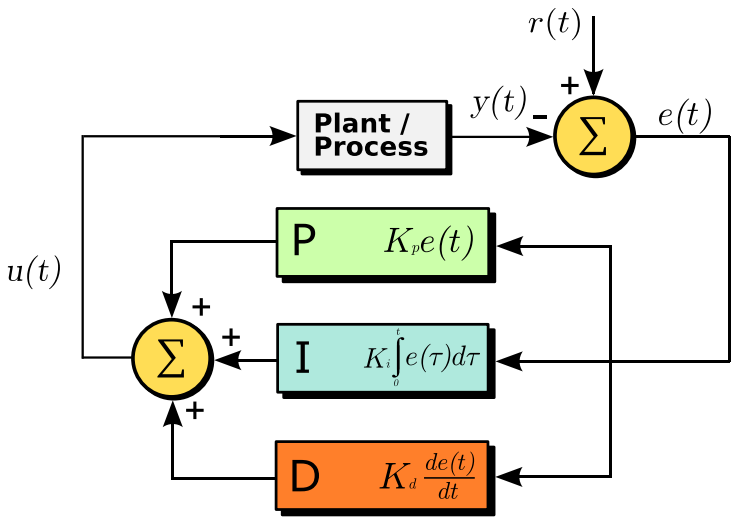


Figura 2.7: Imagen controlador PID. El bucle de realimentación modifica el parámetro de entrada al sistema mediante la obtención de la variable de error comparando la salida respecto el valor de referencia. Imagen obtenida de [48].

Tipo de Control	K_p	K_i	K_d
P	$0{,}5K_u$	-	-
PI*	$0{,}45K_u$	$1{,}2K_p/T_u$	-
PD*	$0{,}8K_u$	-	$K_pT_u/8$
PID Clásico	$0{,}6K_u$	$2K_pT_u/8$	$K_pT_u/8$
Regla Pessen	$0{,}7K_u$	$2{,}5K_pT_u/8$	$3K_pT_u/20$
Con Oscilación	$0{,}33K_u$	$2K_pT_u/8$	$K_pT_u/3$
Sin Oscilación	$0{,}2K_u$	$2K_pT_u/8$	$K_pT_u/3$

Cuadro 2.1: Tabla de configuraciones base para el ajuste del controlador PID según el método heurístico Ziegler-Nichols. Los parámetros marcados con * son los que han sido aplicados para nuestro sistema. Fuente [49].

En nuestro sistema se ha utilizado esta metodología para dos sistemas diferentes:

- Ajuste de un controlador de tipo PI para la modificación de la temperatura media en el algoritmo de búsqueda desarrollado de la sección 7.3.3.
- Ajuste de velocidad de movimiento a través de un controlador de tipo PD para la plataforma robótica. El valor de error será el diferencial de la distancia recorrida por cada una de las ruedas, el objetivo tenderá a que estas se igualen y por lo tanto la plataforma se traslade en linea recta. Referenciado en la sección 3.3.3.

3

Placa microcontroladora y plataforma robótica

3.1. Introducción

Los algoritmos de localización no son practicables sin una plataforma móvil que traslade el sistema sensorial hacia la fuente. Nuestro objetivo trata de implementar estrategias de localización y por tanto el diseño del robot seleccionado tendrá que ser de fácil integración con nuestro sistema para el envío de instrucciones a partir del algoritmo de búsqueda.

Seleccionamos la plataforma QuickBot B cuya característica principal es la de ser una plataforma totalmente libre diseñada por Rowland O'Flaherty con motivación educativa [50]. A este sistema se le han realizado algunas modificaciones para hospedar el sistema sensorial de las narices electrónicas.
El núcleo central será la placa microcontroladora Beaglebone Black, la cual en última instancia tendrá control sobre la plataforma robótica, el control sensorial y ademas será responsable autónomo de la toma de decisiones pues en él se ejecutaran los algoritmos de localización gracias a su potencia de procesado.

3.2. Placa microncontroladora Beaglebone Black

3.2.1. Características generales

Las placas Beaglebone son una familia de microcomputadores de hardware libre cuyas características principales son su baja potencia, sistema operativo embebido basado en Linux y su enfoque para las plataformas *open-source* útiles en entornos de investigación, educativos y en proyectos de bajo coste.
La ultima versión de esta familia es la denominada Beaglebone Black de la figura 3.1 en la cuál se presenta la integración de los últimos componentes para plataformas móviles y cuya potencia de procesamiento se sitúa por encima del estándar en computadores de sobremesa de hace 10 años.

Las características principales están expuestas en la tabla 3.2.1, pero cabe destacar para nuestro propósito el reducido volumen de la misma y la posibilidad de ejecutar código de alto nivel gracias a su sistema operativo embebido.

Facilidades para el usuario como los modos de alimentación del sistema, la tarjeta de red o el número de pines de sus puertos de expansión han hecho que este microcontrolador pueda utilizarse para multitud de aplicaciones en el ámbito de la electrónica.

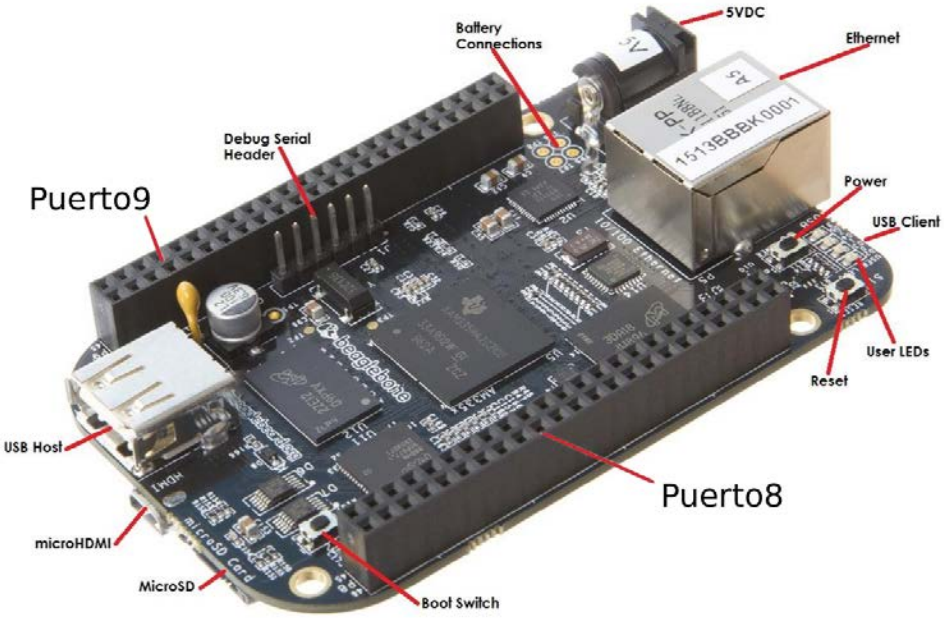


Figura 3.1: Tarjeta microcontroladora Beaglebone Black. Imagen de [3]

	Características
Procesador	Sitara AM3358BZCZ100 1GHz, 2000 MIPS
P. Gráfico	SGX530 3D, 20M Polygons/S
Memoria SDRAM	512MB DDR3L 800MHZ
Memoria Flash	4GB, 8bit Embedded MMC
PMIC	TPS65217C PMIC regulador y LDO adicional.
Soporte Debug	Opcional Onboard 20-pin CTI JTAG, Serial Header
Alimentación	miniUSB USB or DC jack || 5VDC Externo via Expansion Header
PCB	9cm x 5cm || 6 layers
Indicadores	1-Power, 2-Ethernet, 4-User LEDs controlables
HS USB 2.0 Cliente	Acceso a USB0, Modo cliente miniUSB
HS USB 2.0 Host	Acceso a USB1, Tipo A Socket, 500mA LS/FS/HS
Puerto Serie	UART0 acceso via 6 pin 3.3V TTL Header. Header is populated
Ethernet	10/100, RJ45
Conector SD/MMC	microSD , 3.3V
Entrada Usuario	Botón Reset,Boot y Power
Salida Video	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 1920x1080@24Hz w/EDID Support
Audio	Via HDMI Interface, Stereo
Conector Expansión	Power 5V, 3.3V , VDD-ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO (69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2), XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)
Peso	1.4 oz.(39.68 gramos)

Cuadro 3.1: Características generales de placa microcontroladora Beaglebone Black

3.2.2. Pines de entrada/salida

Los puertos de expansión son el método mediante el cual se pueden conectar electricamente el control de otros sistemas a la placa microcontroladora a través de sus pines. En nuestro caso, se utilizarán como medio de interconexión con la plataforma robótica y el sistema sensorial diseñado.
El detalle del mapeado de los modos en los que funciona cada uno de los pines se puede encontrar en el anexo B.2. En particular, aunque detallaremos más adelante las conexiones en detalle, los modos necesarios para ambos sistemas serán los siguientes:

- **Alimentación externa.** Pin del sistema para modo de alimentación externo a la placa microcontroladora. Este mecanismo nos permite dar movilidad a al microcontrolador sin cablear la alimentación de la placa a través de la entrada jack del sistema. En nuestro caso, alimentaremos el sistema a través del circuito de la rama de alimentación de la plataforma robótica.

- **Analógico.** Pines de entrada analógicos con resolución de 1 milivoltio. Entrada máxima al sistema a través de este tipo de pin de hasta 1,8 voltios lo que repercute en un acondicionamiento de señal adecuado. Las señales analógicas de entrada que utilizaremos serán las correspondientes de las narices electrónicas y codificadores de rueda de la plataforma robótica.

- **Digital.** Pines de entrada/salida digital. Valores digitales 0 o 1 coincidentes con 0 y 3,3 voltios a la salida. Utilizados para las señales de control de los circuitos integrados de la plataforma robótica y del sensor de humedad y temperatura externo al sistema de referencia (AM2302) visto en la sección 6.3.1.

- **PWM.** Pines de salida caracterizados por hacer uso de la técnica PWM mediante la cual se hace variar una señal digital a alta frecuencia. El resultado trata de simular una señal analógica mediante la variación del ciclo de subida en la señal periódica trasmitida, figura 3.2. Se utilizará este tipo de pin para regular tanto el calentamiento de las narices electrónicas como de los motores que trasladan la plataforma.

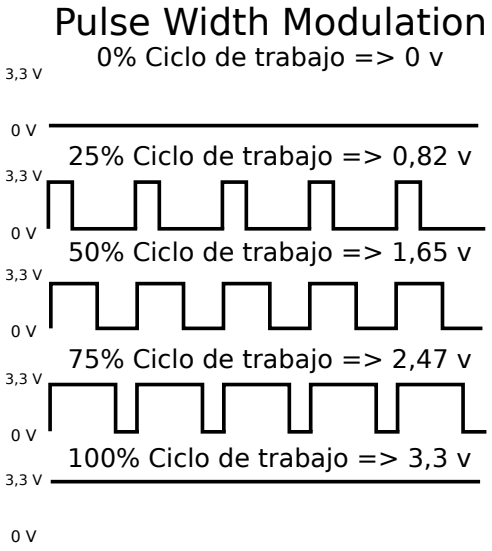


Figura 3.2: Ejemplo Pulse Width Modulation(PWM). La variación del ciclo de trabajo a alta frecuencia permite simular el voltaje constante de una señal analógica. Resolución de 0.1 puntos correspondiente a 3 milivoltios. Imagen modificada de [51]

3.2.3. Limitaciones de la aplicación en el sistema

Identificaremos claramente las limitaciones electrónicas de los pines de la placa microcontroladora para entender los acondicionamiento de señal que deberemos de realizar en nuestras señales procedentes de las narices electrónicas del sistema integrado.

	Sistema BBB		Necesarios en sistema completo	
Modo	No pines	Voltaje	No pines	Voltaje
PWM (O)	12	0-3,3(V)	6	0-5(V)
ANALOG (I)	7	0-1,8(V)	6	0-5(V)
DIG (I/O)	46	0 o 3,3(V)	5	0 o 3,3(V)

Cuadro 3.2: Limitaciones impuestas por nuestro sistema en la placa microcontroladora BBB

La limitación principal es la debida al PWM del sistema, su rango dinámico de salida se sitúa de 0 a 3,3 voltios mientras que el sistema de narices electrónicas artificiales que queremos desarrollar necesita un control de 0 a 5 voltios. Para ello nos valemos del circuito con transistor y con pin PWM como control de la base que hemos diseñado en la sección 5.3.2. Este circuito nos permitirá trasladar el rango de salida del sistema hasta los 5 voltios necesarios obtenidos de la rama de alimentación.
Para el caso de las entradas analógicas, se seleccionará adecuadamente el valor de la resistencia que permita alcanzar el rango de variación del sensor en el circuito divisor de tensión para la rama de muestreo de la figura 5.6.

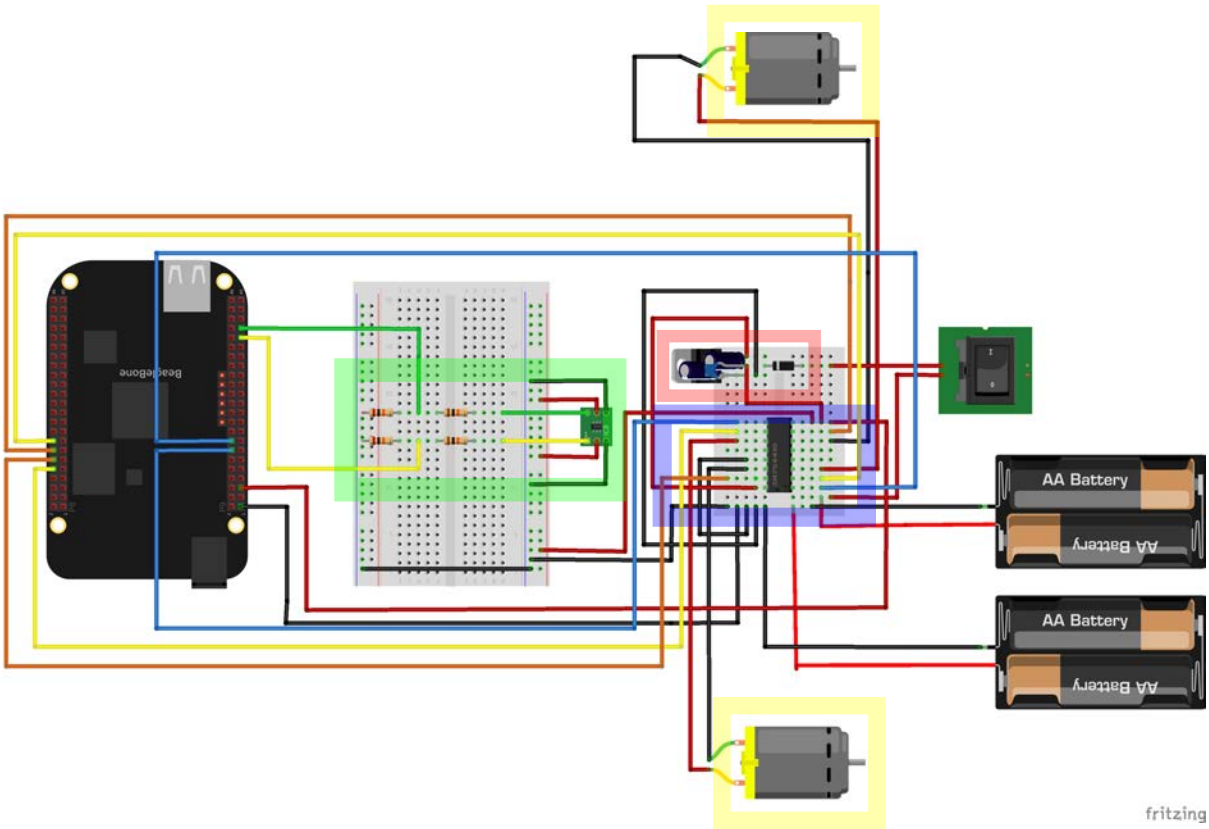


Figura 3.3: Conexionado plataforma robótica. Resaltado en rojo, el modulo de alimentación. Resaltado en azul, el puente en H o driver de potencia. Resaltado en verde, el codificador de las ruedas. Resaltado en amarillo, motores.

3.3. Plataforma robótica

La plataforma robótica será el medio mediante el cual trasladaremos los sistemas olfativos dirigidos por la toma de decisiones de los algoritmos. Se ha seleccionado una plataforma con dos ruedas direccionales y otra de apoyo omnidireccional para la consecución del objetivo. Al ser ambas ruedas independientes nos permite rotar la plataforma en su totalidad y realizar pequeños movimientos.

El detalle de la construcción y el paso a paso del ensamblado y conexionado de las módulos puede encontrarse en el apartado QuickBot MOOC v1 del siguiente enlace [50]. La lista de todos los componentes necesarios para la construcción de la plataforma está en la sección Anexo B.1. En esta sección, por tanto, resaltaremos la función de las partes fundamentales tanto hardware como software y las mejoras aplicadas a la plataforma para hospedar a las narices electrónicas.

3.3.1. Componentes

Los componentes electrónicos y módulos incluidos en la construcción de la plataforma robótica así como su conexionado con la placa microcontroladora están especificados en la figura 3.3. A continuación, detallaremos las partes más significativas.

Regulador de tensión

El circuito regulador de tensión del sistema está resaltado en rojo en la figura 3.3. Tiene como finalidad obtener una señal estable de salida que servirá de alimentación al microcontrolador y a los subsistemas de la plataforma robótica.

El encapsulado LD1085 de baja caída en su modelo TO-220 nos facilita esta función al otorgarnos señales estables de salida con un mínimo filtrado de 10μF. El circuito de la figura queda más detallado en el esquemático de la figura 3.4. En el se observan las señales V_{CC1} y V_{CC2}. V_{CC1} tendrá como destino la alimentación simétrica del driver de potencia. En el caso de V_{CC2} se conectará tanto en la alimentación simétrica del driver como en los demás subsistemas (BBB y codificadores de rueda) que demandan 5 voltios de tensión estables.
Para mayor detalle el regulador de tensión descrito llevara adherido un disipador de calor para el montaje TO-220 como medida de protección.

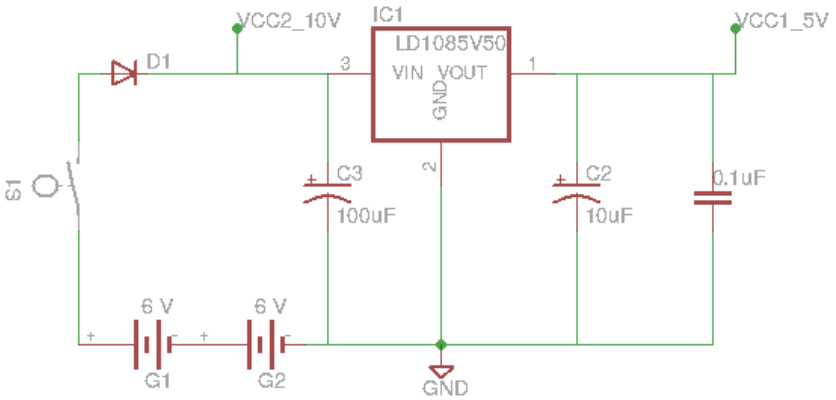
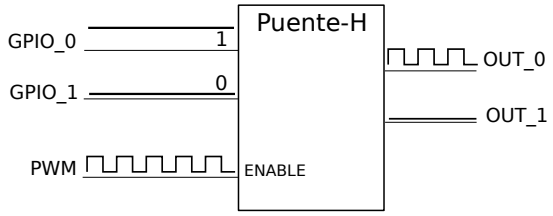


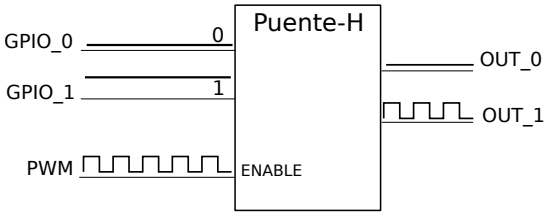
Figura 3.4: Regulador de tensión de la plataforma robótica. Entrada de voltaje de las baterías recargables a través de un pulsador y un diodo de protección. Filtrado de entrada y señal V_{CC2}. Filtrado de señal salida de 5 voltios (10 y 0.1 μF).

Driver de potencia

El driver de potencia también llamado puente-H permite suministrar la potencia necesaria en los motores para el movimiento mecánico. El control preciso de la velocidad es un difícil objetivo que depende no solo del control eléctrico sino también de la calidad de los motores y del circuito de alimentación.
El driver que hemos utilizado es el componente SN754410 manufacturado por Texas Instruments y resaltado en el circuito de conexiones en azul, figura 3.3. Es un circuito cuádruple con dos entradas habilitadoras.
En el circuito de la figura 3.5 queda representado los dos comportamientos que permitiremos en el driver de potencia del circuito. Existe una señal enable cada dos entradas que serán habilitadas por la señal PWM. De esta forma modulamos el pulso de salida y controlamos la potencia suministrada a los motores.



(a) Señal salida modulada 0-1



(b) Señal salida modulada 1-0

Figura 3.5: El enable del circuito maneja dos entradas, la salida será la entrada cuando el enable este activo, en cualquier otro caso la señal de salida son 0 voltios, de esta manera la señal de salida es modulada suministrando al circuito receptor la potencia necesaria. Dos comportamiento para un único motor que maneja una rueda de la plataforma.

El circuito SN754410 es un circuito cuadruple, esto nos permite tener control del sentido de ambos motores. El control se realiza activando un pin u otro a la entrada del circuito dependiendo del sentido del giro, en la salida del circuito la señal de entrada a los motores obtendrá la señal de potencia diferencial modulada alternativamente en función de la entrada. El motor utilizado es DG01D-A130GEARMOTOR del fabricante DAGU. El circuito de la figura 3.7 referencia el conexionado final de los motores.

En la tabla adjunta 3.3 están los valores representativos del driver de potencia obtenidos del fabricante. Si comparamos con la tabla 3.2 de limitaciones de la placa microcontroladora podemos asegurarnos de su correcto funcionamiento a través de los pines digitales y PWMs del sistema.

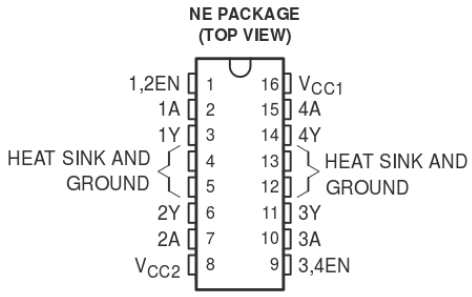


Figura 3.6: Esquemático de conexión SN754410.Procedente del datasheet, Texas Instruments.

Variable	Valor
V_{inHIGH}	2 - 5,5 (V)
V_{inLOW}	-0,3 - 0,8 (V)
V_{CC1}	4,5 - 5,5 (V)
V_{CC2}	4,5 - 36 (V)
I_O	$\pm$ 1 (A)
V_O	-3 - V_{CC2}+3 (V)

Cuadro 3.3: Tabla limitaciones circuito integrado SN754410. Comprobado para BBB en 3.2.

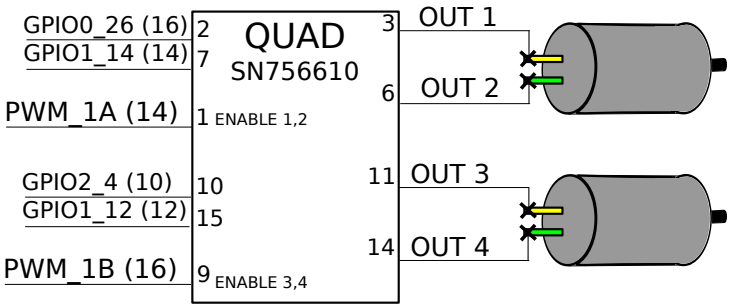


Figura 3.7: Conexionado final driver de potencia en motores de la plataforma robótica. Números de pines referencia del integrado a la figura 3.6. Sentido de giro por pin digital, velocidad en ciclo de trabajo del PWM.

Codificadores de ruedas

El codificador de rueda es un circuito que permite, mediante la integración de sensores infrarrojos, conocer la posición de las ruedas del sistema. Obtener un *feedback* sensorial del movimiento es de vital importancia en los sistemas móviles ya que por una parte, necesitamos conocer tanto la distancia recorrida como la velocidad del movimiento; y por otro lado, los sistemas compuestos de motores independientes tienden a introducir errores observables en el recorrido en linea recta de distancias largas. Este efecto es debido a la demanda eléctrica irregular de los motores.

El codificador de rueda, resaltado en la figura 3.3 en color verde, está compuesto por el circuito eléctrico integrado, el acondicionamiento de señal y la pieza codificadora que va insertada en el motor.

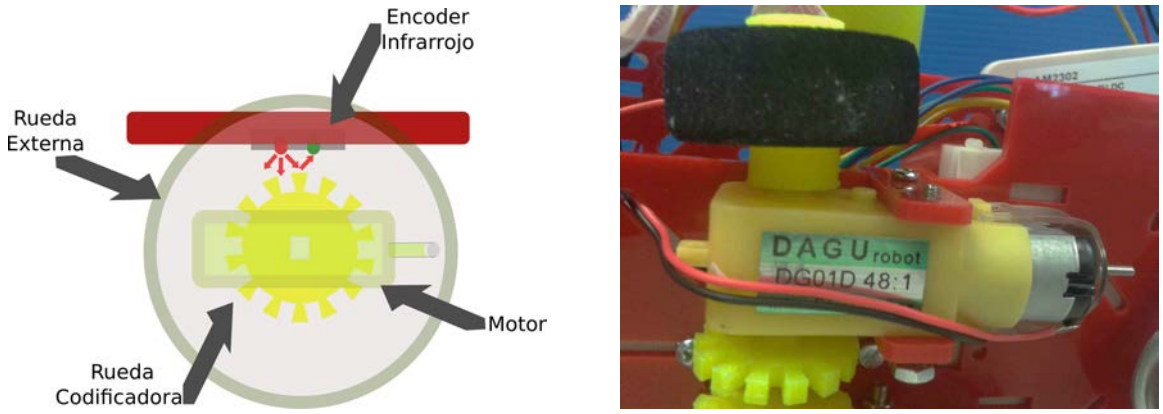
- El circuito eléctrico integrado consta de dos módulos que corresponden a cada una de las ruedas direccionales de la plataforma. Cada uno de los módulos consta de un sensor emisor y otro receptor de referencia QRE1113. El emisor emite señales infrarrojas que serán recibidas por el receptor. Los valores de voltaje, que posteriormente leeremos desde el pin analógico del sistema, tendrán dependencia directa con la distancia del objeto que hace reflejar la onda emisora.

- El acondicionamiento del circuito mediante un divisor de tensión de la alimentación de 5 voltios respecto una red de resistencia traslada la señal del circuito integrado al rango de entrada de nuestro sensor analógico (1,8 V).

- La pieza codificadora va insertada en los motores y es la parte mecánica que permite el funcionamiento del sistema. El sensor infrarrojo emitirá una señal que será reflejada en uno de los dientes de este codificador o en uno de los espacios interdentales. De esta forma, podemos discriminar cuando la rueda ha girado en un porcentaje de giro cuya resolución viene dada por el número de dientes de esta pieza codificadora.

La figura 3.8 muestra como se realiza la integración del codificador completo. Para ello la plataforma física posee unas aberturas especialmente diseñadas para estos codificadores de rueda. Este tipo de codificador tiene resolución de 16 pasos por vuelta, es decir que únicamente podemos medir el movimiento de la rueda en 16 partes la longitud de la circunferencia descrita por la rueda física. Por lo tanto, y como se explicará más adelante, puesto que los codificadores van adheridos al motor y no a la rueda se puede insertar ruedas más pequeñas para aumentar la resolución de la misma. El software de lectura de este componente se encuentra detallado en la función *countTics* de la sección C.1.1.

El diseñador de la plataforma [50], seleccionó este tipo de codificadores por sencillez. Cabe resaltar, que existen otro tipo de codificadores más complejos con mayor resolución y que codifican más información como el sentido de giro de cada rueda.



(a) (b)

Figura 3.8: Integración de las piezas del codificador de rueda para el *feedback* de la señal movimiento. La plataforma consta de dos módulos independientes de este tipo para cada uno de las ruedas direccionales. Figura a representación desde el lateral. Figura b visto desde la planta.

3.3.2. Partes impresas

Con objetivo la obtención de un sistema completo integrado y fiable se han añadido o retocado ciertas partes físicas de la plataforma robótica con diseño de estructuras 3D.
La impresión tiene tres pasos:

1. **Diseño**. En nuestro caso, la herramienta *opensource* OpenScad permite realizar diseño de estructuras de forma programática mediante la creación de polígonos, traslación de ejes o rotación de ángulos y operaciones sobre las estructuras como la unión o la diferencia entre ellas.

2. **Creación stl.** Ampliamente conocido como el formato estándar en la representación geométrica de estructuras. Este archivo únicamente contiene los valores geométricos y en ningún caso el escalado u otro tipo de atributos. Al ser un formato estándar el diseño de las piezas puede realizarse con cualquier herramienta de diseño tipo CAD (SolidWork, FreeCad).

3. **Impresión.** El último paso de la impresión mediante el cual se configuran los parámetros de dimensionado y de detalle de la pieza en la impresora 3D. Finalmente se ejecuta el proceso en la impresora 3D que imprimirá la pieza capa a capa como observamos en la figura 3.9.

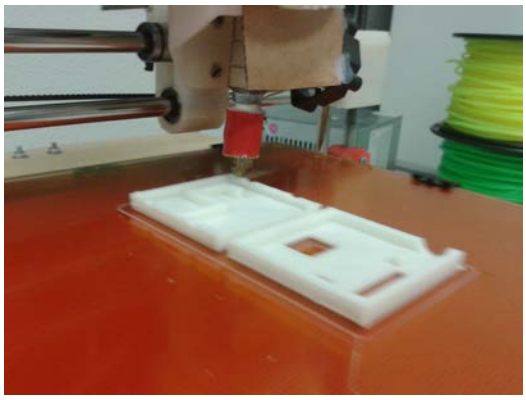
Expondremos continuación el listado de las piezas. Todas ellas han sido realizadas con plástico tipo PLA.

- **Caja protectora Olus.** Pieza protectora del encapsulado que permite la adhesión en cualquier superficie.

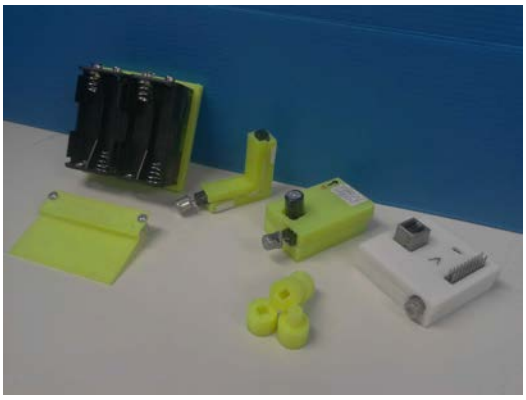
- **Caja protectora narices artificiales v1.** Pieza protectora de las narices integradas pertenecientes a la versión 1.

- **Caja protectora narices artificiales v2.** Pieza protectora de las narices integradas pertenecientes a la versión 2. Código en el anexo D.0.3.

- **Ejes de ruedas.** Se integra en la plataforma ruedas de diámetro más pequeño para aumentar la resolución. El eje de los motores tiene una forma poco estándar con lo que se diseño esta pieza para fijar las ruedas al eje del movimiento. Código en el anexo D.0.4.
- **Extensores de plataforma.** Para la correcta adhesión de las narices se diseñaron estos soportes adheridos a la plataforma robótica mediante tornillos.
- **Soporte de pilas.** Con objeto la estabilidad de la plataforma se diseño este soporte para encajar las pilas y que no provocaran tensión en el cableado anexo a ellas. Código en el anexo D.0.5.
- **Encoder de rueda.** Debido a un problema durante la ejecución de las pruebas de localización, las ruedas codificadoras originales de la plataforma se deterioraron. Con este motivo, se diseño una pieza similar.
- **Otras piezas.** En la consecución de algunos experimentos se han realizado el diseño de otras piezas como la plataforma fija que sustituía a la móvil inicialmente o piezas extensoras situadas en los sensores para observar la mejora obtenida.



(a)



(b)

Figura 3.9: Imagen de la impresora 3D realizando la impresión de las piezas externas adheridas a la plataforma robótica. Piezas impresas de arriba a abajo e izquierda a derecha: soporte de pilas, caja protectora nariz v1, caja protectora nariz v2, caja protectora Olus, extensor de plataforma y ejes de las ruedas.

3.3.3. Software

Una vez tenemos la plataforma construida y conectada se prepara el código de control de los módulos integrados. Al tratarse de un microcontrolador con sistema operativo embebido nos permite programar en alto nivel, la ventaja fundamental es el elaborar lógicas más complejas de control sin tener en cuenta los detalles de comunicación con los periféricos de la placa.
En este caso hemos programado en lenguaje de programación Python. Las librerías de comunicación para el control de los pines y sus modos en la placa BBB han sido descargadas en el siguiente enlace [52]. El código de control de los motores ha sido desarrollado basado en el código de Rowland O'Flaherty [50].

Se ha optado por un diseño software formado por dos hilos como se observa en la figura 3.10, los hilos del programa ejecutarán código especializado en una única tarea del sistema. En el caso de la función adjustSpeed, la figura 3.11a muestra el diagrama de flujo del codigo. La medición de distancias a través de las ruedas, función measureDistance, en la figura 3.11b.
Explicaremos a continuación cada uno de los módulos que conforman el codigo del sistema del anexo C.1.1.

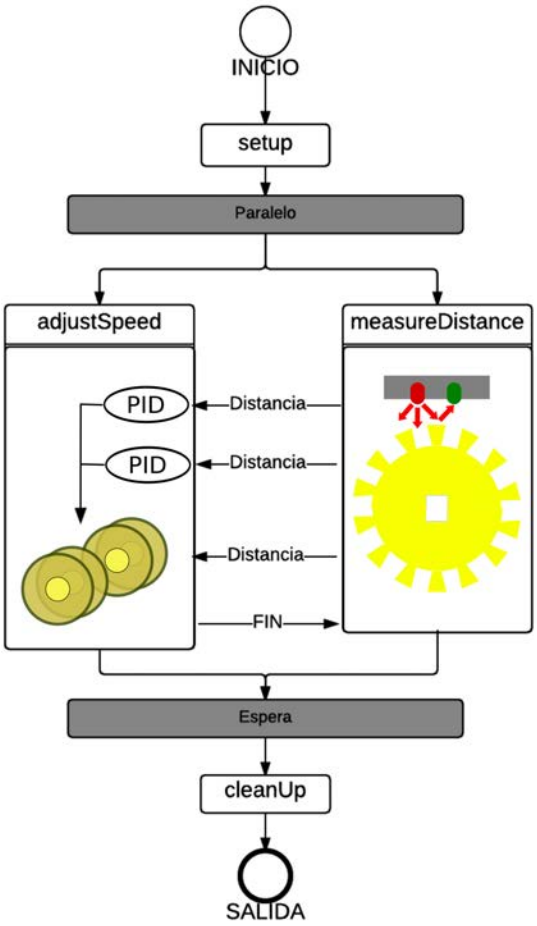
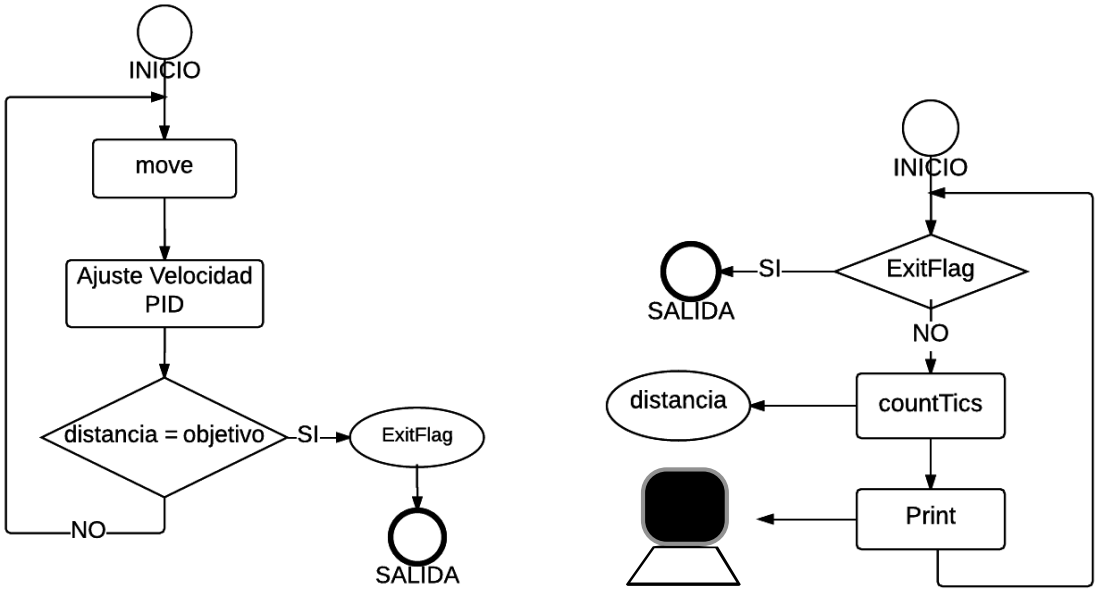


Figura 3.10: Diagrama de flujo del código principal de movimiento de la plataforma robótica. Dos grandes bloques resaltados cuyos nombres hacen referencia a las funciones del código que representan en el código del anexo C.1.1, el dedicado al control y ajuste de la velocidad de las ruedas, y el dedicado a la medición de los codificadores de rueda y distancias recorridas.



(a) (b)

Figura 3.11: Diagrama de flujo de los bloques del software que compone el movimiento de la plataforma. En la figura a queda representando el bloque de ajuste de velocidad de la plataforma, en la figura b se observa el diagrama de flujo del bloque de medición de distancia. El paso de parámetros entre bloques queda representado en el diagrama con una elipse.

- **setup.** Inicialización del sistema. Función encargada de invocar a las funciones de configuración de cada uno de los pines a utilizar en el sistema.
- **cleanup.** Finalización del sistema. Puesta a 0 de los motores y finalización en el control de los pines de conexión con la plataforma.
- **move.** Rutina encargada de asignar el valor de entrada en el ciclo de trabajo de los pines de PWM en el sentido correcto del movimiento. Linea 83 moveRobot.py del anexo C.1.1.
- **adjustSpeed.** Función encargada de realizar modificaciones en la velocidad. A través de un controlador PID(PD) se introduce una variable de ajuste que variará la velocidad en función de la distancia recorrida por las ruedas. La variable de error proporcional(e_p) es la diferencia actual de las distancias recorridas de las ruedas, el parámetro perseguido sobre el que se diferencia puede variarse en función de el tipo de curva que queremos que describa la plataforma en su movimiento.
  En este caso, únicamente se ha realizado con el objetivo de 0; es decir, con movimiento rectilíneo que debido a las desviaciones eléctricas de los motores ha resultado de gran utilidad en el recorrido de largas distancias. Linea 166 moveRobot.py del anexo C.1.1.
- **countTics.** Función de muestreo de los sensores infrarrojos. Para discernir cuando avanzamos en el movimiento. Para ello se parametrizan valores umbral de tensión sobre los que confirma el reflejo infrarrojo de lo que llamamos tic (medida de división de la rueda) y valores de repetición para filtrar el ruido en el sistema. Linea 116 moveRobot.py del anexo C.1.1.
- **measureDistance.** Rutina por encima de countTics, encargada de acumular los tics y medir las distancias. Esta función posee el testigo global de todas las demás funciones para ordenar la finalización al alcanzar la distancia objetivo. Linea 122 moveRobot.py del anexo C.1.1.

Existen dos tipos de movimiento de la plataforma denominados como *moveRobot* y *turnRobot*. Los scripts son similares con la diferencia principal que en el caso del segundo las ruedas se mueven en sentido contrario entre ellas creando una rotación sobre su eje. El controlador PID(PD) del ajuste de velocidad únicamente está implementado en la función *moveRobot*. El valor de entrada serán los grados del giro descrito por la plataforma. Para esto se calcula la longitud de la circunferencia que describen ambas ruedas y se calcula el porcentaje de esta longitud que debe de trasladarse la plataforma para describir el angulo introducido. En este caso, no existe el ajuste automático de velocidad.

3.4. Conexionado final del sistema

Uno de los objetivos del proyecto era la extracción del acondicionamiento de los sensores de forma que el microprocesador de la plataforma robótica pudiese integrar el sistema de narices artificiales. En este capítulo se ha detallado la construcción, conexionado y ensamblado de la plataforma robótica.
Finalmente, incluiremos la figura 3.12 en la cual se detalla el conexionado final de la tarjeta microcontroladora BBB con el sistema final integrado a modo de guía de conexión. Los sistemas que hemos integrado en la plataforma son dos: narices artificiales versión v2, cuya evolución está explicada en el capítulo 5; y sensor de humedad y temperatura, referenciado en la sección 6.3.1.

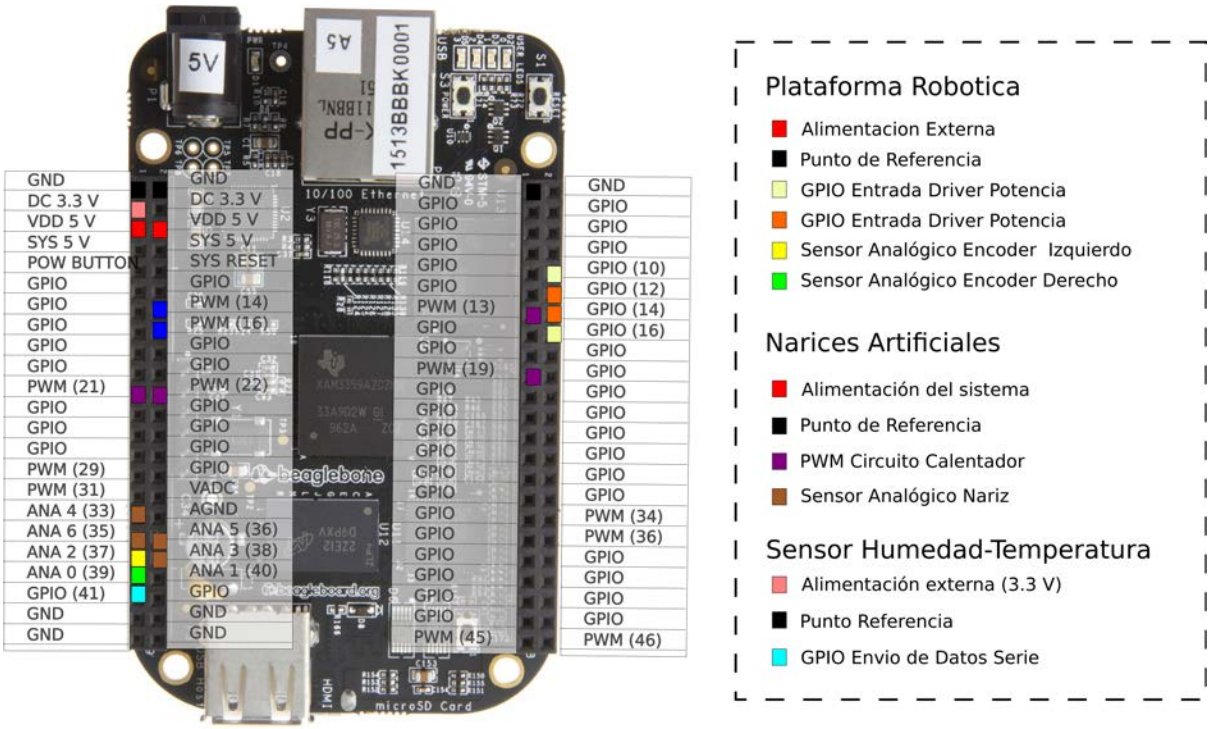


Figura 3.12: Conexión de pines BBB en el sistema final integrado. Conexión con el control de la plataforma robótica, cuatro de las narices artificiales y sensor externo de humedad y temperatura.

4

Caracterización de Sensores y entorno de pruebas.

4.1. Introducción

El siguiente capítulo tratará de argumentar los pasos iniciales en los que se cimientan los algoritmos desarrollados en el capítulo 7. La ordenación del mismo posee un carácter evolutivo mediante el cual las mejoras en las mediciones mediante control se ven reflejados en los posteriores pasos.
Dado el carácter complejo de los sensores de detección de odorantes y la limitada experiencia de estos sistemas en aplicaciones tanto comerciales como dentro del ámbito de la investigación, el primer paso para la puesta en explotación de los algoritmos será la de conocer las herramientas con las que contamos para su control. De igual forma las mediciones iniciales de las concentraciones medidas nos darán un punto de partida para valorar las situaciones posteriores y conocer el ámbito de la dispersión de odorantes a través de la perspectiva del sensor.

Comenzaremos estudiando las características del sensor para dar paso al estudio de los métodos de control de los sensores y finalizando con la caracterización del entorno de pruebas sobre el que correrán los algoritmos.

4.2. Sensores

Los sensores tipo MOS que anteriormente hemos detallado, sección 2.2, obtienen el valor de concentración del odorante a medir mediante la inversa de su conductividad. Los principios de diseño y funcionamiento de los mismos están detalladamente expuestos en la literatura [53, 18] y en la sección 2.2. La medición obtenida es característica del odorante, pero es a su vez resultado de las condiciones previas a la exposición, el único factor sobre el que podemos ejercer pleno control será la temperatura de calefacción de su placa interna y nos valdremos de ésta para proporcionar diferentes comportamientos a los sensores vinculados a la sensibilidad adquirida.

Es esta sección, estudiaremos de forma experimental los principios básicos de funcionamiento de los sensores, los factores a tener en cuenta para las mediciones obtenidas y determinaremos la temperatura idónea de calentamiento sobre la que nuestros sensores ejecutarán eficientemente los algoritmos desarrollados. La denominada ventana de temperatura óptima que definiremos a continuación.

4.2.1. Ventana de temperatura

La ventana de temperatura se define como el régimen dinámico del sensor sobre el que aparecen ciertos comportamientos deseables en el sistema. La obtención de la denominada temperatura será el objetivo de esta sección.

La temperatura en el sensor, repercusión directa del voltaje aplicado en las bornas del calentador, añade complejidad a nuestro sistema. No solo modifica la sensibilidad de los sensores, como posteriormente demostraremos, si no que también es capaz de inferir diferentes dinámicas; podemos por tanto realizar cierta analogía entre el acondicionamiento de la temperatura de calefacción del sensor y la virtualización de sensores con distintos comportamientos característicos.
Cabe destacar que los sensores de fábrica están diseñados para su utilización con voltajes de calefacción de 5 voltios como máximo, y mientras no se recomienda superar este umbral, no especifica nada del tipo de comportamiento esperado en los límites inferiores. Nuestra hipótesis basada en experimentos previos documentados en [2, 54, 55, 56, 57], parten de la base que la información transitoria obtenida a voltajes de calefacción inferiores a los recomendados otorgan valiosa información para las tareas de diferenciación y localización de odorantes entre otras.

A lo largo de esta sección vamos a comparar y caracterizar los dos sistemas que hemos desarrollado en el trascurso del proyecto. A modo de ejemplo, las imágenes de las ventanas a,c y e de la figura 4.2 corresponden al sistema Olus, es el punto de partida de nuestro estudio en la caracterización y estudio de los sensores para el diseño de los protocolos de utilización en los sistemas; las imágenes de las ventanas b,d y f corresponden al sistema final integrado, el cual utilizamos en las pruebas definitivas del algoritmo bidimensional. El detalle de las narices electrónicas utilizadas en estos experimentos se encuentran en el capítulo 5 aunque adelantaremos que en todos los experimentos aquí expuestos el sensor utilizado será el modelo TGS2611 del fabricante Fígaro [9].

Cuando iniciamos el sistema y ponemos un determinado voltaje en el calentador lo que encontramos al muestrear la señal es lo que observamos en la figura 4.1. La curva de los sensores tiene una tendencia a la alza que se corresponde con el estado inicial en el que los sensores buscan la estabilidad del sistema en el entorno de exposición. Definimos estabilidad en las señales del sistema, como aquellos valores muestrales que se mantienen en el mismo promedio durante un número consecutivo de muestras definidas por nuestra ventana muestral.

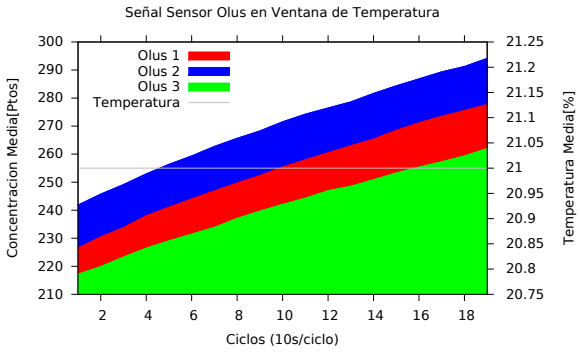
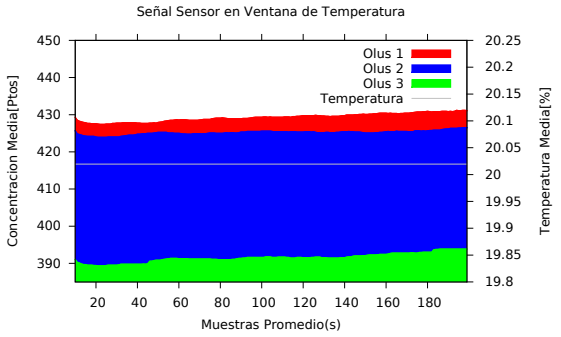
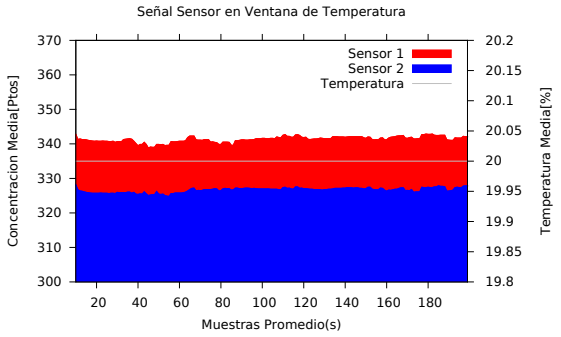


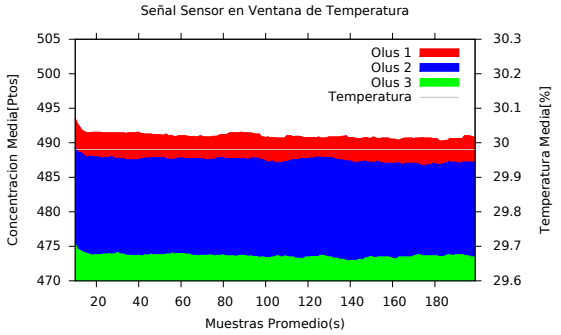
Figura 4.1: Sistema Olus, sensor TGS2611. Inicio de la experimentación con sensores. Muestreo de calentamiento de la señal del sistema Olus. Calentamiento del sistema fijo a una temperatura del 21 % del máximo.



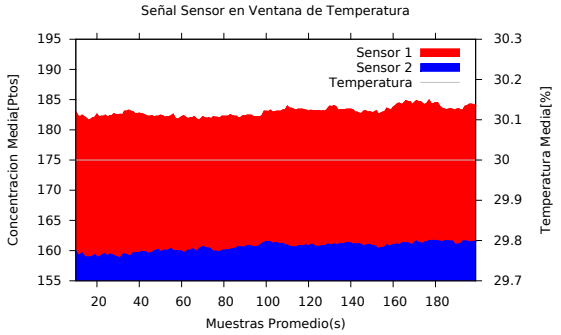
(a) Sistema Olus 20 %.



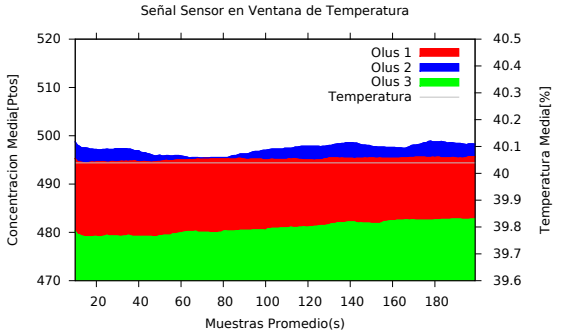
(b) Sistema integrado 20 %.



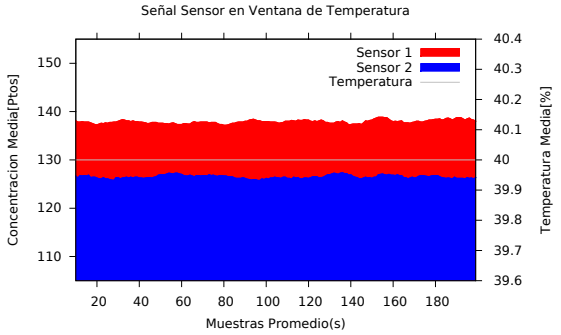
(c) Sistema Olus 30 %.



(d) Sistema integrado 30 %.



(e) Sistema Olus 40 %.



(f) Sistema integrado 40 %.

Figura 4.2: Sistema Olus y sistema integrado, sensores TGS2611, sin exposición de odorante. Estudio de temperaturas de calefacción en los diferentes sistemas. Temperaturas de calefacción entorno al 20, 30 y 40 % del máximo recomendado por el fabricante (5v). Demostración de estabilidad en ambos sistemas con el protocolo inicial. Figuras a, b y c pertenecientes al sistema inicial Olus. Figuras d, e y f pertenecientes a las narices electrónicas del sistema integrado versión 2 de la sección 5.3.

Inicialmente, se pensó que la estabilidad era dependiente de la temperatura a la que calentábamos los sensores como observamos en la figura inicial 4.1, posteriormente y tras el desarrollo del sistema integrado, observamos que muestras estables a bajas temperaturas eran posibles si calentabamos a los sensores del sistema el tiempo suficiente como mostramos en el protocolo de inicio de la sección 6.4.1.
De esta forma se inicia el protocolo de inicialización de los sensores y se estudian las propiedades características de cada temperatura que posteriormente veremos en la sección 4.2.3.

La figura 4.2, muestra la estabilidad conseguida en ambos sistemas a diferentes temperaturas de calefacción. En ambos casos, se han eliminado las muestras iniciales por no ser representativas.

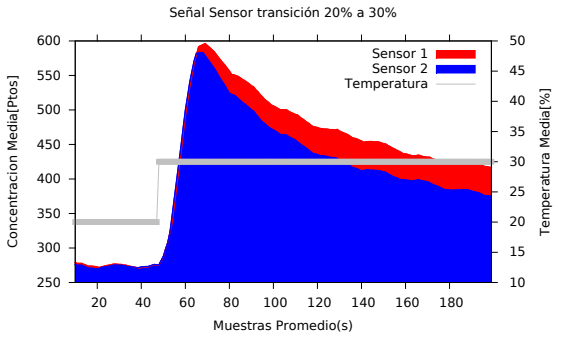
4.2.2. Histéresis

La histéresis se define como el efecto sucedido de respuesta en un sistema dependiente no solo del estímulo a la entrada del mismo si no también de la historia de anteriores entradas y dinámicas sucedidas en momentos previos. Este efecto es determinante en nuestro sistema puesto que las mediciones instantáneas no presentan información, éstas pueden ser ocasionadas por el estímulo actual o por la herencia de dinámicas anteriores. Es por esto que el protocolo de inicio cobra importancia en el funcionamiento del sistema.

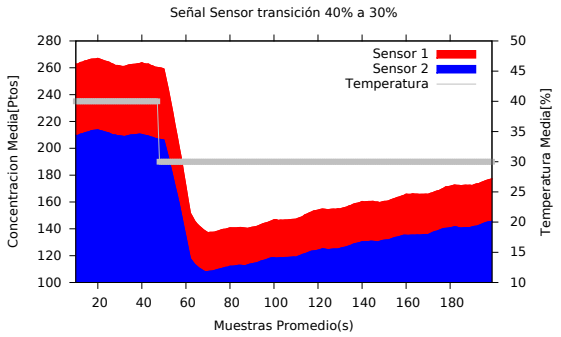
Para demostrar este efecto planteamos la siguiente figura comparativa 4.3.

- La figura 4.3a muestra la influencia en el sistema de un salto de temperatura en el calefactor del sensor. El paso realizado es incremental de 20 % de temperatura al 30 %.
- La figura 4.3b muestra la influencia en el sistema de un salto de temperatura en el calefactor del sensor pero en este caso decremental de 40 % de temperatura al 30 %. En ambas figuras influenciadas por la temperatura se observan comportamientos equivalentes de sentido contrario.
- La figura 4.3c se muestra el efecto de histéresis influenciado por la exposición a odorante.
- La figura 4.3d representa la figura referencia a la misma temperatura que el resto de figuras pero en estado estable, no hay exposición anterior a odorante ni cambio de temperatura, esta señal caracteriza el estado de estabilidad deseable en nuestro sistema.

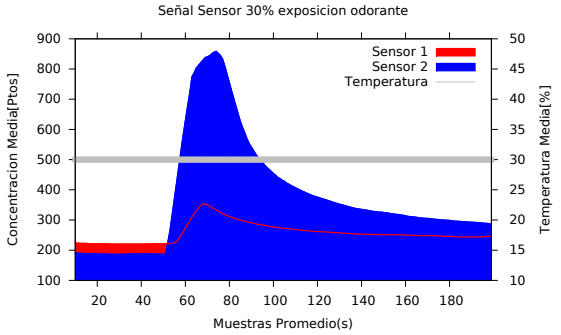
Tomando la referencia de la señal d, podemos ver lo diferentes que pueden llegar a ser las señales de los sensores aún encontrándose a la misma temperatura. Esto efecto demuestra como parte de la información de nuestro sistema se encuentra en las diferencias temporales, en las tendencias y en los valores de variabilidad, eliminando como valor confiable los valores inmediatos medidos de conductividad de los sensores.



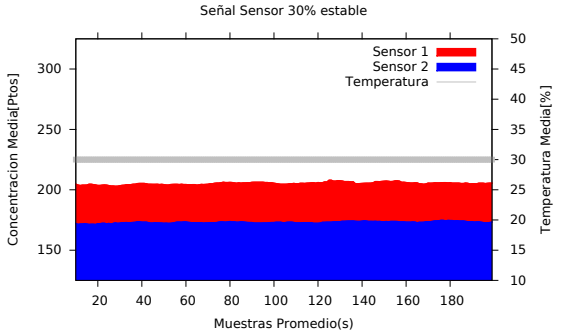
(a) Señal posterior Temperatura 20 %.



(b) Señal posterior Temperatura 40 %.



(c) Señal posterior a la exposición de odorante.



(d) Señal estable a 30 %.

Figura 4.3: Efecto de histéresis en las señales de los sensores TGS2611. Subfigura d tomada como referencia al 30 % del valor máximo de temperatura(5v). Subfigura a y b muestra el efecto de la histéresis provocado por el cambio de temperatura, en el primer caso, por un incremento en la temperatura; en el segundo caso, por un decremento en la temperatura. Subfigura c, efecto de histeresis ante la exposición de etanol al 10 % de concentración. Apreciación de pendientes indeseadas provocadas por estados anteriores del sistema.

4.2.3. Metodología de caracterización

La caracterización de los parámetros que influencian la dispersión de odorantes de forma unívoca no es tarea fácil, como anteriormente detallamos, existen multitud de variables complejas que afectan a nuestro sistema. Partiendo de este punto, hemos desarrollado un protocolo experimental para la observación de este tipo de parámetros en relación con las variables de control de nuestro sistema tales como las distancias a la fuente, los reactivos utilizados o las temperaturas de calentamiento de nuestros sensores.

A continuación vamos a parametrizar la medición de parámetros característicos ante la exposición de odorantes, éstos están representados en la figura 4.4 y son los siguientes:

- **Exposición.** Definida por el incremento en concentración del sensor desde el punto de estabilidad hasta el pico de mayor concentración medida, ésta medida es a su vez dependiente de la sensibilidad del sistema al odorante.

- **Durabilidad.** Número de muestras en el cual el odorante influencia la toma muestral de forma considerable. Este parámetro es medido desde que se inicia la pendiente de concentración de la muestra hasta que vuelve a su estado de estabilidad. A su vez, podemos caracterizar al mismo tiempo un tipo de ventana de temperatura si nunca recobra la estabilidad para un número de muestras posteriores a la exposición.

- **Variabilidad.** Parámetro de inestabilidad del sistema ante la presencia de odorante. Esta medida será obtenida por el número de máximos locales representados en la señal del sistema.

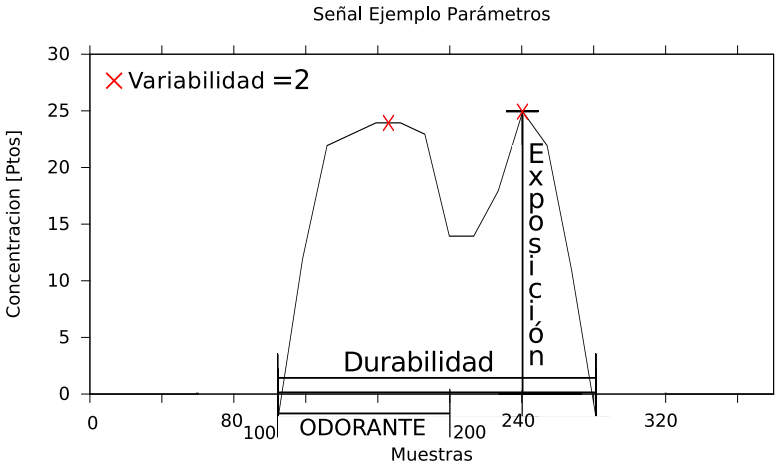


Figura 4.4: Figura ejemplo de la metodología y los parámetros a tener en cuenta en el estudio de la dispersión de odorantes.

Es conocido que los sensores tienen un comportamiento no lineal ante la presencia de odorantes con respecto a la temperatura de calefacción. La realización de un estudio experimental utilizando esta metodología tiene por objetivo la selección de la temperatura de calefacción que facilite la explotación de los algoritmos.
En particular, el parámetro que buscamos es la ventana de temperatura que anteriormente hemos presentado en la subsección 4.2.1 y que conformará la temperatura inicial del algoritmo.

Partiendo del estudio de este parámetro obtenemos las siguientes hipótesis del sistema:

- A altas temperaturas, ventanas superiores al 50 %, la sensibilidad es muy pequeña el parámetro de exposición se mantiene en el mínimo respecto el resto de temperaturas. La durabilidad es baja pero el recobro a señales estables es muy rápido. Este tipo de comportamiento es muy útil dada su estabilidad si utilizamos los sensores en modo detector, es decir, que activaremos una señal cuando el valor de concentración supere el límite establecido.

- En temperaturas entorno al 50 % la sensibilidad empieza a aumentar pero la variabilidad que obtenemos es mucho mayor. Debido a este comportamiento las señales podrían tomar valores oscilantes y repercutir en errores de la toma de decisión.

- A bajas temperaturas, la sensibilidad empieza a disminuir de nuevo pero el efecto que define el estado es un recobro muy lento, con lo que podría dar lugar a error con mayor facilidad al verse tan afectado por la histéresis, en temperaturas por debajo de 20, la señal no alcanza el recobro en la ventana definida en la figura 4.4.

Para nuestro problema buscamos que la sensibilidad del sistema sea elevada pero a la vez que mantenga una variabilidad baja y recobro activo de la señal. Como hemos comprobado experimentalmente la ventana 30 % de la temperatura nos ofrece estas características ademas de mantenerse en un lugar intermedio del rango lineal de sensibilidad apreciado en las figuras de la siguiente sección 4.3.1. Los algoritmos desarrollados tomaran esta temperatura como punto de partida, y las señales estables serán siempre caracterizadas por esta señal de voltaje en el calentador.

4.3. Aplicación de técnicas de modulación

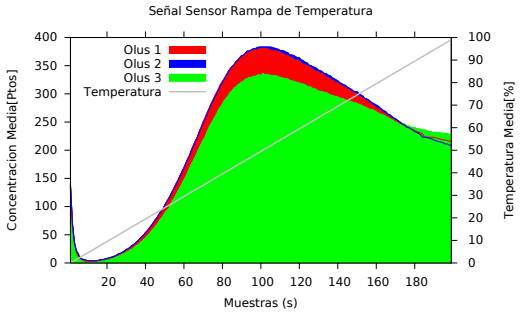
En la literatura se han usado diferentes tipos de modulación de temperatura para la consecución de diferentes estudios. En algunos casos la modulación de temperatura se realiza con una señal de forma cuadrada [55], en otras la técnica empleada se realiza con forma de incremento lineal [54] o incluso a través de impulsos de temperatura [56] pero en todas ellas se ha demostrado la aparición de diferentes comportamientos fisicoquímicos en el sensor dependientes del control de la temperatura.
El control mediante modulación de temperatura nos otorga una fuente de información en los valores transitorios del sistema, una técnica para minimizar el efecto de la histéresis y modificar la sensibilidad del sistema a nuestra conveniencia [1, 57, 2]. Estas serán las funciones que cumplirán las técnicas en nuestro sistema.

Como hemos podido empezar a vislumbrar en secciones anteriores la temperatura en la que se emplaza la placa del sensor mantiene un rol vital en la obtención de información a la hora de explotar nuestro sistema a través de las narices electrónicas. En esta sección aplicaremos las técnicas usadas con anterioridad en estudios realizados en el GNB [2, 1] utilizadas en este tipo de sensores.

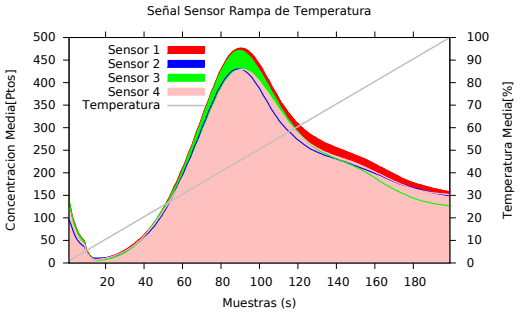
4.3.1. Rampa de temperatura

Aunque algunos autores de referencia utilizan una fase inicial de limpieza con gases inertes para la eliminación de efectos indeseados, ante la complejidad del procedimiento hemos podido demostrar que ante un incremento lineal entre el mínimo de temperatura y el máximo, los valores de sensibilidad del sistema se reinician en cierta medida y el efecto de histéresis es imperceptible para nuestros análisis, este comportamiento ya había sido antes utilizado en [1] en el cual calentaban al sensor al 100 % de su temperatura máxima para limpiar el sensor de posibles impurezas.
Por lo tanto hemos estandarizado un procedimiento de reinicio de los sensores para la reproducibilidad de los experimentos realizados. La primera parte consta de esta rampa de temperatura a incrementos lineales que modifica la sensibilidad de los sensores hasta la obtención de señales de concentración entorno al cero, este efecto es independiente de la exposición anterior del sistema.

La figura 4.5 representa la señal de los sensores durante este procedimiento de *limpieza*. De la misma figura podemos obtener una aproximación al margen dinámico de temperaturas sobre el cual la sensibilidad del sensor es más o menos lineal con la temperatura. Este margen se encuentra entorno al 20 % y el 50 % y será esta limitación de temperatura la que aplicaremos al algoritmo de localización.
Por encima del 50 % algunas señales de los sensores comienzan a curvarse con lo que la relación se vuelve fuertemente no lineal. Aunque los sensores son intrínsecamente diferentes y provienen de estados diferentes este comportamiento es altamente reproducible en ambos sistemas desarrollados.



(a) Señal Rampa Sistema Olus.



(b) Señal Rampa Sistema Integrado.

Figura 4.5: Incremento lineal de temperatura en ambos sistemas desarrollados. Representación de la primera técnica modulatoria de la temperatura aplicada en el encendido e inicio de los sistemas. Limpieza de sensores y minimización de efecto de histéresis. Obtención del rango lineal del sistema

4.3.2. Modulación sinusoidal

Bloque fundamental para el procesado de la señal concentración de los sensores. Estudiaremos el impacto de la aplicación de este tipo de modulación aplicado a la temperatura de calefacción de los sensores.
Anteriormente en las conclusiones obtenidas de la sección 4.2.3 hemos podido demostrar que la sensibilidad del sistema ante la presencia de odorantes tiene relación directa con la temperatura del calentador del sistema. En esta sección observaremos como influencia la oscilación de la temperatura de calefacción en las señales de concentración.

Ventajas de la modulación

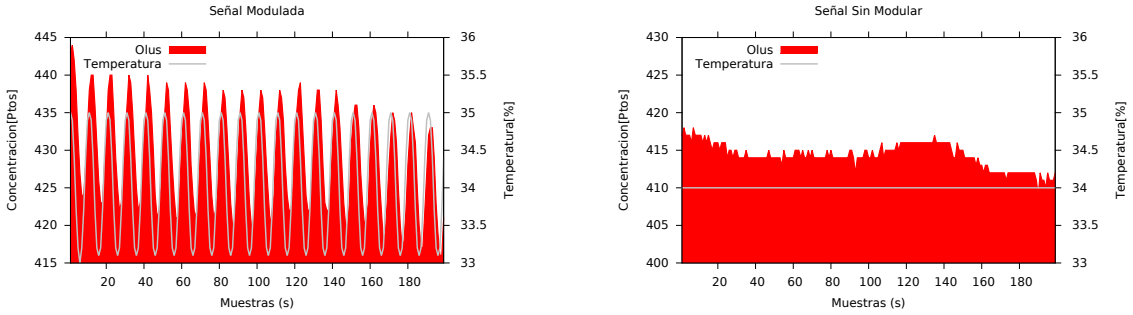
Si bien es cierto que cada sensor posee una curva característica de sensibilidad única con respecto la temperatura de calefacción, podemos demostrar que las pequeñas variaciones de temperatura son seguidas linealmente por todos los sensores tipo MOS con los que hemos experimentado. Este comportamiento es apreciable en la figura representativa 4.5.
La modulación nos otorga robustez frente al ruido en la señal al inducir una señal periódica procedente del aumento y disminución de la sensibilidad. La obtención de las señales periodicas también nos ofrece facilidad en el post-procesado de la información de los sensores.

Al aire libre, como hemos realizado las tomas muestrales de la figura 4.6 se aprecian pequeños picos irregulares de la señal sin control de temperatura. Estos picos los consideramos ruido de la señal cuyo origen más probable sea la temperatura del calentador.

Obtenemos dos ventajas claras de la aplicación de esta técnica:

- La respuesta del sistema se vuelve más regular sobre la señal seno aplicando la modulación. La variación de la señal obtenida de los sensores es altamente correlada con la señal de la temperatura. Este comportamiento, nos otorga facilidad en el tratamiento y uso de las señales.

- La aparición de picos de ruido característicos en la señal sin variación sinusoidal desaparecen en la señal aplicada con el perfil seno de temperatura. El ruido obtenido en la señal a temperatura constante no es apreciable en la señal sinusoidal ya que la señal de los sensores es altamente coherente con el perfil modulatorio, no aparece ruido en forma de picos en las señales de esta ventana. El calentamiento dinámico aplicado nos permite mantener a



(a) Señal Temperatura Modulada.

(b) Señal Temperatura Constante .

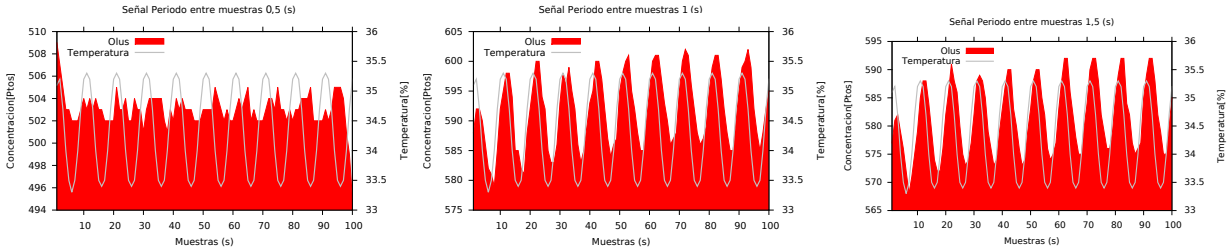
Figura 4.6: Efecto del perfil sinusoidal en las señales provenientes de los sensores TGS2611. Comparativa entre señales a misma temperatura, en la ventana a con modulación de temperatura, en la ventana b con temperatura constante. Observamos picos de ruido en la señal sin modulación que quedan eliminados en el perfil modulatorio.

la señal de concentración del sensor en un estado controlado de variación.

Frecuencia de muestreo

La frecuencia de la toma de muestras del sistema es de vital importancia. Si la frecuencia fuera muy baja obtendríamos información redundante y sería ineficiente; si la frecuencia fuera muy alta el sistema no sería capaz de responder a las temperatura impuestas y perderíamos información.
En el sistema de medición la frecuencia viene dada por la inversa del tiempo de espera entre muestreo y muestreo de los sensores. En este tiempo de espera el calentamiento ejerce control sobre la sensibilidad del sensor. Se puede observar más técnicamente en el código adjunto del anexo C.

La figura que mostramos ofrece información para la toma de decisión en lo que respecta a la frecuencia. Hemos seleccionado aquella que sigue de forma óptima la señal muestral respecto el perfil modulatorio de la temperatura.
Las muestras que mostramos en la figura 4.7 han sido tomadas en las mismas condiciones y con independencia muestral entre sí; reproduciendo el experimento se han obtenido resultados equivalentes e igual conclusión.



(a) f = 2 muestras por segundo.

(b) f = 1 muestras por segundo.

(c) f = 0.6 muestras por segundo.

Figura 4.7: Comparativa de frecuencias en sensor. Apreciación de mejora en la estabilidad de la señal sinusoidal derivada de la modificación en temperatura según la señal de control seno. Aprovechamiento de tiempo entre muestras en la señal a frecuencia 1 muestra por segundo.

Las ventana con frecuencia inferior no sigue el perfil de temperatura debido a que no llega a calentar el sensor, el resultado es una señal muestral con poca apariencia sinusoidal e inestable. En el ultimo caso de 0,6 muestras por segundo, la señal sigue de forma correcta el seno establecido pero las tomas muestrales pero se incrementaría en el tiempo con respecto la frecuencia 1 para obtener el mismo número de muestras.
Como observamos en la figura 4.7 existen valores de frecuencia que añaden ruido al sistema debido a la ineficacia del control de temperatura. Seleccionamos finalmente la frecuencia de 1 muestra por segundo para la aplicación en nuestro sistema.

4.4. Entornos de prueba

Otro factor clave en el desempeño de nuestro sistema pasa por el ambiente externo en el que se sitúa, como podemos ver en muchas de las referencias citadas de la sección 2.3, los algoritmos de búsqueda tienden a aproximaciones diferentes con relación al entorno de explotación, por lo que podemos caracterizar al algoritmo en sí mismo por el entorno en el que se desarrolla.
La primera sección introducirá las narices electrónicas que posteriormente detallaremos en el capítulo 6, diferentes aproximaciones a las narices electrónicas conllevan diferentes elementos de control y ajuste de los parámetros implicados. En la segunda sección se definirá el entorno de pruebas, los elementos que componen una sala o un espacio abierto juegan un papel importante en la dispersión de los odorantes y por lo tanto en la caracterización de las perturbaciones que afectaran a las medidas obtenidas de las narices electrónicas. Finalmente, se definirá la representación de la fuente de odorante, tanto por su forma como por el posicionamiento, concentración y reactivo.

4.4.1. Sistemas de prueba

El desarrollo de este proyecto tiene un carácter evolutivo, el proceso por el cual nos familiarizamos con el problema juega un papel fundamental a la hora de la puesta en funcionamiento de las estrategias de control ya conocidas. Es por esto que trataremos de explicar brevemente los sistemas sobre los que se han desarrollado los algoritmos, aunque en posteriores capítulos entraremos más en detalle tanto en su parte electrónica, capítulo 6, como en el tipo de pruebas realizadas, sección 7.4, y las particularidades del algoritmo.

Nariz Olus

Basada en la nariz electrónica desarrollada en el GNB [1, 2, 8], la figura 4.8 muestra el sistema y el esquemático del mismo está expuesto en el anexo A. El sistema cuenta con sensores tipo MOS modelo TGS2611 de la casa Fígaro, posee dos microcontroladores PIC18LF4550 para la posible conexión de sensores externos y la comunicación con el mismo se realiza a través del puerto serie de datos controlado por el protocolo RS485.
Este diseño ha sido probado en otras ocasiones ofreciendo robustez electrónica tanto de sus componentes como de los sensores por lo que ofrecen versatilidad a la hora de afrontar diferentes problemas con fuentes de odorante.

En el lado del software, se proporcionó unas librerías en lenguaje de programación C++ desarrolladas por Fernando Herrero-Carrón [2], donde se implementaba funciones básicas de muestreo en los encapsulados así como comunicación con el control de la temperatura de calefacción a través del pin PWM proporcionado en el microcontrolador.
En este tipo de sistema, el microcontrolador PIC se encarga de la manipulación de las señales internas, es necesario la comunicación con el mismo a través del envío de parámetros sobre su puerto serie. Esta rigidez es una de las motivaciones por las que se decidió el diseño de un sistema de nariz electrónica únicamente formada con la parte de acondicionamiento de señal y que fuera compatible con cualquier microcontrolador como unidad central del sistema.

Las narices electrónicas habían sido utilizados en otros proyectos durante un largo periodo de tiempo en el cuál se habían sometido a los mismos a condiciones de temperatura y humedad extremas, condiciones que otorgaron a los sensores de envejecimiento prematuro otorgando capacidades físicoquímicas alteradas respecto a sus equivalentes nuevos. Este efecto puede ser observado en comparación con los mismos sensores nuevos en la sección 7.4.4.

Figura 4.8: Encapsulado Olus. Control de señal a través de controlador integrado en placa PIC18LF4550.

Nariz Integrada

Desarrollada en la duración del proyecto, se implementa una nariz electrónica con la intención de realizar un sistema integrado en la placa microcontroladora que controlará la plataforma robótica sobre las que se van a montar finalmente las narices. La figura 4.9 muestra una imagen de la nariz.
El modelo integrado pretendió en todo momento integrar los mismos mecanismos de control electrónicos que su análogo en los encapsulados para que a su vez, poder migrar los algoritmos y estudios realizados anteriormente con los dispositivos encapsulados Olus.

Las integración con la placa microcontroladora que será utilizada tanto de rama de alimentación como de rama de control del sistema son el reto con respecto el diseño electrónico que apareció en el desarrollo de los mismos. El calentamiento será realizado a través del PWM de la placa y es por ésto que el acondicionamiento de ese módulo del sensor es la parte más importante.
En el posterior capítulo 5 detallaremos como fue la evolución de las narices integradas partiendo del diseño básico planteado por el fabricante hasta la finalización en este sistema.
Los sensores utilizados han sido los mismos que en los encapsulados, modelo TGS2611 de la casa Fígaro, con lo que se han podido utilizar para ambos sistemas los mismos reactivos y principios de control en la práctica de los algoritmos de explotación.

El sistema final es, como su propio nombre indica y en yuxtaposición con el sistema inicial, integrado y autónomo con lo que en la placa microcontroladora se desarrollaran todos los algoritmos de toma de decisión y control de los distintas partes que componen el sistema olfativo artificial utilizado en la búsqueda de odorantes.

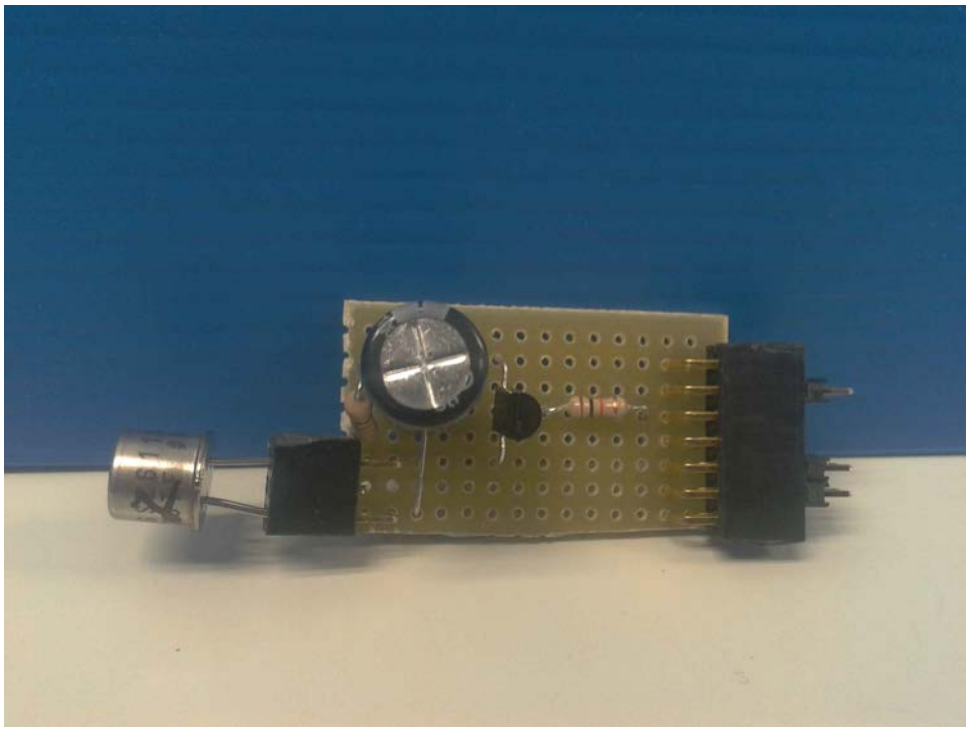


Figura 4.9: Diseño nariz integrada en placa de prototipado. Diseño final v2. Integración con BBB.

4.4.2. Entorno de explotación

La definición del entorno sobre el que se van a desarrollar la pruebas es fundamental para entender las premisas sobre las que construimos el algoritmo de búsqueda. La sala de pruebas se caracteriza por:

- **Ambiente de dispersión.** Se explota el algoritmo en un ambiente cerrado en la cual las corrientes son más o menos controladas o al menos despreciables en comparación con un entorno al aire libre. Debido a la dispersión de la fuente, los algoritmos desarrollados en ambientes controlados pueden extraer información del odorante de manera continua, el rastro de odorante no va a ser eliminado totalmente; por otro lado, en un tipo de ambiente abierto los algoritmos se enfocan más en la obtención de la pluma de odorante y explotan para ello tanto las señales de los sensores de odorantes como la integración de otros sensores como anemómetros o brújulas electrónicas.
- **Presencia de obstáculos.** La superficie sobre la que se explota el algoritmo es una superficie cuadrada de aproximadamente 4 metros de largo por 4 metros de ancho, es una superficie lisa y no existe obstáculo ninguno entre el posicionamiento inicial de la búsqueda y la linea de dispersión de la fuente de odorante.

Figura 4.10: Entorno de experimentación.

4.4.3. Reactivos y concentraciones

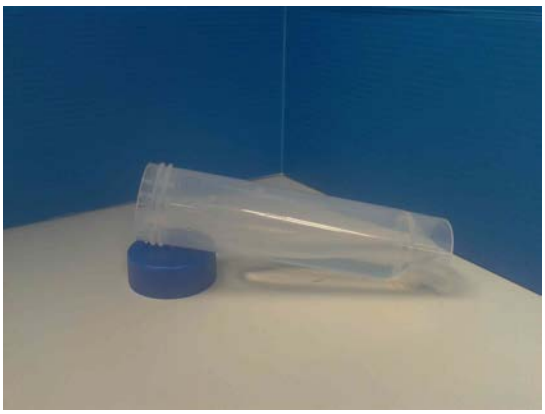
Dada la hoja de características de los sensores modelo TGS2611, que será el utilizado en nuestras narices, observamos que en la aplicación de detección de gases la sensibilidad máxima del sensor está caracterizada por etanol, iso-butanol y metano. Utilizaremos etanol por facilidad en su adquisición.

Utilizamos dos medios para la representación de la fuente de odorante en el algoritmo, inicialmente y dado el caracter unidimensional de la dispersión, utilizamos un tubo como se indica en la figura 4.11a. Conforme se realizaron experimentos tomando decisiones en dos ejes se utilizó la fuente de la figura 4.11b, el odorante se sitúa entre los dos platos y se dispersa de forma uniforme por toda la superficie de experimentación.

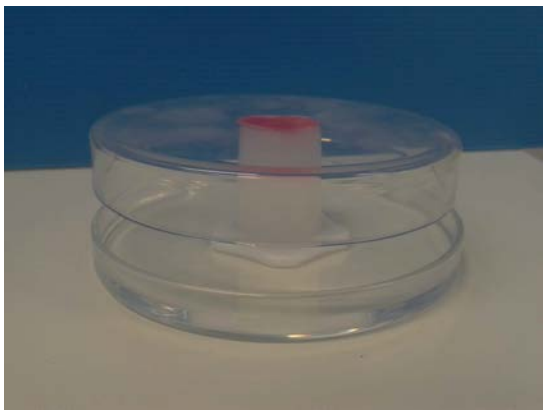
Para ambos casos se tuvo en cuenta que la dispersión debía de realizarse cerca del suelo como vemos en la sección 2.2.2 del estado del arte, puesto que la dispersión efectiva para la búsqueda debe de realizarse unos milímetros por encima del suelo.

Las concentraciones de las fuentes han sido experimentalmente seleccionadas aplicando los criterios de medida de la sección 4.2.3.

- La dispersión de la fuente lineal se realizará con 10 cl de etanol a 10 % inclinandolo hasta alcanzar una altura respecto la entrada del tubo de 5 mm facilitando así la volatilidad. Este método de difusión nos permite representar la fuente como una fuente continua de dispersión lineal creando un entorno homogéneo.
- La dispersión de la fuente circular se realizará con 30 cl de etanol a 50 %. Este método mantiene la dispersión bidimensional al crear una abertura lateral entre los dos platos colocados uno encima del otro. La alta concentración de odorante ha sido seleccionada como anteriormente describimos de forma experimental, concentraciones menores han sido desechadas debido a la falta de señal obtenida en distancias del algoritmo (125 cm).



(a) Fuente dispersión lineal.



(b) Fuente dispersión circular.

Figura 4.11: Representación fuente de odorante. En la figura a se encuentra la fuente de odorante de tipo lineal, usada en los experimentos en una dimensión tanto del sistema Olus como del integrado. El odorante para este caso son 10cl de una disolución de etanol en agua en una concentración del 10 %. En la figura b se encuentra la fuente de odorante de tipo circular, usada en los experimentos de tipo bidimensional debido a su dispersión homogénea en 360 grados. La concentración de etanol en este caso es del 50 % con una cantidad de 30cl.

5

Narices electrónicas: acondicionamiento para modulación de temperatura

5.1. Introducción

En este capítulo abordaremos el desarrollo de los sistemas diseñados para la captación de odorantes que se han desarrollado en la plataforma robótica. Debido a la complejidad del sistema desarrollado para la explotación del algoritmo, observable en el próximo capítulo, y partiendo de los diseños Olus se diseñaron narices electrónicas portables con capacidad de control desde cualquier microcontrolador.
El sistema resultado dispondrá de una única unidad de control centralizada para la plataforma robótica y las narices electrónicas. En ella se automatizaran los procesos de muestreo, ajuste de parámetros y toma de decisiones que modificarán el posicionamiento de sistema integrado. De esta manera obtenemos un dispositivo autónomo e integrado en lugar de soluciones híbridas como observamos en el sistema Olus 6.2.

Mediante el estudio del acondicionamiento de señal necesario en este capítulo se expondrán los sistemas desarrollados de captación de odorantes para la monitorización, control y explotación de entornos con dispersión de odorantes. En una primera instancia se desarrolla según las instrucciones del fabricante, lo que hemos denominado versión 1; posteriormente en la posterior evolución del sistema, denominado como versión 2 de los sensores, analizaremos las fuentes de ruido de la señal para obtener señales robustas que puedan integrarse tanto en nuestra placa microcontroladora como en otra de semejantes características.

5.2. Sensores v1

5.2.1. Introducción

En primer lugar se planteó el sistema más sencillo posible tal como observamos a nuestro sensor acondicionado en la hoja de características [9]. En el mismo la señal analógica del sensor a utilizar se obtiene del divisor de tensión de la fuente de alimentación(5 voltios) respecto al sensor midiendo en la resistencia de carga. De esta forma y siendo proporcional al nivel de partículas de odorante en el sensor, se obtiene la señal inversa de la resistividad en el sensor de odorante. A este valor le hemos denominado puntos de concentración de odorante y aún siendo

un valor de voltaje, será la variación del mismo la que se considerará explotable para nuestros sistemas de localización de odorantes.

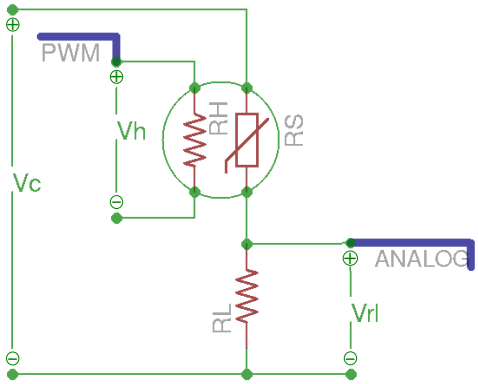


Figura 5.1: Circuito acondicionador TGS2620

Variable	Valor
Vc	5 (V)
Vh	5 (V)
Vrl	Medida
Rh	83 (Ω)
Rs	1 a 5 (kΩ)
Rl	0.45 (kΩ) min

Cuadro 5.1: Tabla de Valores TGS2620

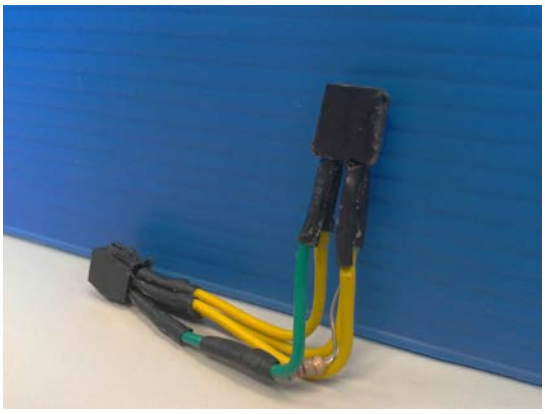
5.2.2. Integración de sensores

Partiendo de este diseño se encapsuló cableando las conexiones a un zócalo y se anexionó un plástico protector diseñado con impresora 3D que otorga resistencia y facilidad de integración, código en anexo D.
El valor de la resistencia del divisor de tensión respecto el sensor de odorante únicamente desplazará la escala del sistema, para su selección se tuvieron en cuenta los valores de concentraciones máximos y mínimos de forma que no sobrepasara el límite de 1,8 voltios impuesto por el pin analógico de la BBB, tabla 3.2.
Finalmente la implementación del sistema en la plataforma con los cuatro sensores que componen el sistema quedaría tal y como observamos en la figura 5.2.

5.2.3. Medición de la calidad del sistema

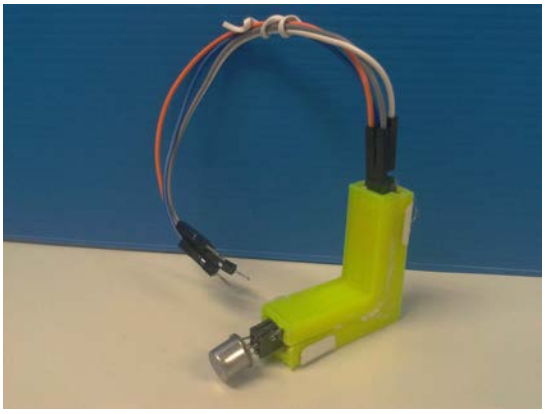
Las pruebas realizadas sobre este esquema fueron, como es el objetivo de la integración, realizadas en la placa microcontroladora BBB que finalmente hospedaría el control de la plataforma robótica y de los sensores.
El calentamiento del sensor se realizará mediante los pines dedicados al PWM que mediante la técnica de modulación de pulsos a alta frecuencia consigue mantener un determinado voltaje en uno de sus pines. Tomaremos ventaja de este método para controlar la calefacción del sensor de forma precisa, que es una de las características explotables de nuestro sistema.
La obtención de las señales analógicas se extrae de cualquiera de sus pines de entrada reservados al ámbito analógico cuyo rango de voltaje de entrada es 0-1,8(V) con resolución de 1 (mv).

Las señales obtenidas mediante esta integración son comparadas utilizando la medida de la desviación típica respecto su señal promedio sin exponer los sensores a fuentes de odorante. El valor ideal debería de ser 0 que correspondería a la señal constante en el tiempo, la tabla 5.2 muestra los resultados obtenidos a dos temperaturas por encima del 50 %, estos valores pueden compararse con los obtenidos en la tabla 5.3 del sistema evolucionado.

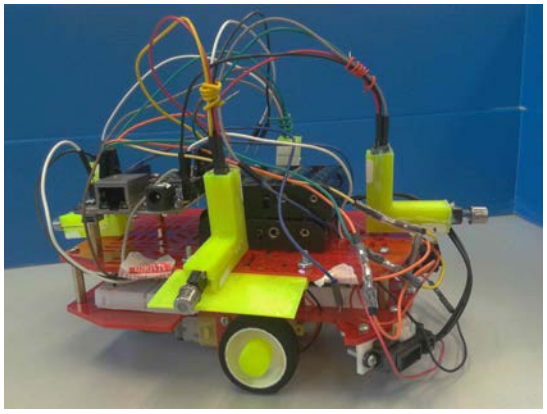


(a) Conexionado básico con RL

(b) Piezas impresas



(c) Unidad sensorial



(d) Integración final sensores

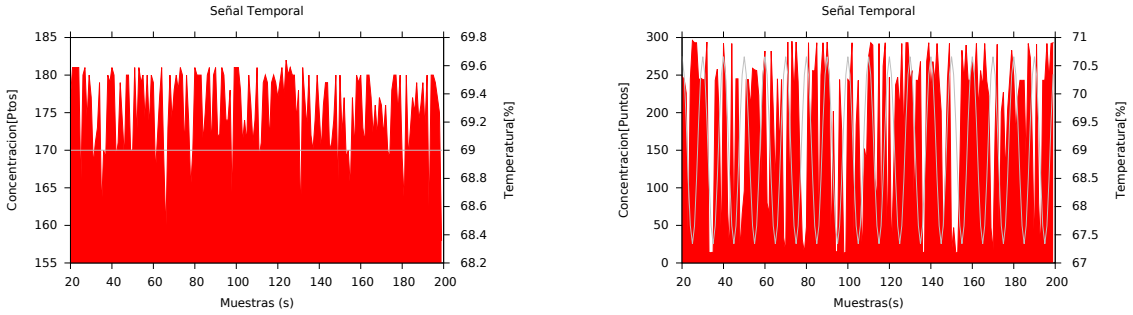
Figura 5.2: Integración de sensores v1. Las imágenes representan los sensores con el acondicionamiento realizado, las piezas impresas y la integración final en la nariz electrónica y la plataforma robótica multinariz v1.

Temperatura (%)	Desviación típica σ (Ptos. Concentración)
45	18,41
64	13,57

Cuadro 5.2: Medida de dispersión media de todos los sensores a temperatura constante respecto la señal promedio conformada sobre 200 puntos de muestreo. Señales tomadas sin exposición de odorante. Señales reescaladas a 5 voltios para posible comparativa con tabla de versión v2 5.3.

Una referencia visual de fácil seguimiento puede arrojarnos luz acerca del comportamiento ruidoso de la señal caracterizada por la tabla 5.2.
La figura 5.3 nos muestra como la señal de los sensores no es en ningún caso constante como cabría esperar del comportamiento ideal del sistema. Ambas se consideran señales muy ruidosas alejadas de las señales obtenidas mediante los encapsulados iniciales por lo que una toma de decisión basada en estas señales pueden ocasionar errores debidos al ruido. En particular, la señal es más ruidosa al aplicar la función seno en el calentamiento de los sensores, y además en ningún caso se observa seguimiento del mismo o periodicidad de la señal.

La señal de calefacción que vamos a utilizar es, en voltaje, inferior al límite recomendado por el fabricante para este tipo de acondicionamiento. Es por esto que el acondicionamiento debe de ser más robusto frente al ruido. La fuente de ruido será estudiada y minimizada en la siguiente sección en la cual desarrollaremos la nariz integrada v2.



(a) Calentamiento constante.

(b) Calentamiento Sinusoidal.

Figura 5.3: Ruido en sistema v1, sensor TGS2611. Apreciación de ruido en señales estable. Dos casos posibles de calentamiento: ventana a, temperatura estable a 69 %; ventana b, calentamiento sinusoidal con temperatura media 69 %. El 69 % es respecto 3,3 (V), la medida reescalada a 5 (V) equivaldría a 45 %.

Este sistema no es explotable en nuestro algoritmo debido al ruido inducido. Este sistema podrá ser utilizado para otro tipo de aplicaciones las cuales no se necesite un control del voltaje en la temperatura de calefacción. El funcionamiento de este tipo de acondicionamiento quedó demostrado en el trabajo de fin de grado de Carlos Garcia Saura [12] en el cual la nariz electrónica se calentaba de forma constante a 5 voltios, como recomienda el fabricante, y los sensores localizaban la fuente de odorante mediante un umbral binario que se activaba cuando se aproximaban a la fuente.

5.2.4. Conclusiones

Este sistema resultó deficiente para nuestra aplicación por los siguientes motivos:

- Debido al manejo de las señales de calentamiento que aplicaremos en nuestro algoritmo, debemos de calentar por debajo de la temperatura recomendada. Este comportamiento, demostramos posteriormente en la tabla 5.3, aumenta la desviación típica de los valores muestrales pero además, la modulación utilizada también la aumenta, en conjunto, las señales son demasiado ruidosas para ser explotadas.
- El rango de trabajo de los sensores no es cubierto por este diseño. Puesto que tenemos la limitación del PWM de nuestro microcontrolador BBB a 3,3 voltios debemos de aplicar un acondicionamiento externo a la señal de entrada para alcanzar los 5 voltios.

5.3. Sensores v2

5.3.1. Introducción

Tras la aparición de los efectos no deseados observados en las pruebas realizadas de la versión 1 de nuestros sensores, se plantea el uso de diferentes componentes de apoyo que aporten a las narices de la estabilidad necesaria para la ejecución correcta de las técnicas utilizadas en el algoritmo desarrollado. La introducción de elementos a nuestro sistema es incremental en el orden de aparición de este capítulo por lo que finalmente el sistema queda como el circuito de la figura 5.6.

- **Transistor NPN.** La introducción del transistor como componente activo suministrara la corriente necesaria para alcanzar el rango dinámico de tensión en el calentador de los sensores. Trasladara los 3,3 voltios que suministra el PWM a los 5 voltios que será el límite máximo de la señal de calefacción. El uso de un transistor se plantea como la solución más robusta en cuanto a estabilidad valorando otras aplicaciones como circuitos amplificadores.
- **Condensadores.** Para filtrar el ruido de nuestra señal así como el desacoplo de la fuente de alimentación con respecto los componentes del circuito.
- **Operacional.** Componente que en modo seguidor de tensión, aísla las etapas de nuestro diseño. Estudio para un posible trabajo futuro en el que se añadan nuevas etapas en cascada a la salida analógica del sistema.

5.3.2. Selección de componentes

Transistor

El transistor será el componente mediante el cual ampliaremos el rango dinámico del calefactor hasta los 5 voltios que es la temperatura máxima de calefacción (Vh). El control de la base se realiza mediante el pin del PWM de la BBB, que como vemos en la tabla 3.2, está limitado en tensión. Para tener una relación directa entre PWM y calefactor a la vez que exista un punto de corte coincidente a 0 voltios, elegimos un tipo de transistor NPN.
A continuación, es la alimentación del sistema y la resistencia de calefacción del sensor la que marca el cálculo de las corrientes de colector dado que alcanza la corriente máxima cuando el sistema consume los 5 voltios de alimentación. Por consiguiente, el valor de amplificación de entrada según las ecuaciones de control del transistor de la ecuación 5.1, es:

$$I_C = 5v/R_H \qquad \beta = I_C/I_B \tag{5.1}$$

$$I_B = 6mA \text{ máximo del pin PWM (BBB)}$$

$$\beta = 10 \tag{5.2}$$

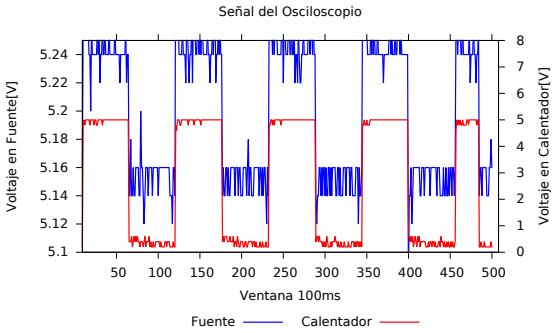
El valor de la amplificación es muy pequeño y no será este parámetro importante para la selección del transistor NPN. Finalmente entre la oferta disponible seleccionamos el PN2222A que cumple con las especificaciones del sistema.
La base del transistor se conectará con el PWM de la placa microcontroladora para garantizar un control óptimo de la tensión de calentamiento a través que la resistencia de protección de base de 1KΩ. Esquemático final 5.6.

Condensadores

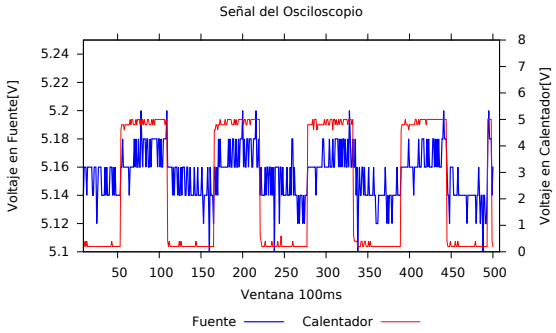
La introducción de condensadores en el circuito sensorial de la nariz se hace prioritario al observar el ruido obtenido en las señales de los sensores 5.3. Tras la experimentación con valores entorno a los microfaradios en configuraciones de desacoplo de fuente situados en los puntos críticos del sensor(calefacción a masa y alimentación a masa) 5.6 no obtenemos resultados aceptables en comparación con la calidad obtenida en los encapsulados Olus.

Planteamos el estudio de las señales eléctricas a un nivel más detallado para conocer la fuente de ruido de la señal. Finalmente observamos en la primera figura como el PWM de la señal provoca una caída de tensión en la linea de alimentación 5.4a.
Este ruido es provocado por la rama de calefacción del sistema conectada a la alimentación con el acondicionamiento del transistor detallado en la anterior sección. La rama de alimentación además está a su vez conectada con la rama de medida del divisor de tensión del sensor y por lo tanto puede afectar en gran medida las mediciones del sistema.

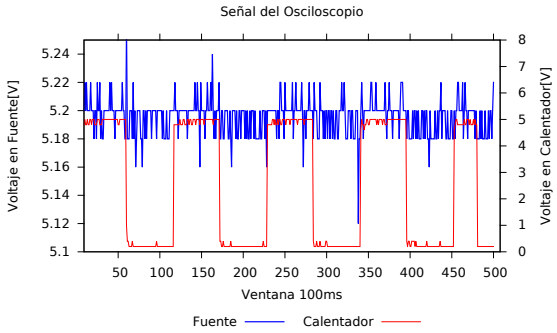
Por lo tanto para el filtrado efectivo de ruido provocado por el PWM nos basamos en la utilización de un condensador en configuración de *bypass*. Referenciando la guía de filtrado [58], seleccionamos condensadores del orden de los 100μF para las frecuencias del orden de Khz del PWM de la señal.



(a) Figura inicial. Alimentación(Vc) frente a calentamiento(Vh).



(b) Alimentación(Vc). Filtrado 100μF.



(c) Alimentación(Vc). Filtrado 1000μF.

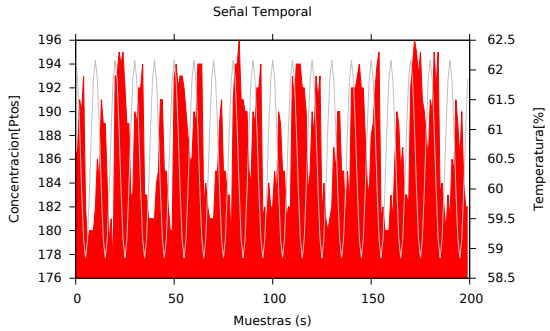
Figura 5.4: Bypass de la señal de alimentación debido al PWM. Medidas del circuito realizadas en bornas del calentador del sensor (Vh) y de la entrada del mismo (Vc) que representa la linea de alimentación del circuito. Filtrado de la señal ruido proveniente del PWM al tener el calentador conectado a la alimentación del circuito según la figura 5.6.

Concluimos finalmente que el ruido observado por la configuración *bypass* a 1000μF es suficientemente pequeño como para no afectar a las mediciones del sensor.
Configuraciones más exhaustivas podrán estudiarse como trabajo futuro con la introducción de un filtrado mayor o el aislamiento de estas dos ramas con el operacional en modo seguidor de tensión de la siguiente sección.

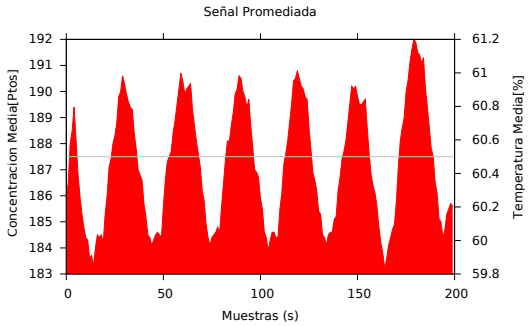
Operacional (trabajo futuro)

Se estudia la posibilidad de introducir una etapa de aislamiento para eliminar posibles *glitches* en el sistema debidos al acoplo de posibles etapas consecutivas. Para ello nos servimos del amplificador en circuito seguidor de tensión que gracias a su alta impedancia de salida aislará etapas posteriores.
La selección del operacional se realizo por los siguientes parámetros: alimentación, ya que disponemos únicamente de 5 voltios para alimentar nuestro circuito; margen de funcionamiento entrada y salida, el comportamiento *Rail to Rail*, definido por el alcance de valores límite de salida equivalente a los de entrada, en principio este comportamiento era deseable para el uso del operacional en modo sumador utilizado en el circuito de calentamiento.

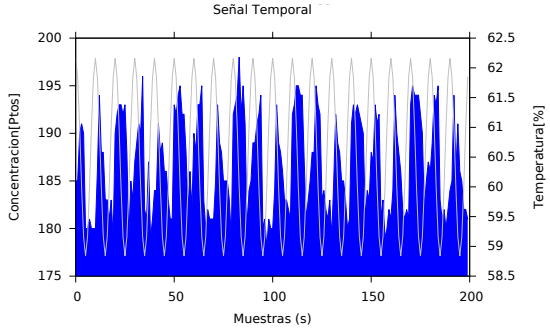
La figura 5.5 nos muestra el resultado de la aplicación de un control senoidal de la temperatura, las imágenes muestran una comparativa entre la señal salida del sensor y la señal tras la etapa de aislamiento tanto en muestras del sensor como en los valores promedios.



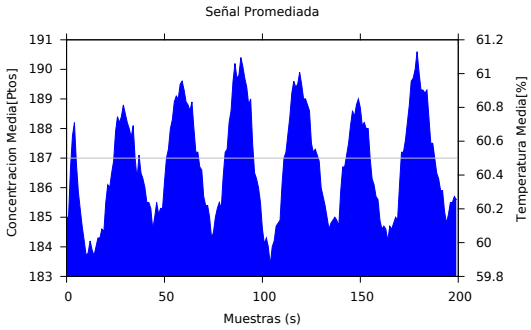
(a) Señal temporal obtenida del sensor(Vrl).



(b) Señal promediada obtenida del sensor(Vrl).



(c) Señal temporal obtenida del operacional(Varl).



(d) Señal promediada obtenida del operacional(Varl).

Figura 5.5: Señales salida del circuito divisor de tensión que conforman el valor de muestreo del sensor en la figura 5.6. En el segundo caso se aplica a su salida el circuito amplificador TLV2462 para observar su comportamiento. Señales idénticas, el circuito es compatible con el microcontrolador.

Realizando el análisis de la dispersión se concluye finalmente que funciona correctamente ya que los valores son idénticos, el microcontrolador tiene aislada su entrada analógica por lo que se descarta la utilización en el modo aquí expuesto. Destacamos que el uso de este amplificador si puede ser utilizable para el procesamiento posterior de las señales del sistema en otro tipo de trabajos.

Un posterior amplificado con control selectivo del rango de medición para obtener resoluciones mayores podría ser una aplicación interesante en el campo de las narices electrónicas para el cual deberíamos de utilizar este circuito en el sistema propuesto y con este tipo de microcontrolador.

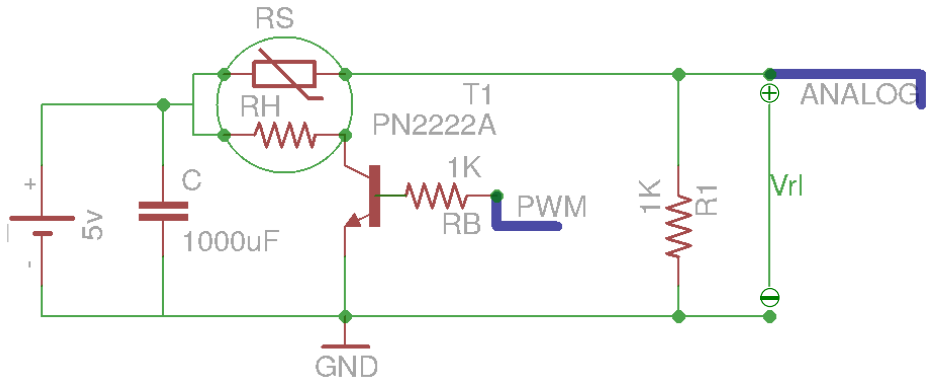


Figura 5.6: Circuito de acondicionamiento final con transistor PN2222A y condensador de 1000μF en régimen de *bypass* del sensor Figaro. Linea de alimentación equivalente a Vc. Red de divisor de tensión a la entrada analógica. Red de calentador del sensor.

5.3.3. Medición de la calidad del sistema

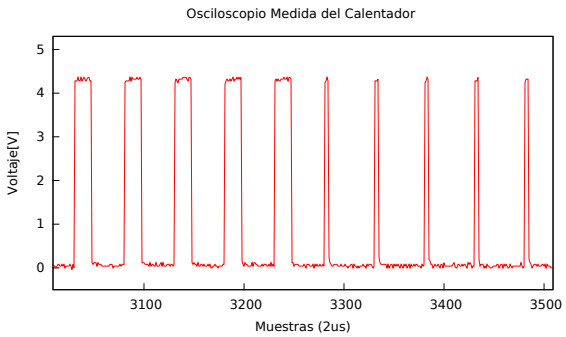
La valoración del correcto funcionamiento del sistema en las dinámicas de control que vamos a utilizar en el algoritmo desarrollado es una parte fundamental para eliminar errores debidos al sistema en el desarrollo del algoritmo.

Calefacción del sensor

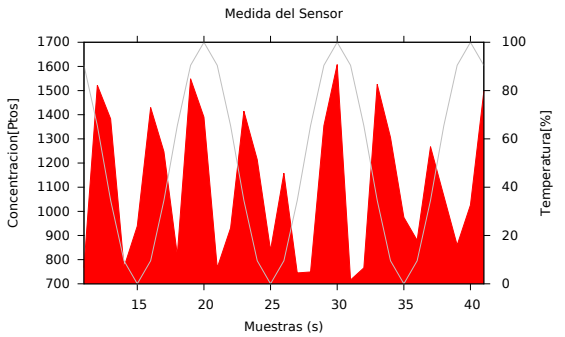
Elemento fundamental en el algoritmo de búsqueda debido a la necesidad de un control exhaustivo de la temperatura de calentamiento del sensor. El transistor es utilizado como elemento intermedio en el control de la red de calentamiento en el circuito 5.6 y es su correcto funcionamiento el que se expondrá en las siguientes pruebas.

Nuestro sistema se expondrá a la tensión de calentamiento en los modos que normalmente se utilizarán para la explotación de los algoritmos de localización:

- **Señal de tensión periódica.** La señal senoidal de temperatura es una parte fundamental del algoritmo permitiendo mediante el promedio de la señal conseguir valores más estables reduciendo el factor sensibilidad cruzada entre sensores. Por lo tanto será de vital importancia medir la correcta obtención de señales periódicas de tensión que serán aplicadas al calefactor.



(a) Señal obtenida del calefactor(Vh)

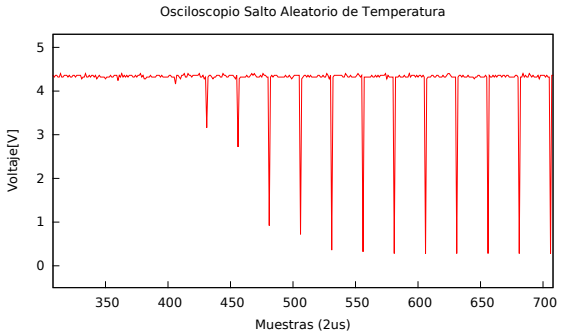


(b) Señal obtenida del sensor(Vrl)

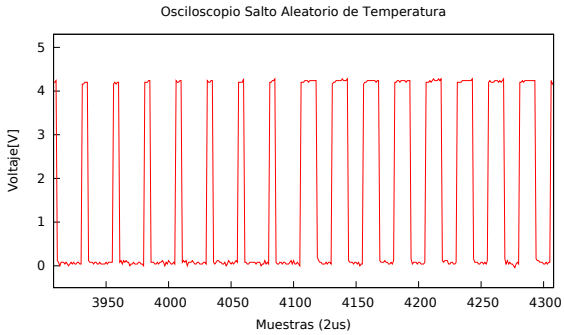
Figura 5.7: Disminución de la tensión por salto periódico en señal de variabilidad máxima (0 %a100 %). Cada una de las muestras tomadas en la figura a representa 2 μsegundos de tiempo.

En la figura 5.7, que modeliza este comportamiento, no se observan picos, cambios bruscos o rizado adicional en la señal de calefacción.

- **Salto de tensión aleatorio.** Debido a la naturaleza adaptativa del algoritmo de búsqueda desarrollado existe un factor de cambio en la temperatura debida a incrementos en la concentración medida de los sensores. Este comportamiento es simulado mediante la realización de saltos aleatorios en la temperatura.
  En la figura 5.8 se pueden apreciar dos figuras de salto aleatorio en los casos más extremos, en el peor de los casos vemos señales intermedias durante un tiempo despreciable respecto el muestreo de la señal.



(a) Señal obtenida desde la tensión máxima(Vh)



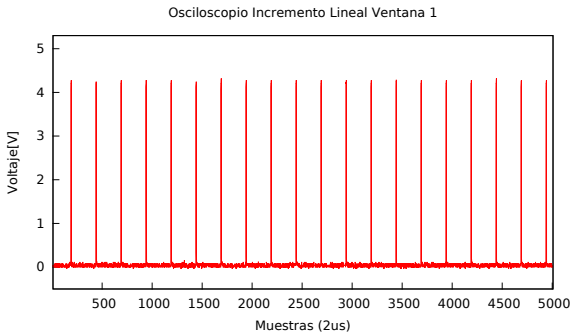
(b) Señal obtenida desde la tensión intermedia(Vh)

Figura 5.8: Disminución de la tensión por salto periódico en señal de variabilidad máxima (0 %a100 %). Cada una de las muestras tomadas representa 2 μsegundos de tiempo.

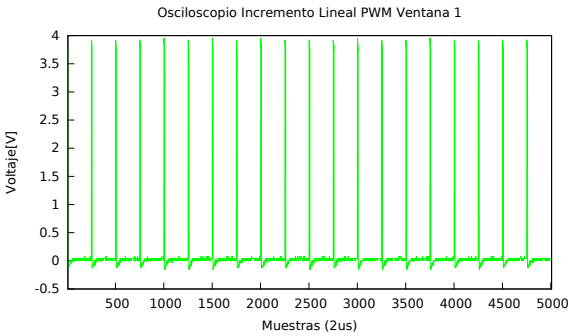
- **Incremento lineal de tensión.** Comportamiento utilizado como limpieza del sensor en el inicio del algoritmo de localización. El sobrecalentamiento del sensor a través de la aplicación en el calentador de una señal de tensión suficientemente alta da lugar a una eliminación de las partículas adheridas a la placa interna sobre la que se mide la conductancia.
  Macroscópicamente, se obtiene una curva de tensión lineal de 0 a 5 voltios como vemos en la figura anterior 4.5, sin embargo, al ser la señal procedente del PWM de la placa microcontroladora BBB, microscópicamente se puede observar el cambio en el ancho del pulso. En ventanas proporcionales a la frecuencia del PWM(2Khz) es donde se podrán observar sus efectos como observamos en la figura 5.9.

  A efectos prácticos observamos como la señal salida (PWM) de la placa está limitada a 3.3 voltios debido al rizado mientras que la señal procedente del calentador una vez situado el transistor en modo de amplificador se mantiene en el régimen de alimentación que se sitúa entorno a los 5 voltios.

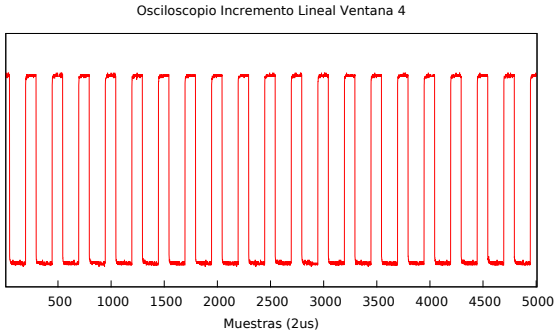
El sistema de calentamiento por tanto cumple con las expectativas exigidas en el diseño inicial. El control de voltaje es tan preciso como para cambiar el ancho del pulso de ciclo a ciclo y representar el seno de la señal del calentador. Además es capaz de realizar cambios de voltaje entre los límites inferior y superior del rango de trabajo sin aparecer señales intermedias inestables que puedan afectar al calentamiento de la placa interna del sensor.
Por último, la modificación del rango de trabajo es trasladado de forma efectiva para utilizar los valores de 0 a 5 voltios disminuyendo la resolución del sistema de forma inevitablemente.



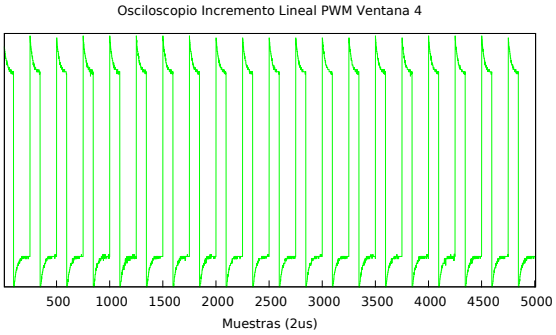
(a) Inicio Vh.



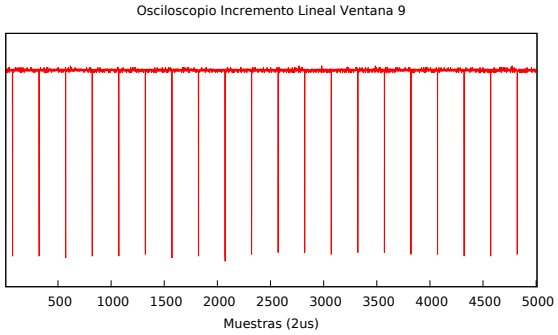
(b) Inicio PWM.



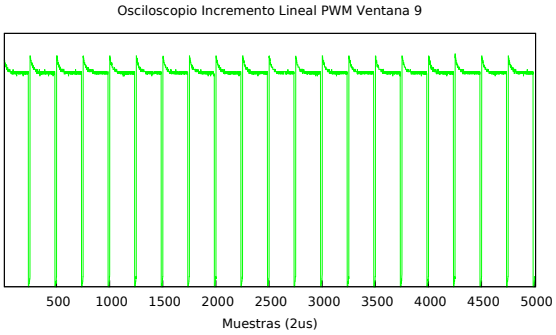
(c) Ventana 4 Vh.



(d) Ventana 4 PWM.



(e) Ventana 9 Vh.



(f) Ventana 9 PWM.

Figura 5.9: Señal de tensión a incrementos lineales del sensor de 0 a 5 voltios. Apreciación del incremento lineal del ancho del pulso. Se contrasta el pulso del pin PWM de la placa, resaltada en verde(PWM); con respecto la señal de tensión aplicada en el calentador(Vh), señal resaltada en rojo. Cada una de las muestras tomadas de las figuras representa 2 μsegundos de tiempo.

Ruido del sistema

- **Factor de ruido.** El ruido del sistema final viene caracterizado por la fuente de alimentación, ya que el resto de componentes son dependientes del sensor y por lo tanto no es posible discriminar entre el factor de ruido electrónico y la variación real de la conductancia que es la medición misma.
  La figura 5.10 representa la fuente de alimentación sometida a los procesos de saltos aleatorios de temperatura que suponen los cambios más extremos en el calentador. En la misma se observan dos frecuencias de ruido. Tomamos los valores de alta frecuencia como despreciables respecto los valores del muestro situados entorno 1 hercio.

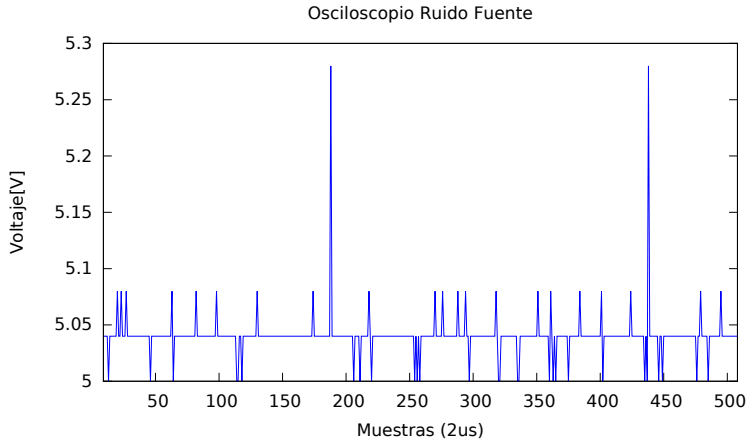


Figura 5.10: Señal de tensión de Alimentación con un único sensor (Vc). Apreciación de dos frecuencias de ruido. Cada una de las muestras tomadas representa 2 μsegundos de tiempo.

- **Medida de desviación.** Al igual que analizamos en el caso de la primera versión del sistema, comprobaremos los datos de la desviación estándar para valores de temperatura. Observamos como se ha disminuido considerablemente el ruido de la señal respecto los valores de las narices v1 de la tabla 5.2. Estas narices formarán el sistema integrado del que hablamos a lo largo de todo el proyecto y las señales de muestreo pueden ser observadas en detalle en el estudio de la modulación sinusoidal 4.3.2.

Temperatura (%)	σ (Ptos Concentracion)
20	1,61
30	1,52
*40	1,61
*50	1,50
*60	1,36
70	1,29
80	1,13
90	0,79

Cuadro 5.3: Medida de dispersión media de la toma de muestras de los sensores que conforman el sistema a temperatura constante. Posible comparativa de temperaturas equivalentes en las medidas marcadas por * con tabla de narices v1 5.2.

6

Sistemas olfativos artificiales desarrollados

6.1. Introducción

En el contexto de este trabajo definimos un sistema olfativo artificial como el conjunto de dispositivos encargados del muestreo, procesado y análisis de las señales de naturaleza difusiva que caracterizan la dispersión de las fuentes odorantes. De esta manera éstos sistemas están conformados por dos grandes grupos:

1. Narices electrónicas, que son dispositivos capaces de trasladar los estímulos de las partículas que conforman los odorantes al ámbito de las señales eléctricas. También puede incluirse aquí las estrategias de procesado de la información obtenida, ya que el acondicionamiento electrónico es un tipo de procesado de señal. A su vez, pueden estar formadas por las estrategias software de control que como hemos demostrado anteriormente ejercen un control activo para el filtrado del ruido.

2. Protocolos y sistemas ajenos a las narices electrónicas que crean una única unidad con el objetivo de la localización de fuente. Los sistemas olfativos artificiales también están compuestos por este tipo de metodologías:

    - Dispositivos externos. La plataforma robótica conforma un sistema externo a las narices electrónicas que permite la explotación del entorno mediante el traslado de las narices. Los sistemas olfativos artificiales también introducen frecuentemente otro tipo de sensores en sus estrategias de localización.
    - Protocolos de utilización. Para el correcto funcionamiento de las narices electrónicas es necesario la utilización de protocolos concretos que permitan explotar la información del entorno. En concreto el efecto de histéresis de este sensor o la utilización de varias narices electrónicas para la toma de decisión hacen que esta parte sea crítica en el desarrollo del sistema olfativo artificial.
    - Algoritmos de localización. Las estrategias que guían el movimiento y localización de las fuentes de odorante es la última pieza que completa el sistema con el objetivo concreto de la localización.

Dado el desarrollo de dos sistemas para la ejecución de algoritmos de localización de fuentes de odorante presentaremos este capítulo para describir la integración de cada uno de ellos y las partes que los componen. De igual manera nos permitirá establecer diferencias y similitudes así como las motivaciones de cada uno de los pasos establecidos en el diseño final de los sistemas.

Dividiremos el capítulo según los sistemas, por un lado el sistema Olus, caracterizado por los encapsulados; por otro lado el sistema integrado, resultado final del acondicionamiento de los sensores para aplicar las técnicas de modulación en temperatura del sistema sensorial en la plataforma robótica; finalmente se explicará con detalle el software que comparten ambos sistemas.

6.2. Sistema Olus

Sistema inicial formado por la nariz electrónica diseñada en [8] para el desarrollo de estrategias de búsqueda de odorantes. En esta primera aproximación se optó por diseñar un sistema en el cuál se pudiera comenzar a experimentar con las estrategias adaptativas de búsqueda de odorantes. Por lo tanto, enfocados en el centro sensorial, se identificó cual sería la arquitectura óptima para la comunicación entre las unidades.

En la figura 6.1 se observa la representación del sistema final. Cada unidad tiene un centro propio de procesamiento por lo que la integración del sistema tiene como principal desafío el establecer una arquitectura de comunicación que permita la toma de decisiones entre la unidad sensorial, representada por las narices Olus, y aplicada sobre la unidad motora, representada desde la placa microcontroladora BBB de la plataforma robótica.
Detallaremos cada unidad que compone el sistema, la comunicación implementada y el software encargado del correcto funcionamiento del mismo.

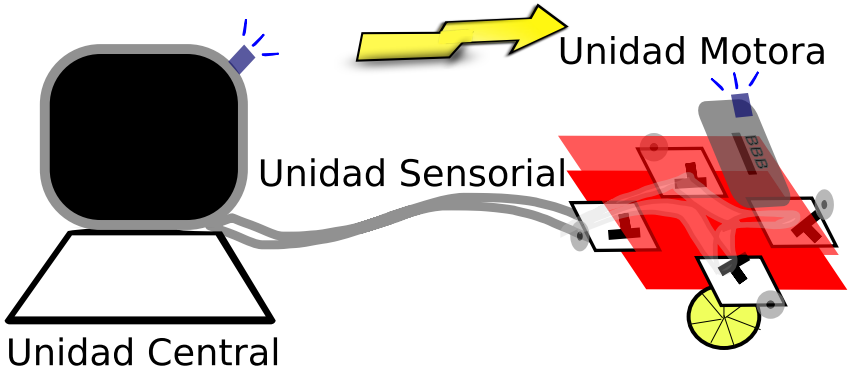


Figura 6.1: Sistema Olus. Tres unidades diferenciadas en la figura con sus unidades de procesamiento independientes interconectadas entre si.

6.2.1. Unidad sensorial

Unidad encargada de la captación de odorantes. En este caso, el sistema es el denominado Olus, esquemático del mismo en el anexo A.2.

Hardware

En el mismo se pueden distinguir tres tipos de componentes:

- **Sensores.** El encapsulado Olus está diseñado para el acondicionamiento de un sensor quimiorresistivo tipo MOS, en este caso utilizaremos el modelo TGS2611 del fabricante Fígaro y un sensor de temperatura TC1047 soldado debajo del mismo. Esta es la terminación sensorial del sistema.

- **Componentes de acondicionamiento.** Para el acondicionamiento del sistema son necesarios componentes electrónicos que permitan procesar las señales obtenidas de los sensores en el núcleo de procesamiento. Circuitos integrados de operacionales con referencia LMC6484 o transistores tipo MOSFET son algunos de los componentes con esta función.

- **Componente de procesamiento.** Los dos microprocesadores PIC18LF4550 que están integrados en el encapsulado tienen distintas finalidades.
  Uno de ellos estará dedicado a la captación de odorantes. En particular, se conectará con las redes eléctricas de acondicionamiento y obtención de la señal del sensor tipo TGS para mantener el calentamiento de forma continuada desde la llegada de corriente a sus pines.
  El otro es usado como expansión, tiene como finalidad el procesamiento de sensores externos al encapsulado. Está conectado al zócalo de expansión de forma que se puedan integrar en el mismo sistema distintos sensores para crear un sistema completo.

- **Componentes de comunicación.** Al crear un encapsulado independiente es necesario el establecer un medio de comunicación con el exterior para que, como en nuestro caso, se pueda establecer una estrategia de control sin reprogramar el pic para cada caso. El encapsulado cuenta tanto con una interfaz USB, marcado en el diseño adjunto como bridge MCP2200 y cuyo software de comunicación aun no está integrado en el PIC, como con pines de comunicación serie.

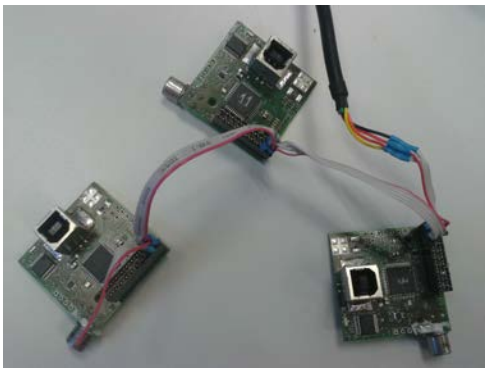


Figura 6.2: Unidad sensorial del sistema Olus. Conexión serie de tres encapsulados con salida USB en el otro extremo. Los tres módulos diferenciados en la figura son tres narices electrónicas independientes con direcciones de red asignadas.

Comunicación

La comunicación con la unidad sensorial se establece mediante el conector de salida del esquemático. Los cuatro pines iniciales son los reservados para la conexión con la nariz a través del protocolo FTDI RS485. Los pines reservados están denominados con los símbolos A+, B-, V_{DD} y GND en el esquemático y son la guía de referencia para conectar el cable serie de conexión.

El protocolo interno de comunicación diseñado en [1] permite la conexión en serie de todos los encapsulados tal y como observamos en la figura 6.2. El microprocesador tiene una dirección interna identificativa que permite, análogamente a una red de área local, procesar únicamente los paquetes que van dirigidos a el mismo desde una topología tipo bus de datos.

La comunicación con las narices se realiza a través del siguiente protocolo interno de envio de paquetes. Las tramas de envío y recepción están prefijadas según lo establecido en la imagen 6.3. La trama la componen 10 bytes de información, la conexión es bidireccional aunque la recepción de datos del sistema Olus se realiza tras el envío de una trama de petición. Las tramas están definidas por los siguientes campos:

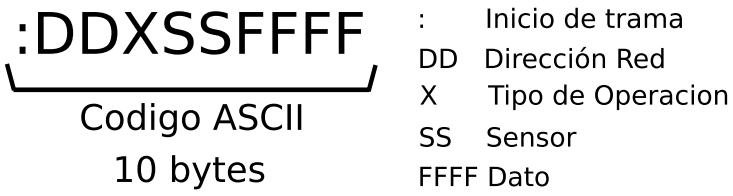


Figura 6.3: Trama de comunicación del sistema Olus. 10 bytes total de paquete.Inicio de trama, dirección de red del encapsulado, tipo de operación, dirección local en el encapsulado del sensor y dato son las subtramas que componen el paquete.

- **:** Es el carácter de inicio de trama. Delimitador de trama.
- **DD** Espacio reservado para la dirección de red asignada a cada encapsulado. Este una dirección única asignada mediante la programación del PIC.
- **X** Tipo de operación a realizar. Existen dos tipos de tramas de comunicación: las tramas de solicitud de información están designadas con el número 3. Las tramas de modificación de parámetros de los sensores se designan por un 6 en este campo reservado.
- **SS** Sensor dentro del encapsulado al que solicitamos el tipo de operación. Como hemos comentado con anterioridad los encapsulados están diseñados para la integración de distintos tipos de sensores en sus pines. En el PIC están asignadas estas direcciones de forma predefinida. El sensor TGS2611 está definido por el codigo 01 para el muestreo de información y 05 para su calentamiento.
- **FFFF** Espacio reservado para el dato. Sea el caso que queremos realizar una escritura de parámetro en sensor, será posteriormente a la petición el espacio reservado para el dato. Si solicitamos información de uno de los pines en la trama de vuelta obtendremos el dato delimitado en este espacio. Los valores de lectura escritura están digitalizados en el PIC y tienen un rango de 0 a 1024 representando el 0 y los 5 voltios respectivamente.

En la figura 6.4 exponemos un ejemplo de comunicación entre la ejecución del código y el encapsulado Olus, en éste queremos obtener y modificar datos del sensor TGS2611 con dirección de red 11.
En la figura se puede ver gráficamente la siguiente comunicación establecida:

1. El codigo solicita la lectura a la dirección de red 11 del sensor 5 que corresponde con el calentador del sensor.
2. El encapsulado le contesta dando su dirección de red que el valor actual está fijado en el calentador a 555.
3. El código escribe en la misma unidad de red 11 que fije el valor del calentador al 340.
4. De nuevo el código solicita información a la dirección de red 11, esta vez acerca de la señal obtenida en el sensor TGS para observar como ha modificado el comportamiento en el sensor dado el cambio de temperatura.
5. El encapsulado le contesta con un valor en el sensor de 188.

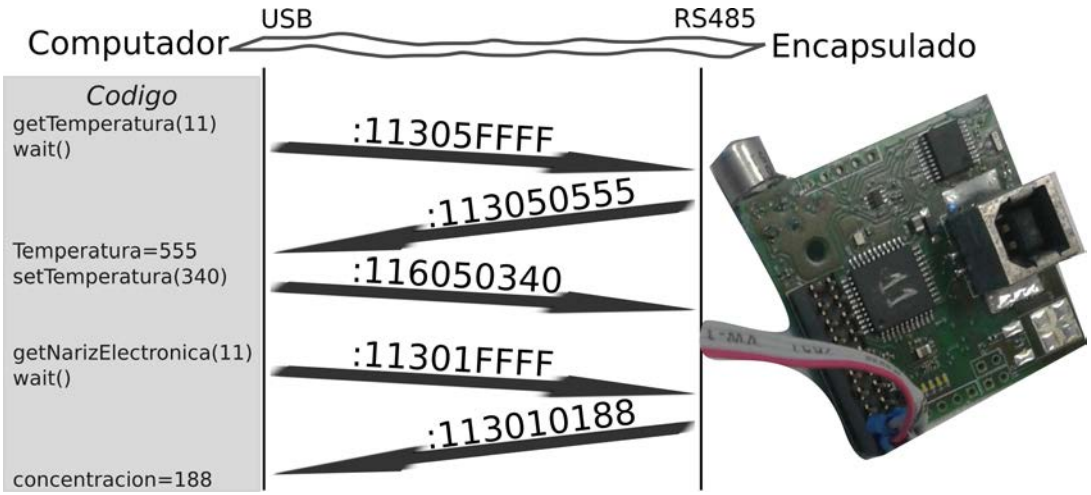


Figura 6.4: Envío de tramas de comunicación entre codigo ejemplo ejecutado en computador y encapsulado Olus. Las tramas enviadas se corresponden con la obtención del valor de concentración medida en el sensor TGS2611, la lectura del actual valor de temperatura en el calentador y la escritura de un valor definido de temperatura.

6.2.2. Unidad motora

Plataforma robotizada con la función de trasladar el sistema sensorial en función de la instrucción recibida. El hardware de la unidad motora está controlado por la tarjeta microcontroladora BBB. En el capítulo 3 se explican los detalles tanto de los componentes y construcción de la plataforma, sección 3.3, como de la placa microcontroladora que controla toda la unidad, sección 3.2.

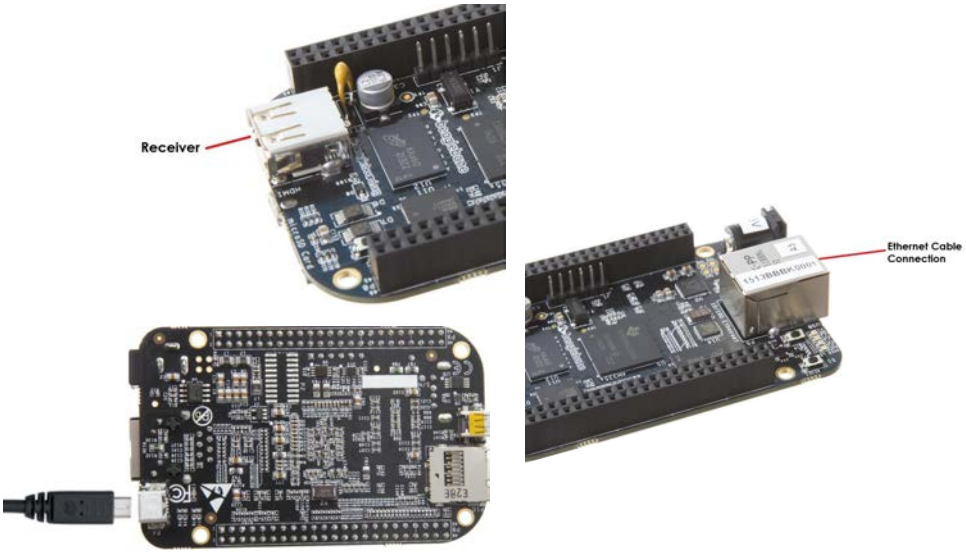


Figura 6.5: Modos de conexión en la placa microcontroladora BBB. Conector serie microUSB, conector USB y uso con adaptador de red o a través de cable de ethernet. Imágenes obtenidas de [3].

Para la comunicación de la unidad en el sistema, existen tres modos de conexión con el modulo de procesamiento BBB expuestos en la figura 6.5:

- **Conector microUSB.** Conector del sistema a través de cable USB, es el modo de comunicación por defecto.
- **Cable de red Ethernet.** Tras configurar los parámetros de red podemos conectarnos a través de cable LAN con la tarjeta BBB. Para ello, se reserva este adaptador en el sistema.
- **Adaptador USB.** El único modo de conexión inalámbrica es a través de un adaptador WIFI acoplado en la entrada USB del sistema. Los parámetros de red son configurables de la misma forma que con el cable de red desde la interfaz de conexiones del sistema operativo.

En todos los casos utilizamos el protocolo ssh para conectar de forma remota con la placa microcontroladora. Tanto si conectamos a través del cable microUSB como al conectarnos en área local se puede utilizar el dominio *beaglebone.local* para conectarse a través del protocolo ssh.
En nuestro caso, utilizamos tanto el cable de red como un adaptador modelo DWA-121 con D-Link como fabricante para conectarnos a través de la red. La conexión inalámbrica sería el medio ideal para iniciar los algoritmos de forma remota ya que permite a la plataforma robótica trasladarse libremente.

La integración final de la unidad sensorial y motora en el sistema quedaría como observamos en la imagen de la figura 6.6.

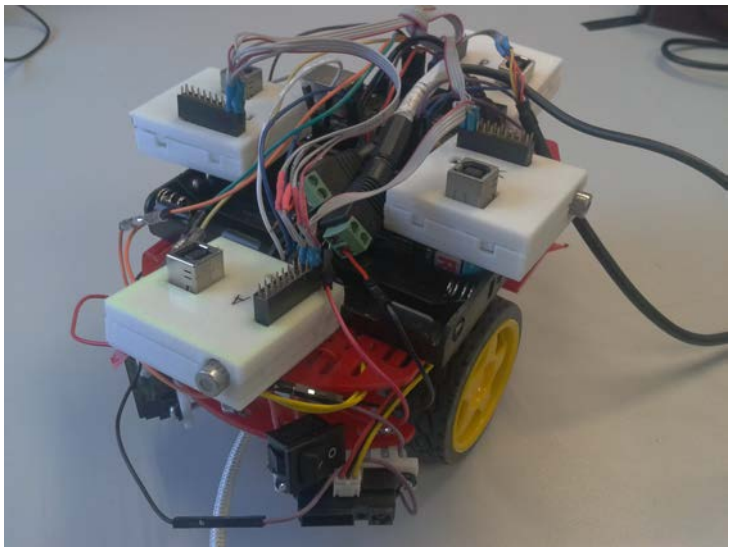


Figura 6.6: Integración de la unidad sensorial en la unidad motora. Debido a la falta de espacio, los sensores trasversales se adhieren a la superficie de las baterías.

6.2.3. Unidad central

Unidad básica en el procesado de información y el modulo intermedio de conexión entre las demás unidades del sistema. La unidad central, o computador, es la unidad que ejecuta los algoritmos y toma las decisiones que posteriormente transcribirá en el código de ejecución de la unidad motora tal y como se aprecia en la figura 6.8 del sistema completo.
Por lo tanto su tarea es la de ejecutar el software de procesado, representación y ensamblado entre las partes de los distintos sistemas que lo integran.

Procesado de señales

El software desarrollado para el muestreo del sistema Olus tiene como punto de partida las librerías desarrolladas por Fernando Herrero programadas para el sistema desarrollado citado en la bibliografía [2]. El sistema usado en aquel momento aún siendo diferente tiene como similitud el mismo núcleo PIC y por lo tanto el mismo protocolo de tramas desarrollado en [1] para la comunicación entre sistemas de la sección 6.2.1.

Partiendo de las librerías heredadas se desarrollaron los códigos en su lenguaje de programación nativo, C++. Tomando las librerías como modelo se desarrolló una nueva clase denominada *multinose* con objetivo de estructurar en la misma clase las particularidades, direcciones y tipos de sensores que conforman este encapsulado denominado Olus. La referencia al codigo de la clase se encuentra en el anexo C.2.1. Cabe destacar que este trabajo de reescritura no habría sido posible sin las clases previas implementadas en las librerías así como la ya probada con anterioridad configuración de conexión.
Los requisitos iniciales para compilar el código desarrollado para el sistema Olus son tanto el enlazado a las librerías de comunicación con las narices, como la instalación y enlace con las librerías *boost* [59], puesto que se utilizan sus métodos en las librerías heredadas.

El codigo desarrollado ha evolucionado a lo largo de este proyecto desde el código inicial de muestreo de las narices electrónicas referenciado como main en el anexo C.2.2. La evolución ha sido tanto en el aspecto de inclusión de lógicas para la localización de fuente, como en el procesado de las señales.

En este último sentido, queremos remarcar un detalle. Inicialmente el procesado de las medias obtenidas de los sensores se obtenían mediante ventanas no coincidentes con lo que obtuvimos en la primera etapa del proyecto señales con menor variabilidad.
Posteriormente, se estandarizo el sistema con el sistema integrado al procesar las medias con una ventana continua de promediado deslizante, este ultimo procesado se puede observar en la linea 189 del codigo: *main diferencial básico ventana solapante*, en el anexo C.2.3.

En el primero de los casos de la figura 6.7 encontramos que la señal promedio era diezmada con el factor del número de puntos de la función senoidal que marcaba el promedio de la señal de esta manera la ventana deslizante representa valores promedio con mayor información intermedia. Figura comparativa de procesado 6.7.

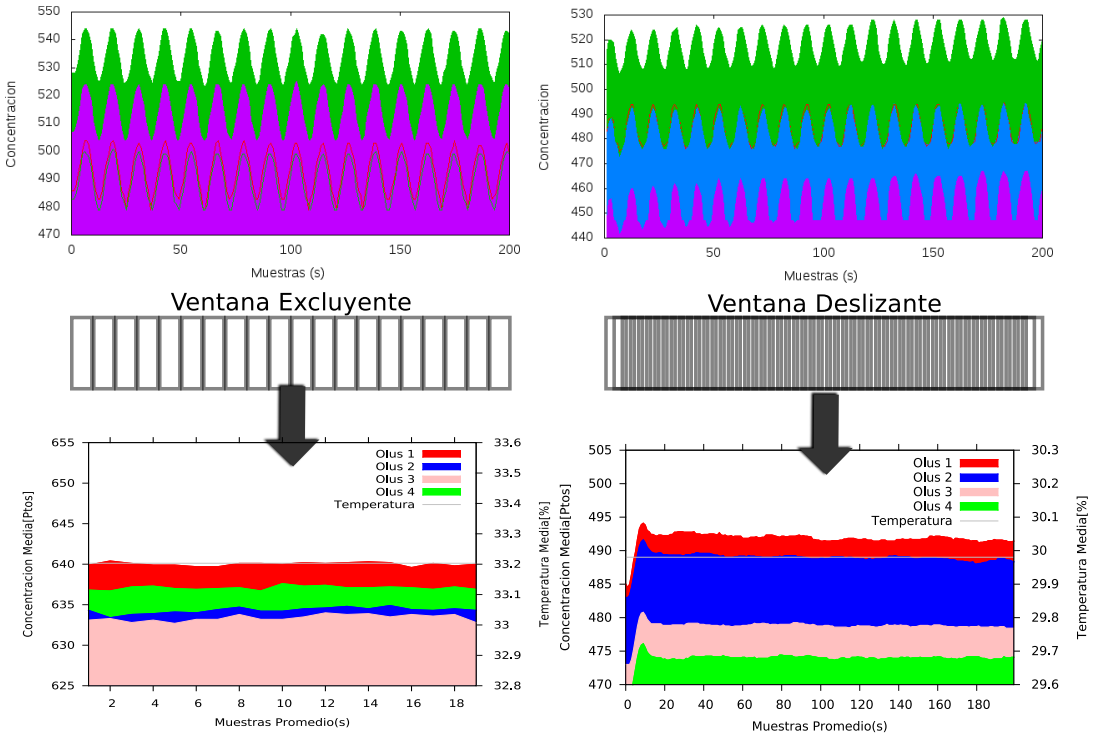


Figura 6.7: Ejemplo de procesado de señal en sistema Olus. Inicialmente se usaban ventanas excluyentes obteniendo muestras promedio diezmadas por el número de puntos de la señal. Mejora del sistema al introducir ventana solapante incrementando la información y por lo tanto la variabilidad de la señal. Las señales han sido modificadas variando los valores promedio originales para la mejor visualización de las señales procesadas. Los valores de concentración obtenidos no son dependientes ni de la temperatura ni del procesado de la señal.

Arquitectura del sistema

La tarea principal de la unidad central es la de encapsular las unidades que conforman el sistema para, estableciendo la comunicación entre las unidades, redireccionar el flujo de datos de forma correcta. El software encargado de esta tarea, por tanto, unirá cada uno de los códigos que controlan cada unidad.

El muestreo de los sensores se realiza mediante código C++ ejecutado en la unidad central, pero las decisiones que toma el mismo deben de salir del sistema para convertirse en entrada del código que se ejecutara en la plataforma robótica. Utilizaremos la salida estándar del software de muestreo de las narices, el archivo de texto. Leeremos la información contenida y tomaremos una decisión que debemos de ejecutar por conexión remota con la plataforma robótica. El lenguaje seleccionado para realizar este tipo de tarea será el lenguaje Bash.
Bash es el interprete de ordenes de la shell de la mayoría de sistemas operativos GNU/Linux. La realización de scripts en este lenguaje por lo tanto, es similar a la escritura mediante comandos de la shell. La lectura de archivos de toma de decisión y la conexión remota con la unidad motora es la tarea principal del script que hemos realizado para la búsqueda de fuentes de odorante.

En la figura 6.8 se puede observar claramente como es el encapsulado de funciones del sistema completo. El script principal situado en el medio de la figura, tendrá el control del loop de búsqueda, el código que lo conforma está en el anexo C.4.1. El subscript de muestreo ejecutará el código en C++ para la comunicación con las narices y tras la representación de los datos enviará la decisión tomada del procesado de datos a la plataforma que ejecutará el codigo de *move*, el código final de los subcripts se encuentra en el anexo C.4.2 y C.4.3.

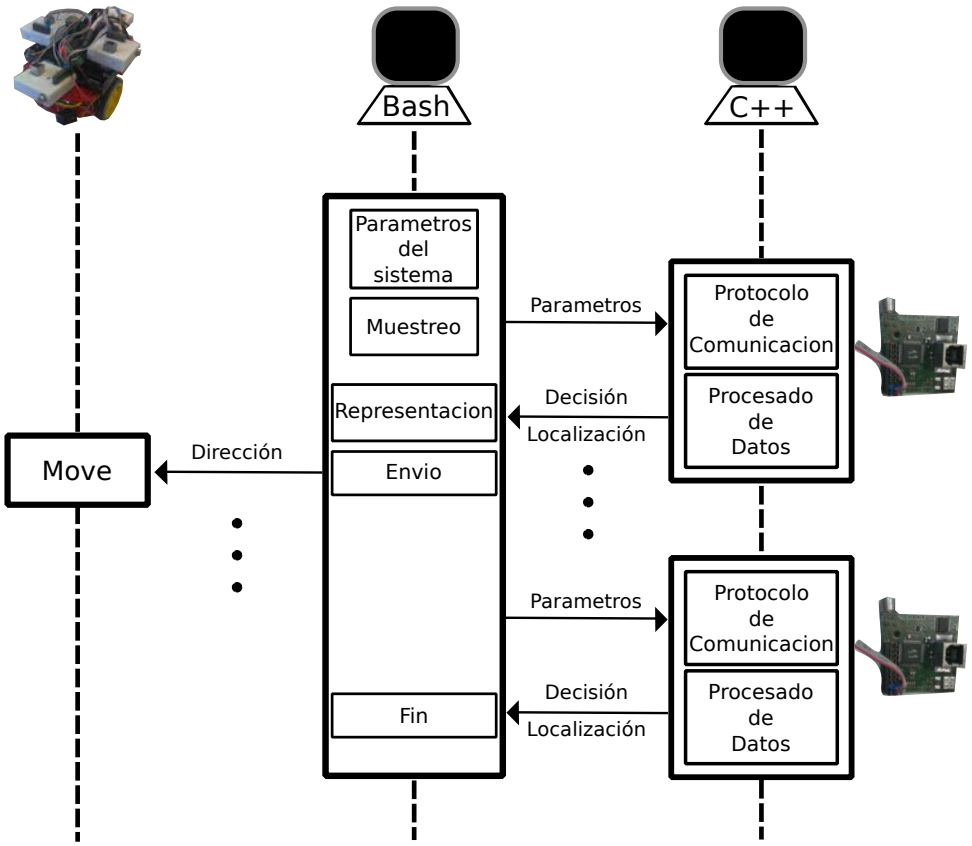


Figura 6.8: Representación de ejecución del software encargado de conectar los diversos sistemas. Desde la rutina principal se ejecuta el software de muestreo y se procesa la decisión para enviarla por conexión de red a través de ejecución en remoto.

6.2.4. Desventajas del sistema

- La utilización de distintos lenguajes de programación repercute en la eficiencia del sistema completo. El envío de parámetros compartidos de los algoritmos entre unidades convierte el sistema en poco flexible, la solución de crear scripts de código para realizar esta tarea de comunicación se convierte en la solución mejor posible para la integración de las partes.
- La comunicación con los encapsulados está caracterizada de tal forma que el cumplimiento del protocolo de comunicación por tramas conlleva un esfuerzo extra a la hora de implementar el sistema. Si bien es cierto que la utilización de las librerías permite la abstracción de la capa de comunicación, las mismas son únicamente utilizables en lenguaje de programación C++ y su instalación no es trivial.
- La utilización del PIC del encapsulado Olus resulta innecesario teniendo el microcontrolador BBB integrado en la plataforma. El sistema integrado acondicionará los sensores para aprovechar la tarjeta microcontroladora como unidad de procesamiento única del sistema.
- La movilidad del sistema, característica más que necesaria en la implementación de los algoritmos de localización, está penalizada por la utilización del cable de conexión serie a protocolo RS485 utilizado en la unidad sensorial.

6.3. Sistema integrado

Sistema desarrollado para la integración de los módulos que conforman el sistema de localización de fuentes de odorante. Desarrollado tomando como punto de partida el sistema Olus pero acondicionando los sensores de forma externa para la modulación de temperatura, presentando la placa BBB como única unidad de control y procesamiento.
Constituye la solución a los problemas planteados en el sistema Olus, sección 6.2.4. La placa microcontroladora con Linux embebido permite libertad en la selección del entorno de programación y los pines de conexión externa otorgan de gran potencial de integración con todo tipo de sensores.

6.3.1. Componentes del sistema

- **Placa microcontroladora.** Unidad de control y procesamiento de información obtenido por las narices electrónicas. El detalle del mismo se encuentra en la sección 3.2. La carga de procesamiento será mucho mayor en este sistema en comparación con el sistema Olus ya que se encargará de ejecutar los algoritmos de búsqueda.

- **Plataforma robótica.** Plataforma móvil sobre la que se trasladaran las narices electrónicas. Dedicamos una sección a sus componentes y ensamblado, sección 3.3.

- **Nariz acondicionada.** Las narices electrónicas que se integrarán en esta plataforma son la última evolución del diseño de las mismas denominado en el capítulo 5 como narices v2, sección 5.3. Estas narices están compuestas por un sensor modelo TGS de Fígaro acondicionado para su uso con técnicas de modulación de temperatura de calefacción.

- **Otros sensores.** Se estudió la posibilidad de introducir otro tipo de sensores que completaran el sistema de captación de odorantes. Algunos estudios referenciados en el estado del arte, sección 2.3, analizan el problema implementando anemómetros, brújulas electrónicas o sensores de humedad y temperatura. La distinguida deriva que podemos observar en la hoja de referencia del fabricante con respecto la temperatura y humedad crea la necesidad de integrar en el sistema un sensor de medición de estos parámetros, la referencia del componente final es am2302 y la imagen del mismo en la figura 6.9. El software de utilización del sensor está expuesto en el anexo sección C.5, se ha utilizado código externo para el protocolo de comunicación.

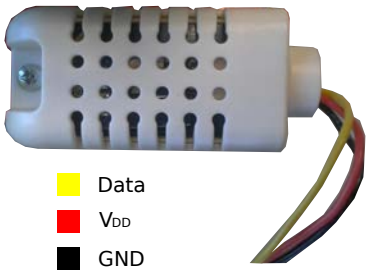


Figura 6.9: Sensor de humedad y temperatura am2302. Componente del sistema integrado. La humedad y temperatura son factores fundamentales en los sensores dada su deriva. Integración en el sistema para discriminar errores debidos a la deriva de los sensores y como objeto de posteriores estudios.

6.3.2. Integración

La integración física de los componentes paso por la impresión de algunas estructuras 3D para facilitar la sujeción de los mismos, estructuras diseñadas en la sección 3.3.2.

La integración eléctrica en la placa microcontroladora se realizo según la tabla de conexiones 6.1, en el diseño de las narices electrónicas se integró en los pines de entrada/salida del sistema un zócalo doble conectado de forma interna para facilitar el conexionado físico creando así una referencia común de alimentación y tierra. Es por esto que en la tabla de conexiones aparecen conexiones dobles para el mismo tipo de pin.
Tanto esta red eléctrica de alimentación común a las narices electrónicas como la referencia de cada una de ellas en el sistema están representados en la figura 6.10. La tabla de conexiones final del sistema integrado queda completa al unirla a la figura de conexiones de la plataforma robótica de la sección 3.12.

Tabla de Conexiones						
	Pin	Nariz1	Nariz2	Nariz3	Nariz4	BBB
Nariz 1	V_{DD}	X	V_{DD}	X	X	V_{DD}
	ANALOG	X	X	X	X	P9 (38)
	PWM	X	X	X	X	P8 (13)
	GND	X	GND	X	X	GND
Nariz 2	V_{DD}	V_{DD}	X	V_{DD}	X	X
	ANALOG	X	X	X	X	P9 (37)
	PWM	X	X	X	X	P9 (21)
	GND	GND	X	GND	X	X
Nariz 3	V_{DD}	X	V_{DD}	X	V_{DD}	X
	ANALOG	X	X	X	X	P9 (33)
	PWM	X	X	X	X	P8 (19)
	GND	X	GND	X	GND	X
Nariz 4	V_{DD}	X	X	V_{DD}	X	X
	ANALOG	X	X	X	X	P9 (39)
	PWM	X	X	X	X	P9 (22)
	GND	X	X	GND	X	X

Cuadro 6.1: Tabla de conexiones en la integración de las narices electrónicas v2 en placa microcontroladora BBB. Las unidades v2 poseen dos pines por cada tipo de pin conectados internamente para facilitar el conexionado. Se puede completar la tabla de conexiones con la figura 3.12.

6.3.3. Software

La integración física de la plataforma conlleva la integración software de la misma para poder explotar al máximo la eficiencia del mismo. La interrelación entre los diferentes componentes se ha realizado a través de la creación de hilos del sistema para diferenciar claramente cada una de las funciones y a su vez, poder mantener un espacio común de variables para mantengan el flujo de datos. Este tipo de software es el que corresponde al de los algoritmos desarrollados de localización de odorantes ya que necesitan todos los módulos del sistema. Todo lo relacionado con los algoritmos de localización estará detallado en la sección 7.3.

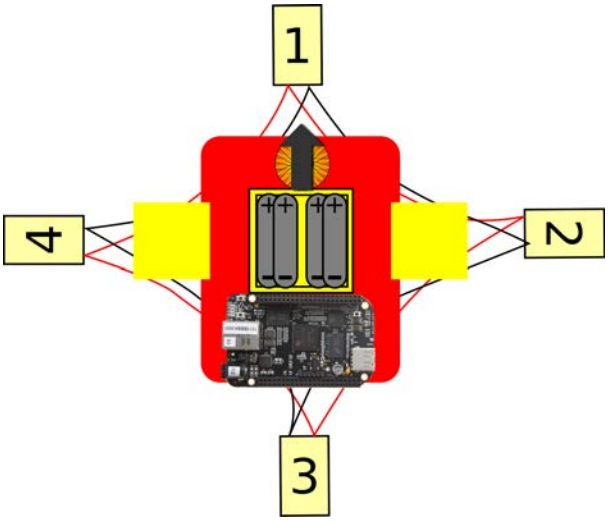


Figura 6.10: Integración de narices electrónicas desarrolladas v2 en la plataforma. El orden de las narices electrónicas corresponde tanto con el de los pines del sistema representado en la tabla de conexiones de la figura 6.1 como con los del software desarrollado del anexo C.6.3.

Existen códigos intermedios que no utilizan tecnología de hilos puesto que ejercen control sobre un único componente. Este tipo de códigos están asociados, análogamente en el sistema olus, a los scripts. Representan los protocolos previos o test de estabilidad y están íntimamente relacionados con el control previo de los sensores quimiorresistivos en la experimentación. En la sección 6.4.1 trataremos de explicar en que consisten estos códigos que representaran el inicio de la ejecución de los algoritmos de búsqueda de fuentes de odorante.

6.4. Software común

Se ha desarrollado software en común para ambos sistemas que conforman el inicio de los algoritmos de localización. Estos protocolos iniciales se han utilizado previamente a la experimentación, son una parte fundamental de ella y cada parte tienen un objetivo fundamental en el sistema:

1. La rampa de temperatura primer bloque del protocolo de inicio tiene por objetivo minimizar el efecto de histéresis, el calentamiento a alta temperatura elimina las particular adheridas y reinicia la sensibilidad.

2. Muestreo de señal con función sinusoidal, el segundo bloque del protocolo de inicio y tiene por objetivo obtener señales del sistema estables.

3. Ajuste de *offset* obtendrá el valor diferencia entre sensores de un mismo eje que tengan un valor constante a lo largo de una toma muestreal.

En conjunto conforman el inicio del protocolo de experimentación que se expondrá en el próximo capítulo y que está representado en la figura 6.11. En este caso haremos una distinción entre protocolo de inicio y el de ajuste de offset debido a que el protocolo de inicio es común para todo tipo de experimentación con narices tanto para una nariz individual como para el uso de varias simultáneamente. El protocolo de ajuste de *offset* solo es necesario cuando queremos comparar distintos sensores entre sí como es el caso de el algoritmo desarrollado.

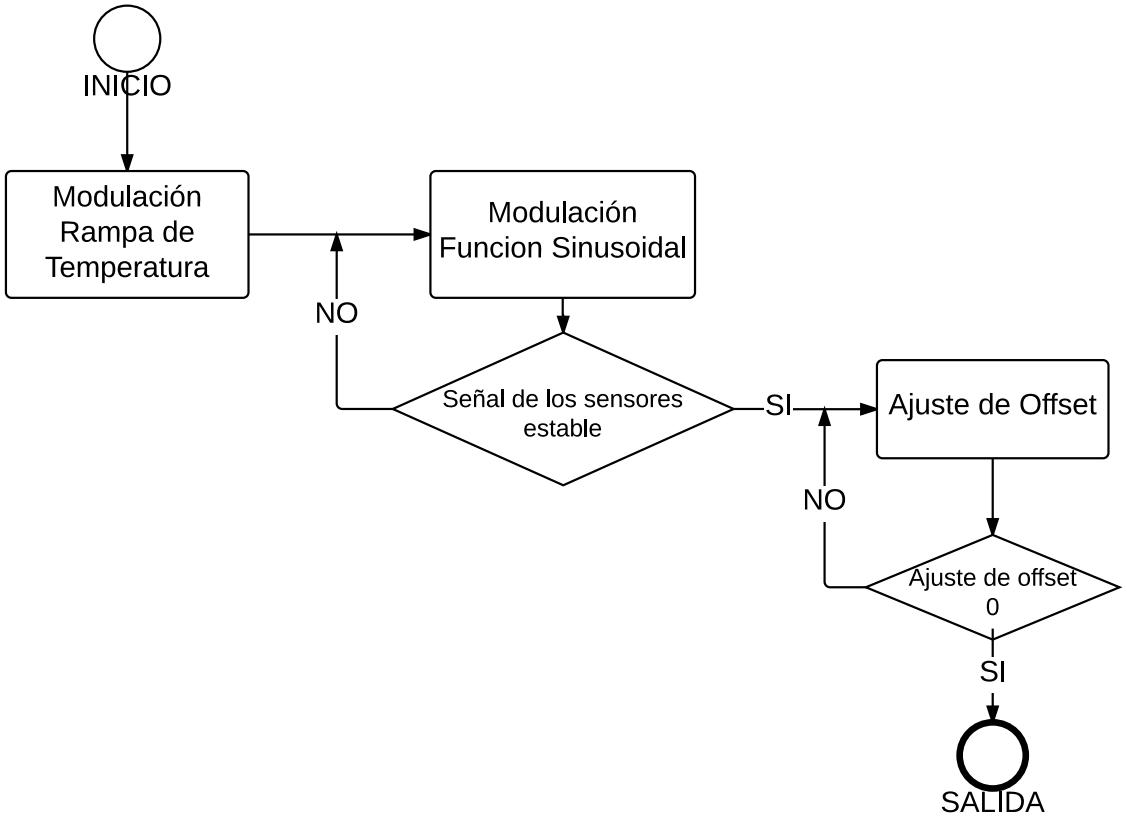


Figura 6.11: Diagrama de flujo del protocolo conjunto de experimentación en las narices electrónicas. Los primeros dos bloques del sistema representan el protocolo inicial de calentamiento utilizado en cualquier tipo de experimentación o análisis de señal proveniente de las narices electrónicas. El último bloque representa el ajuste de offset necesario para comparar más de un sensor del sistema y como en nuestro caso, procesar sus señales en conjunto.

6.4.1. Protocolo de inicio

Inicio del sistema para la captación de odorantes en los sensores. Este protocolo tiene como objetivo la minimización del efecto de histéresis provocado por sobreexposiciones de odorantes o estados no controlados anteriores del sistema. El protocolo disminuye la cantidad de partículas adheridas a la placa interna del sensor a través de una subida de voltaje en el calentador del sensor. Posteriormente se generará un muestreo continuo de los sensores hasta la estabilización de las señales a la temperatura deseada tal y como vemos en el primer bloque de la figura 6.11. El protocolo está formado por dos etapas representadas en la figura C.6.1, explicaremos a continuación cada una de ellas en detalle:

- **Calentamiento.** Etapa de calentamiento máximo del sensor en incrementos lineales descrita en la sección 4.3.1 de aplicación de técnicas modulatorias. Como describimos anteriormente permite limpiar los sensores de impurezas, esta aplicación tiene resultados satisfactorios en su objetivo de minimizar el efecto de histeresis.

- **Estabilización.** Tras la aplicación del calentamiento inicial los sensores quedan reiniciados con sensibilidad muy baja. En la segunda parte del protocolo estabilizaremos la sensibilidad de los sensores obteniendo muestras de forma continua mediante el control de temperatura básico modulatorio a temperatura media constante, subsección 4.3.2.
  La obtención de señales estables de los sensores es dependiente de otros parámetros externos al sistema y relacionadas con el entorno en el que se sitúa la plataforma. El valor de estabilidad de la señal es controlada mediante el cálculo de la tendencia. Este valor se estima de forma externa mediante la señal promedio como la diferencia entre las muestras

finales y el promedio de la señal completa.
Se estima que un valor de estabilidad explotable determinado de forma experimental se encuentra en la unidad, el tiempo medio del mismo se encuentra entorno a los 25 ciclos de 204 muestras que equivalen a 85 minutos en total. En la tabla 6.2 podemos observar un estudio de la evolución de este parámetro tras la primera parte del protocolo, visualmente las señales se corresponden con el primer y ultimo ciclo de la figura 6.12.

Cálculo de pendiente en señales promedio				
	Sensor1	Sensor2	Sensor3	Sensor4
Ciclo 1	12,72	7,73	15,71	10,77
Ciclo 12	1,44	1,54	1,73	2,98
Ciclo 25	0,66	1,31	1,23	1,48

Cuadro 6.2: Tabla de pendientes pertenecientes a una realización del protocolo de inicio de sensores. Obtención de valores pendiente o tendencias del sistema de menor valor que implican señales más estables de los sensores. Se calcula que entorno al ciclo 25 los sensores alcanzan la sensibilidad necesaria para la explotación en los algoritmos de búsqueda. Estos parámetros son dependientes del entorno en el que se sitúan los sensores.

El protocolo queda encapsulado en un único script de calentamiento definido por 4 parámetros: número de muestras de estabilización, temperatura de los sensores en cada uno de los ejes o dimensiones del algoritmo y la variable que indica si utilizamos el programa de calentamiento o solo muestreamos la señal. El código del protocolo para el caso del sistema integrado está expuesto en la sección del anexo C.6.1, el código del sistema Olus es equivalente y por lo tanto no ha sido anexionado.

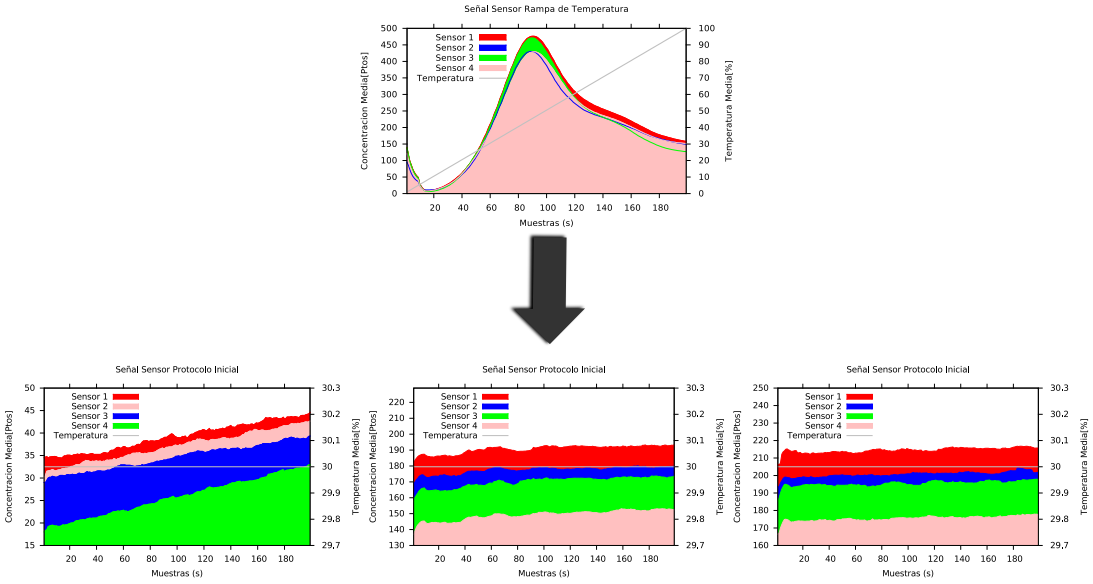


Figura 6.12: Protocolo inicial de calentamiento. Señales promedio de los sensores aplicando el protocolo inicial. Primera figura rampa de temperatura seguida de los ciclos 1 y 25 respectivamente. Análogo al comportamiento estudiado mediante la tabla 6.2 de pendientes observamos como la señal de los sensores pasa a un estado estable con el paso de los ciclos de muestreo.

6.4.2. Protocolo de ajuste de offset

Protocolo de ajuste de parámetros del sistema para la utilización de estrategias multinariz. El objetivo de este protocolo será la obtención de la señal diferencia entre sensores que posteriormente conformarán un eje del movimiento y por lo tanto sus señales serán comparadas entre sí. Es necesario para la utilización de los valores diferencia que marcarán el vector dirección ante la presencia de odorante. Representan el último bloque de código del diagrama de flujo de la figura 6.11,

Sabemos que la sensibilidad a los odorantes para cada sensor es diferente debido a los procesos de fabricación, pero además, el efecto de histéresis y las condiciones del medio hacen que no sea determinista ni siquiera entre ellos mismos. Es por esto que buscaremos los parámetros iniciales del sistema que tengan mejores características para el problema de la localización. En concreto, el valor previo deseado es la estabilidad del sistema.

En la búsqueda de los parámetros del sistema adecuados, necesitamos valores estables referidos a los valores diferenciales entre los sensores que queremos ajustar. Pero este valor, como hemos podido comprobar experimentalmente no es un valor reproducible, cada experimento posee un valor diferencial propio vinculado a una temperatura de calefacción. En conjunto, temperatura y *offset* entre sensores, representan una señal estable para un único experimento. Por lo tanto necesitamos tanto el valor de *offset* entre sensores como el valor de temperatura de calefacción para la obtención de la estabilidad previa en el sistema. Esto lo conseguimos mediante la modificación de temperatura diseñada para el algoritmo de localización. Se ejecutará esta modificación ajustando el offset por cada ejecución del bucle de 6.11.

El proceso descrito anteriormente se ejecutará hasta que la señal de *offset* entre sensores no varíe durante el muestreo de las señales. Aunque las lógicas de modificación de temperatura están relacionadas con el algoritmo de búsqueda el codigo del sistema integrado de ajuste de *offset* puede encontrarse en el anexo C.7.1.

6.4.3. Software de representación

Gracias a la utilización del sistema GNU/Linux embebido en la placa microcontroladora hemos podido migrar los scripts de representación del sistema Olus al sistema integrado. En ambos sistemas hemos utilizado el programa de representación Gnuplot que nos permite representar las señales de las narices electrónicas utilizado tanto para una representación visual de lo sucedido en el algoritmo de búsqueda como para depuración del sistema ayudando a los procesos de implementación de nuevas estrategias mediante la observación de un patrones en las señales.

La ejecución de los scripts de representación ha sido vinculada tanto en los scripts de bash, observable en el protocolo de inicialización, como integrado en el código de localización de fuentes de odorante gracias a la librería subprocess nativa en el lenguaje de programación Python. Ejemplos de scripts para la representación de gráficas están situados en el anexo C.3.1, llamadas al sistema para ejecutar código externo de representación en la linea 520 del codigo Python del anexo C.7.2.

7

Integración, pruebas y resultados.

7.1. Introducción

En anteriores capítulos hemos detallado el estudio de los parámetros de control modificables en nuestro sistema, el diseño de los componentes que lo conforman y la integración de todas las partes desarrolladas. Capítulos 4, 5 y 6 respectivamente. En este capítulo se explicará la evolución de los algoritmos realizados para la consecución del objetivo de la localización de fuentes de odorante.
Como dijimos con anterioridad, el desarrollo del objetivo final ha tenido un carácter evolutivo y experimental, se han extraído una serie de hipótesis que caracterizan la dispersión en entornos controlados de las que se extraen las ideas que conforman los algoritmos iniciales. El algoritmo final ha sido desarrollado y aplicado en el sistema Olus para posteriormente introducirlo en el sistema integrado para el cuál alteramos el protocolo de toma de decisión.

La ordenación de este capítulo es por tanto temporal, comenzaremos analizando los algoritmos iniciales desarrollados, proseguiremos con la explicación del algoritmo final de localización tras el cuál, detallaremos el protocolo de experimentación y los experimentos realizados con ambas plataformas así como los resultados de los mismos.

7.2. Evolución del algoritmo

7.2.1. Introducción

En esta sección se presentaran las estrategias iniciales analizadas en el diseño final del algoritmo. Nos hemos basado en un proceso de análisis experimental aplicando las técnicas modulatorias de la sección 4.3.

En primer lugar se desarrollaron diferentes aproximaciones de algoritmos que modificaban la sensibilidad de los sensores a través de la modificación de la temperatura en ciclo cerrado. El bloque de modificación de temperatura media en el calentador, como se explicará más adelante, intenta disminuir los artefactos característicos que definen la sensibilidad de cada sensor a la vez que previene al sensor de entrar en un estado de reposo.

En el sistema se controlarán dos sensores situados de forma enfrentada como se observa en la figura 7.12.

En los algoritmos desarrollados se modificará la temperatura media del calentador de ambos sensores que componen un eje con el objetivo de modificar la sensibilidad de los mismos.
En primera instancia se hará una primera selección de entre distintas estrategias pensadas para la obtención de señales diferenciales entre ambos sensores que marcarán la dirección de la fuente de odorante. Posteriormente se modificaran los algoritmos seleccionados para realizar una comparativa más exhaustiva añadiendo una mejora al sistema.

7.2.2. Algoritmos de exploración

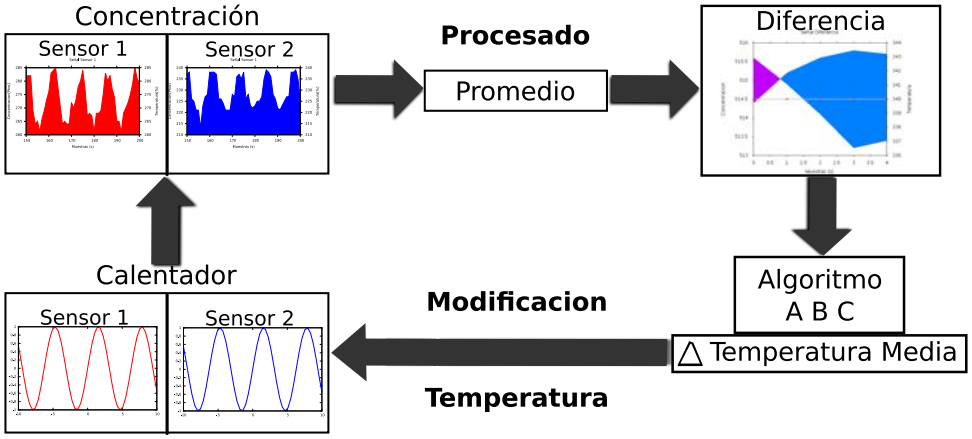


Figura 7.1: Esquema abstracción de la modificación de la temperatura de los sensores sobre el que efectuará control los diferentes algoritmos iniciales. Modificamos el sistema aplicando un incremento en la temperatura media del calentador, este valor es función de las señales diferenciales del sistema.

La experimentación se llevo a cabo en entornos cuya exposición a odorante se realizaba de forma controlada. Inicialmente y para todos los casos, se realiza un calentamiento inicial que representa en los sensores una inicialización común para eliminar el efecto de anteriores exposiciones, este proceso es análogo al introducido en el protocolo de inicio de la subsección 6.4.1.
El objetivo es la obtención de la estabilidad de las señales diferencia entre los sensores, este comportamiento nos facilitará la toma de decisión del algoritmo. Para el cumplimiento del objetivo utilizaremos estrategias de modificación de la temperatura media de los sensores. La premisas de las que se parten dado el entorno son las siguientes:

1. La fuente dispersa el odorante por comportamiento difusivo, por lo que el sensor posicionado más cerca en dirección a la fuente obtendrá, en circunstancias ideales, mayores concentraciones.

2. Los incrementos y decrementos de la temperatura de calefacción afectan a los sensores incrementando o decrementando la sensibilidad de los mismos alternativamente. Aunque esta última hipótesis únicamente se cumple en diferenciales relativamente pequeños de la temperatura en comparación con todo el rango de calentamiento.

3. La dispersión de los odorantes en entornos controlados es susceptible de verse modificada por perturbaciones en el sistema, por lo que el enunciado número 1 no se confirma en todo momento [23].

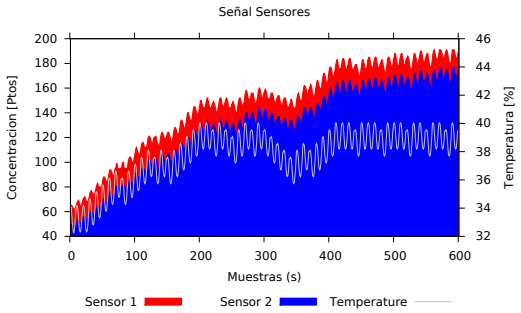
Las señales implicadas son tomadas de las concentraciones medias como hemos explicado con anterioridad en la figura 6.7.
La modificación de temperatura se realizará hacia el signo que incremente la diferencia entre sensores tomando uno de ellos como el sensor referencia del sistema. La figura 7.1 muestra una abstracción del proceso. El proceso final quedará detallado en la sección 7.3.3.

La modificación de temperatura se realizará en este primer caso en pasos fijos, posteriormente se implementaran mejoras proporcionales en el control del sistema.

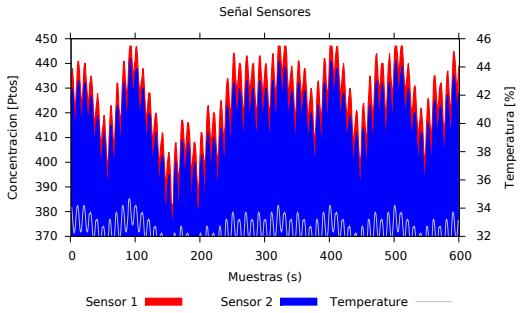
La representación de las gráficas están codificadas de la siguiente manera:

- Cada color representa el signo del valor diferencia. $Sensor_1 - Sensor_2$.
  El color que representa el sentido positivo + del diferencial y por tanto señal promedio mayor en el $Sensor_1$ será el definido por el color rosado. Figura 7.2c representativa de esta situación.
  El color que representa el sentido negativo - del diferencial y por tanto señal promedio mayor en el $Sensor_2$ será el definido por el color azulado. Figura 7.4d compuesta por este sentido del diferencial.
  La temperatura en todos los casos, tanto de la señal de los sensores como la promedio será representada por el color gris.

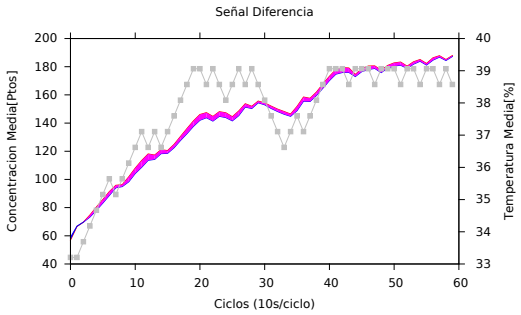
- El término que denominamos dirección es el posicionamiento de la fuente de odorante, la fuente siempre será posicionada en la dirección de la linea que marcan los sensores, por lo tanto existirán dos posiciones permitidas: la posición positiva +, caracterizada por encontrarse la fuente más cerca del $Sensor_1$ y la posición negativa -, caracterizada por posicionar la fuente más cerca del $Sensor_2$.



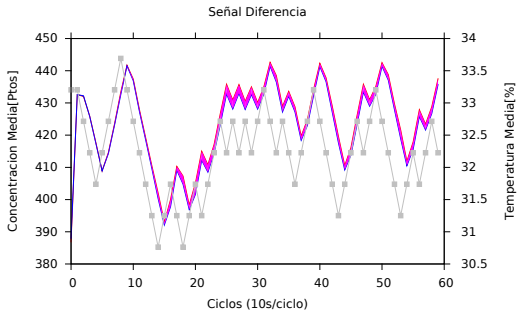
(a) Señales sin offset. Posición (+)



(b) Señales sin offset. Posición (-)

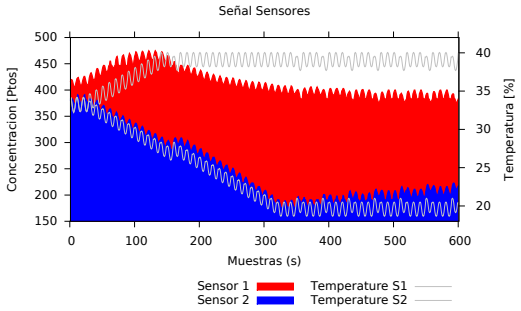


(c) Señales promediadas. Posición (+)

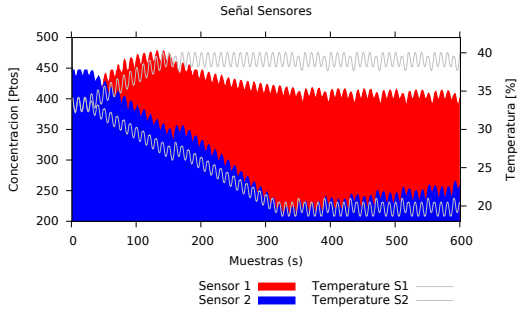


(d) Señales promediadas. Posición (-)

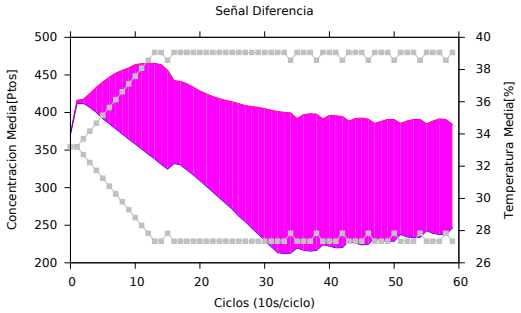
Figura 7.2: Algoritmo de prueba A. Señales del sistema, fuente a distancia de 100cm. Experimentos diferenciados por el posicionamiento de la fuente. En la columna de la figura a la fuente está posicionada aumentando la señal positiva + que representa el sensor 1, en la columna de la figura b la fuente está posicionada aumentando la señal negativa - que representa el sensor 2. El algoritmo modifica la temperatura en el sentido que incrementa la diferencia $Sensor_1$-$Sensor_2$.



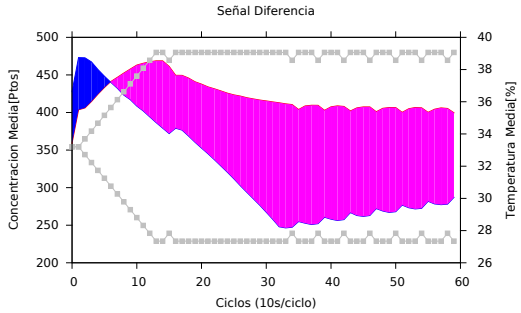
(a) Señales sin offset. Posición (+)



(b) Señales sin offset. Posición (-)



(c) Señales promediadas. Posición (+)



(d) Señales promediadas. Posición (-)

Figura 7.3: Algoritmo de prueba B. Señales del sistema, fuente a distancia de 100cm. Experimentos diferenciados por el posicionamiento de la fuente. En la columna de la figura a la fuente está posicionada aumentando la señal positiva + que representa el sensor 1, en la columna de la figura b la fuente está posicionada aumentando la señal negativa - que representa el sensor 2. El algoritmo modifica las temperaturas de ambos sensores para maximizar la diferencia de señal.

Algoritmo A: maximización de único sentido

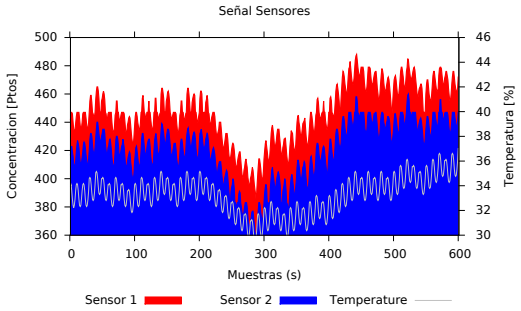
Modificación de temperatura de calentamiento efectuada para ambos sensores simultáneamente. En particular el algoritmo A únicamente maximizara un único sentido del diferencial entre sensores, el marcado por el signo positivo +. La figura 7.2 representa las señales provenientes de este algoritmo.

Algoritmo B: modificación de temperatura inversa

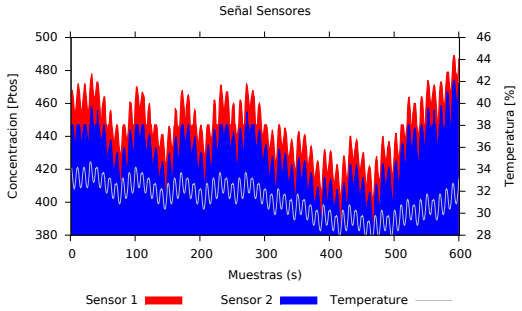
Modificación de las temperaturas de calefacción para ambos sensores de forma independiente. Los sensores modificarán la temperatura con signo el aumento de la señal $Sensor_1 - Sensor_2$ pero será de forma que uno lo haga en el sentido contrario que el otro. Podemos observar como este comportamiento condiciona de forma absoluta el comportamiento de los sensores en las gráficas de la figura 7.3.

Algoritmo C: maximización dinámica de sensores

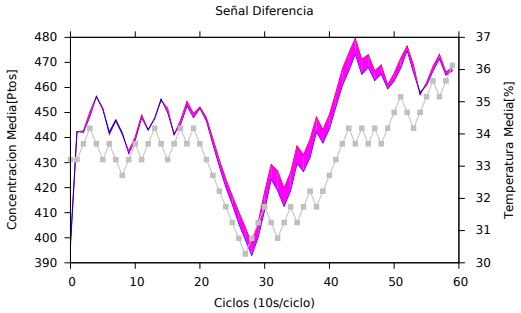
Modificación de las temperaturas de calefacción para ambos sensores simultáneamente. El signo de la modificación de temperatura se calcula en función de uno de los sensores que será seleccionado dinámicamente en la ejecución. En contraposición al algoritmo A, que únicamente modifica la temperatura con relación al $Sensor_1$.
La selección del sensor que maximiza la diferencia viene determinado por la señal diferencial. Si se obtienen diferencias en un sentido + o - de forma consecutiva el algoritmo cambia la referencia a el $Sensor_1$ o $Sensor_2$ respectivamente. La figura 7.4 muestra que la modificación de temperatura es adaptativa a las señales diferencia y por lo tanto a la posición de la fuente.



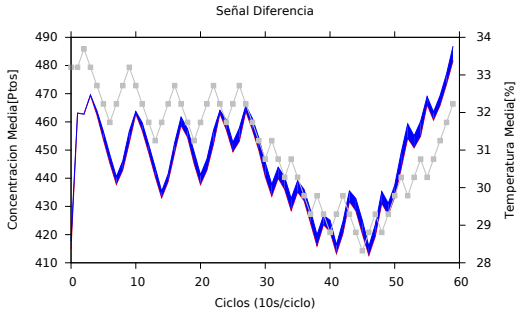
(a) Señales sin offset. Posición (+)



(b) Señales sin offset. Posición (-)



(c) Señales promediadas. Posición (+)



(d) Señales promediadas. Posición (-)

Figura 7.4: Algoritmo de prueba C. Señales del sistema, fuente a distancia de 100cm. Experimentos diferenciados por el posicionamiento de la fuente. En la columna de la figura a la fuente está posicionada aumentando la señal positiva + que representa el sensor 1, en la columna de la figura b la fuente está posicionada aumentando la señal negativa - que representa el sensor 2. El algoritmo modifica la temperatura para maximizar la diferencia guiada por el incremento de señal en un sensor. Las muestras consecutivas de la señal promedio en un sentido cambian el sensor al que beneficia el algoritmo.

Comparativa

Las gráficas obtenidas de los algoritmos explorados representan señales diferenciales entre dos sensores del sistema que serán posteriormente procesadas para formar las decisiones del algoritmo desarrollado. La experimentación se ha realizado en circunstancias ideales y equivalentes entre sí sin la utilización de la plataforma robótica. El movimiento era efectuado manualmente.
Por lo tanto, el objetivo es explorar diferentes conductas del sistema vinculadas con las estrategias de calefacción y realizar un análisis de cual es la estrategia que facilite la toma de decisión para su posterior desarrollo.

Experimentación y comparativa de señales:

- Hemos realizado entorno a 6 experimentos de cada estrategia. En cada uno de los experimentos las narices tomaban muestras en cada una de las distancias y posicionamientos de la fuente alternos (+ y -).
- La comparativa se ha realizado por medio de porcentaje de señales acertadas, se ha tomado como muestra acertada aquella cuya diferencia entre sensores es del mismo signo que la determinada por el posicionamiento de la fuente definido en la introducción. Veremos posteriormente en el algoritmo que cada toma muestral completa de una figura representa al menos una toma de decisión respecto el movimiento.

- La tabla adjunta 7.1 muestra los promedios del número de señales diferencia con signo igual al del posicionamiento de la fuente para cada uno de los experimentos y posiciones. A modo ilustrativo, la señal de la figura 7.3c presentaría un valor de 100 % ya que las muestras diferencia tienen el signo identificativo con la posición de la fuente mientras que en la figura 7.3d se caracterizaría por un valor de porcentaje entre 5 y 10 % de muestras diferencia del mismo signo que la posición de la fuente .

A partir de las gráficas obtenidas podemos descartar el algoritmo B como posible candidato ya que modifica el comportamiento de los sensores condicionándolos en todo momento. La información que obtenemos es inexistente porque no observar variar las señales con dependencia del posicionamiento de la fuente.

Podemos observar como el algoritmo denominado como C obtiene los mejores resultados generales en la tabla adjunto 7.1. A continuación modificaremos el valor de la modificación en temperatura en los algoritmos A y C para observar si el porcentaje del algoritmo C sigue siendo superior.

Comparativa de algoritmos			
Distancia (cm)	A (%)	B (%)	C (%)
100	70	52	76
75	62	52	97
50	42	52	99
25	69	52	97
Total	61	52	92

Cuadro 7.1: Comparativa de algoritmos realizada mediante el número de muestras coincidentes con la posición de la fuente respecto del total. Porcentajes promedios de muestras diferenciales acertadas obtenidas de 6 experimentos. Cada uno de los cuales tomando muestras a cada una de las distancias y posiciones de la fuente de manera independiente.

Modificación dinámica de la magnitud

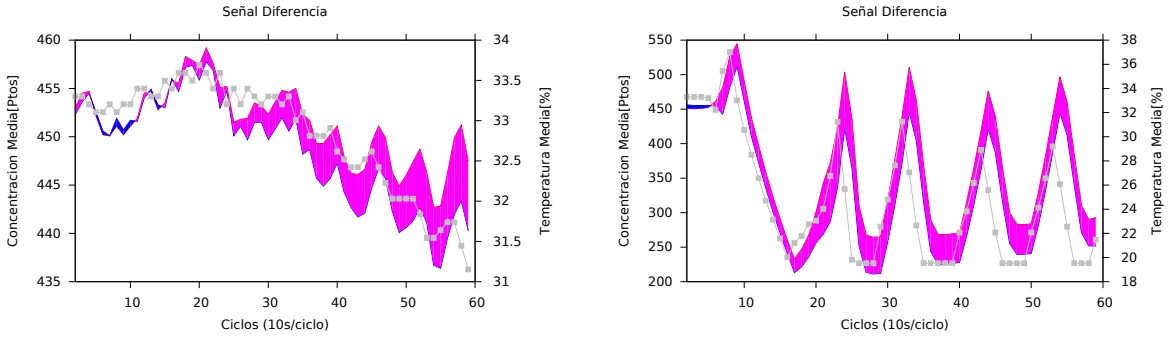
A continuación realizamos una mejora a los algoritmos modificando la magnitud del incremento de la calefacción en función de parámetros medidos en el sistema. La distancia a la fuente caracteriza el entorno y por tanto los probabilidad de los mayores incrementos en la señal de los sensores como se observa en la sección 2.2.2. Esta mejora proporcional, por tanto, nos permitirá una mejor adaptación al sistema adecuando la sensibilidad de los sensores al entorno en función de la distancia a la fuente. El resultado otorga amortiguación ante la aparición de incrementos esporádicos de la pluma a distancias medias y disminuye la sobreexposición de los sensores a distancias cortas.

El control de la magnitud del incremento se realiza mediante un controlador PID(PI) ajustado por metodología Ziegler-Nichols y explicado con anterioridad en la sección 2.4.1. El controlador PID es un conocido método aplicado en el control automático de parámetros con objetivo de minimizar una variable definida como error del sistema (e). Nuestro parámetro de error está definido como la diferencia entre el diferencial inmediato de los sensores que componen cada eje, o lo que es lo mismo, la diferencia de los incrementos en los sensores. Este valor queda representado en la ecuación 7.1 de la sección donde se explica el algoritmo final del sistema.

Como hemos comentado con anterioridad, esta adaptabilidad de la magnitud del incremento repercutirá modificando la sensibilidad de los sensores ante la llegada de una pluma en regiones muy cercanas. Por otro lado los cambios pequeños de la sensibilidad del sistema condición los sensores hacia un estado de estabilidad sin perder la sensibilidad, este comportamiento lo observamos en la figura comparativa de ambos algoritmos proporcionales 7.5.

Ambos sistemas representados en la figura 7.5 han sido ajustados con los mismos parámetros en el controlador PID. Se han seleccionado dos figuras que permiten representar los dos comportamientos que definen el ajuste en la magnitud de cambio del sistema.
En la ventana a se observa como la modificación de concentración en pequeños pasos modifica la sensibilidad igualmente y controla el nivel de la concentración medida; en la figura b, sucede el caso contrario, existe una sobreexposición de odorante, el calentador actúa acorde al diferencial de los sensores que ahora recibirán incrementos mucho mayores, y actúa aumentando los valores de modificación de la temperatura lo que observamos en el conjunto muestreal otorga de estabilidad al sistema.

Además, creemos que existen ciertos patrones en el perfil de modificación de la temperatura que podrían otorgar información adicional respecto la distancia a la fuente o comportamientos poco eficientes de los sensores. Estos datos se reservaran para trabajo futuro.



(a) Algoritmo A (b) Algoritmo C

Figura 7.5: Algoritmos con el mismo valor de los parámetros PID que representan la mejora proporcional del sistema. Muestras promedio obtenidas a 25cm de la fuente con posicionamiento de fuente +. Caso a, modificación de la sensibilidad con pasos pequeños de temperatura señales con poca variabilidad, salto máximo en temperatura de 0.2 puntos porcentuales y variabilidad total de la señal total de 20 puntos. Caso b, ejemplo de modificación de comportamiento por saltos grandes de temperatura pero otorgando estabilidad al sistema como observamos en la muestra periódica obtenida, saltos de temperatura de hasta 5 puntos porcentuales y variabilidad de la señal total de 300 puntos.

Comparativa algoritmos PI		
Distancia (cm)	A (%)	C (%)
100	96	83
75	22	85
50	100	93
25	52	97
Total	68	90

Cuadro 7.2: Tabla comparativa algoritmos aplicando salto por control PID respecto el diferencial del incremento entre sensores. La comparativa se ha realizado con datos obtenidos de 10 experimentos respectivamente. Cada experimento se caracteriza por muestrear en todas las distancias y posiciones posibles. En este caso dejamos fuera de comparativa al algoritmo B por ser el que peor resultados proporciona.

La tabla 7.2 nos ofrece la información, como hemos representado anteriormente, del promedio del número de señales diferencia con el mismo signo que el posicionamiento de la fuente con respecto el total. El número está promediado con 10 experimentos de los algoritmos en los cuales se han tomado muestras en todos los valores de distancia y posicionamiento de la fuente.

El algoritmo A aumenta su porcentaje respecto al paso fijo de la sección anterior, el algoritmo C lo disminuye en pocos puntos porcentuales pero ésto es debido a una iteración que obtuvo resultados particularmente malos.
El algoritmo C sigue destacando por obtener mejores resultados respecto el algoritmo A, la mejora proporcional del sistema obtiene resultados mejores de forma general debido a la aparición de los comportamientos descritos en la figura 7.5 que le otorga adaptabilidad al medio y flexibilidad en la explotación del algoritmo.

7.2.3. Algoritmo C práctico

Hemos realizado el estudio previo de los algoritmos previos y seleccionado el algoritmo C como la estrategia de modificación de temperatura que mejores resultados a obtenido.
A continuación vamos a ajustar lo que antes era un estudio previo a las condiciones reales sobre las que se ejecutara el algoritmo de localización utilizando la plataforma robótica y mejorando la eficiencia.

En las anteriores pruebas, la toma muestreal se realizaba en circunstancias ideales:

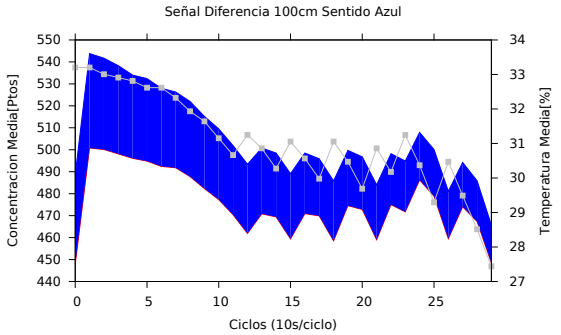
- Las muestras entre distancias eran aisladas por muestras de control.
- El número de muestras por distancia era elevado para identificar posibles errores aleatorios.
- El movimiento era guiado de forma manual las distancias y dinámicas de movimiento eran iguales para realizar la comparativa.

Tras la selección del algoritmo vamos a modificar el protocolo experimental para adecuarlo a la situación de experimentación vinculada con las estrategias de localización, aplicaremos los siguientes cambios:

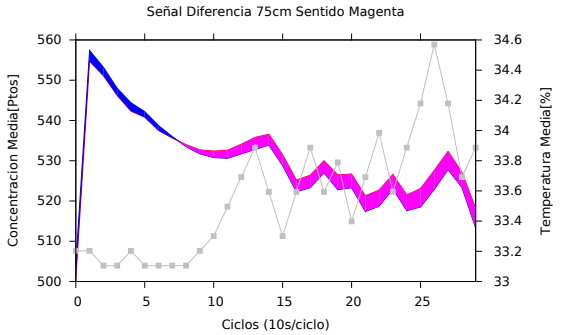
- Las muestras se tomaran de forma continua por lo que tomas muestrales anteriores influenciarán el comportamiento de las siguientes.
- El número de muestras se hará menor para aumentar la eficiencia total del sistema.
- El movimiento será realizado por la plataforma robótica con los errores del mismo y las perturbaciones creadas por el movimiento.

La figura 7.6 ejemplifica las señales obtenidas con esta metodología de experimentación. En ella se observan valores muestrales obtenidos de forma continua con un valor bajo de 300 muestras, recordemos que el procesado inicial se realizaba por ventana excluyente de la sección 6.7. La plataforma es rotada tras cada movimiento para observar la sensibilidad del sistema a las dinámicas del mismo.

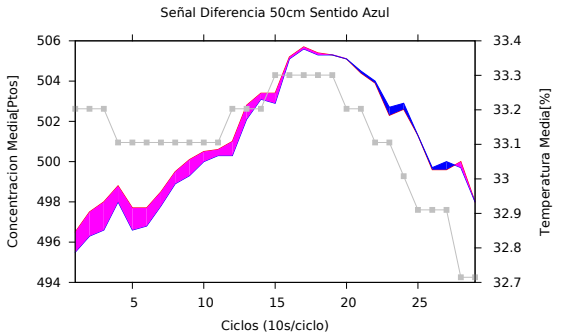
El análisis de la utilización del algoritmo C bajo estas circunstancias ha concluido satisfactoriamente. El número de muestras que reflejan el posicionamiento de la fuente es mayor del 50 % por cada ventana muestral. Por lo tanto, obtenemos suficiente información de este algoritmo para tomar decisiones automatizadas por medio del promedio de estas diferencias que conformarán cada eje del sistema.



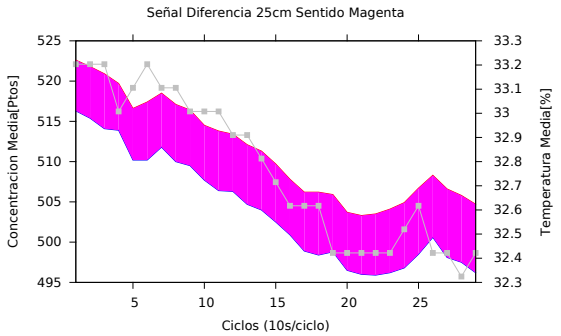
(a) 100cm. Posición (-)



(b) 75cm. Posición (+)



(c) 50cm. Posición (-)



(d) 25cm. Posición (+)

Figura 7.6: Tomas muestrales acortadas, tomadas de forma continua e incluyendo una rotación en el sistema entre movimientos. 100cm, 75cm, 50cm y 25cm son las distancias que representan las figuras a, b c y d respectivamente. Observamos como la sensibilidad del sistema es capaz de discriminar los cambios en la dirección de la fuente de odorante.

7.3. Algoritmo de localización

7.3.1. Introducción

Tras la fase de exploración realizada hemos obtenido una estrategia para maximizar la información sensorial, a continuación se diseñara el algoritmo de localización teniendo en cuenta los componentes que conforman el sistema para dar paso a una explicación más detallada del núcleo del algoritmo.
El algoritmo se ha desarrollado desde la perspectiva del sistema Olus para posteriormente diseñarse en el sistema integrado, ambos diseños compartirán las lógicas de ejecución.

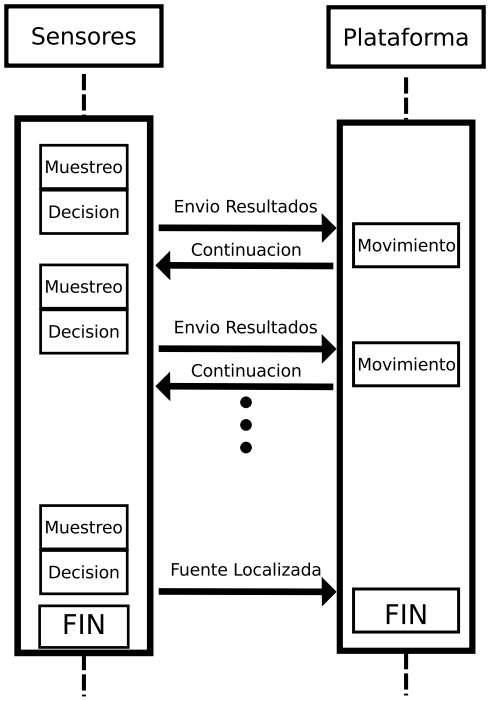


Figura 7.7: Diagrama secuencial del algoritmo desarrollado para el control del sistema de búsqueda de odorante. Para el sistema Olus este esquema representa el script del anexo C.4.1, mientras que en el sistema integrado se encuentra en el código del anexo C.7.2. La arquitectura desarrollada está compuesta por bloques del sistema que se corresponden con hilos compartiendo información y sincronizándose mutuamente por cada toma muestreal de sensores que corresponde a una decisión del sistema. El bucle de muestreo finaliza con el envío de la señal de fuente localizada al bloque movimiento que a su vez enviara la señal finalización a todos los hilos del sistema.

7.3.2. Bloques del algoritmo

Entendemos como bloque, cada uno del conjunto de funcionalidades independientes que conforman el algoritmo. Representan el procesamiento y control de las narices electrónicas, la plataforma robótica y los sensores externos.
La figura 7.7 muestra un diagrama del comportamiento global del sistema, cada bloque encapsulará funciones propias en relación a los sistemas desarrollados. En todos los casos, la comunicación entre bloques se ejecuta secuencialmente. El bloque de sensores muestrea la señal de las narices electrónicas y enviará las decisiones tomadas a la plataforma robótica. Ésta a su vez ejecutará el movimiento y devolverá una señal de llegada a la nueva localización para sincronizar el inicio del nuevo periodo de muestreo.

El sistema Olus ejecutará esta rutina de comunicación a través de los scripts de bash que conforman el sistema en la unidad central. Código del script en el anexo C.4.1.
El sistema integrado ejecutará un único codigo cuya diferenciación de bloques está representada por funciones ejecutadas por hilos del sistema. El código de las mismas, bloque sensores y movimiento se encuentra en las lineas 126 y 450 respectivamente del anexo C.7.2.

Bloque Sensores

Está compuesto por el conjunto de funciones que ejercen control y obtienen datos de las narices electrónicas Olus de la sección 6.2.1 para el caso del sistema Olus o narices electrónicas v2 de la sección 5.3 para el sistema integrado. El código del algoritmo en el sistema Olus se corresponde con el codigo del anexo C.2.4 mientras que en el sistema integrado corresponde con la linea 126 del anexo C.7.2. El esquema de la figura 7.8 muestra el diagrama de flujo de las dinámicas internas del bloque.

- **Muestreo.** Captura de datos procedentes de las narices electrónicas. Los parámetros de frecuencia de muestreo y tipo de señal serán utilizados en el calentador de los dos sensores que conforman cada uno de los ejes. Tras el tiempo de espera, establecido en función de la frecuencia, obtenemos los datos de concentración procedentes de los pines analógicos, figura de conexiones 3.12, o del paquete del PIC como muestra la figura 6.4.

- **Procesado.** Modulo de procesado de las señales de concentración procedentes de las narices electrónicas. El tipo de ventana de procesado del promedio será el parámetro de este bloque, por lo general se ha evolucionado a la utilización de la ventana solapante de la figura 6.7 ya que nos otorga mayor información. A su vez se conforma la señal diferencia entre las señales promedio de cada uno de los sensores que conforman un eje. Estos datos son extraídos a ficheros para su posterior representación en gráficas.

- **Modificación de temperatura.** El conjunto de funciones que conformarán la modificación final en la temperatura media de la función de calefacción en los sensores. La modificación de temperatura tiene como objetivo el aumentar o disminuir la sensibilidad del eje con el fin de aumentar la diferencia entre los sensores que conforman dicho eje. La salida del proceso será la magnitud y signo del incremento en temperatura que modificaran los parámetros del sistema. La función de modificación será detallada en la sección 7.3.3.

- **Toma de decisión.** Tras la finalización del procesado y modificación de la señal de calentamiento una vez por cada ciclo de temperatura, se realiza una toma de decisión.
  La toma de decisión dirigirá el posicionamiento futuro de la plataforma robótica con el objetivo de la localización de fuente. La información procedente de las señales diferencia de las tomas muestrales serán los datos que alimentarán este bloque. Los detalles de esta función se explicarán más adelante en la sección 7.3.3.

- **Sistema bidimensional.** Cada eje ejecutara la parte del algoritmo desde el muestreo hasta la toma de decisión de forma independientes, la interrelación entre ellos y el sistema vendrá dado por cada uno de los desarrollos. En Olus, será el script externo el encargado de tomar las decisiones y ejecutar la nueva iteración de muestreo, en el caso del sistema integrado, se realizará mediante la misma señal de sincronización del bloque plataforma.

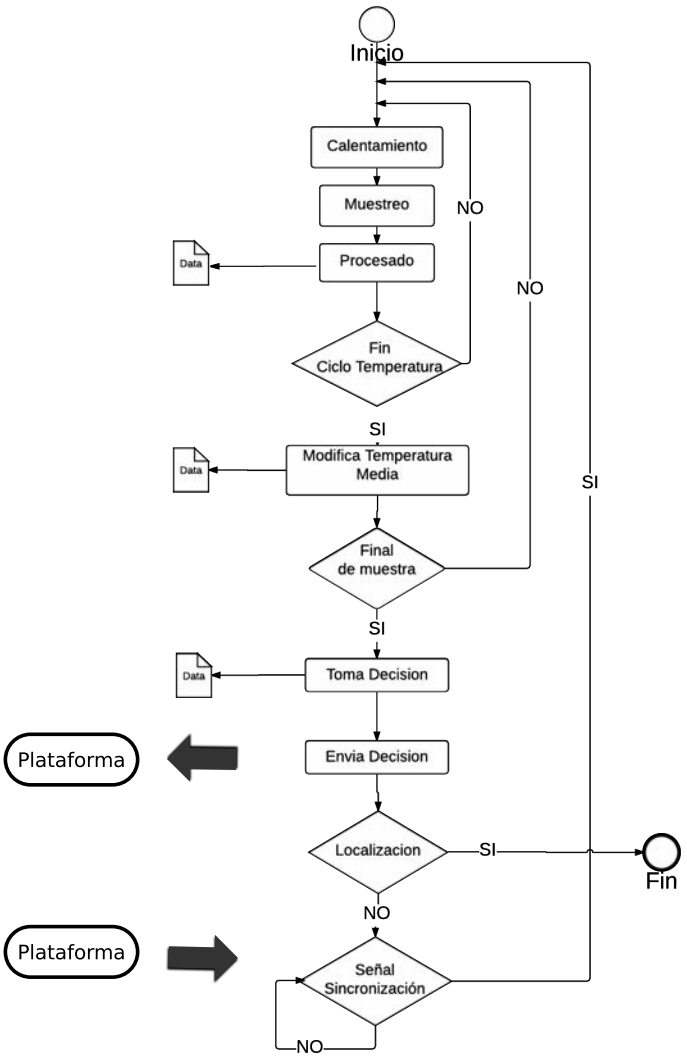


Figura 7.8: Diagrama de actividad del bloque sensor del algoritmo de localización para el control del sistema de búsqueda de odorante. La ejecución de este bloque se realiza simultáneamente para ambos ejes en el caso de la búsqueda bidimensional. De dentro hacia fuera observamos el bucle interno que corresponde al ciclo de temperatura y su posterior modificación, el bucle de toma de decisión marcado por el número de muestras y el bucle de decisiones finalizado por la señal de localización. Observamos los bloques de conexión con la plataforma dependiente de la condición de localización y enviada desde el bloque movimiento.

Bloque Plataforma

El siguiente bloque cuyo diagrama de flujo observamos en la figura 7.9 es el bloque de movimiento de la plataforma robótica. Este bloque tiene la función de procesar las decisiones de los sensores y codificarlas en el control del movimiento. Para el sistema Olus este bloque corresponde en la toma de decisión con el script del anexoC.4.1 y en la conexion remota con la plataforma en el script C.4.3, en el sistema integrado este bloque comienza en la linea 450 del código de localización del anexo C.7.2.

- **Captura y procesado de datos.** conjunto de funciones que extraen las decisiones del bloque sensores para procesarlas en las direcciones finales que serán entrada de la siguiente etapa. Las decisiones individuales de los ejes se normalizan en este bloque para obtener el porcentaje de giro que debemos de aplicar en la plataforma robótica para el caso bidimensional.
- **Tipo de movimiento.** este bloque representa los protocolos de movimiento implementados que están vinculados cada uno con una toma de decisión del bloque de sensores en la sección 7.3.3.
  - **Protocolo Promedio.** El tipo de decisión vinculado con este tipo de movimiento es la decisión por promedio utilizada en el sistema Olus en los experimentos de localización. En este protocolo tras el muestreo y decisión de los sensores el bloque plataforma simplemente moverá hacia la dirección que le marquen los datos enviados y procesados de las decisiones tomadas.
  - **Protocolo Rotado.** Protocolo unido al tipo de decisión rotada. La decisión necesita de dos ciclos de muestreo, es por esto que las decisiones enviadas son tenidas en cuenta como en el anterior protocolo cada dos ciclos de ejecución de este bloque. En el primer ciclo el sistema únicamente rotará la plataforma 180 grados para continuar con otro ciclo de muestreo.
- **Señal de continue muestreo.** Enviará la señal de sincronización en el sistema integrado, o acabará el subscript de movimiento de la plataforma en el caso de Olus, para dar paso al bloque de sensores.

En el sistema integrado, y debido a la división en hilos de los bloques que componen el algoritmo, se genera una espera dado el tiempo de muestreo del bloque sensores. Este tiempo de latencia entre bloques podría ser utilizado para implementar una interfaz de comunicación con el usuario de forma que a través de instrucciones predeterminadas podamos modificar los parámetros del algoritmo o el posicionamiento del robot. Esta posible mejora queda reseñada en trabajo futuro, sección 8.2.

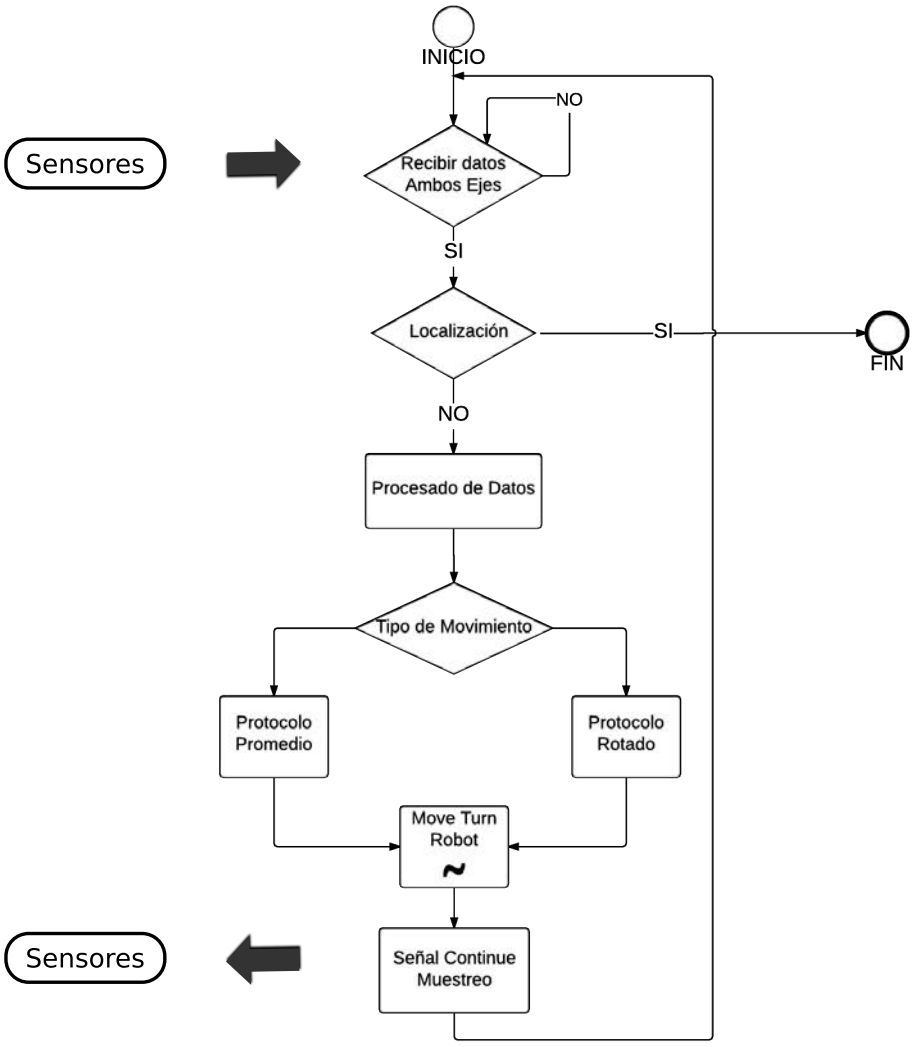


Figura 7.9: Diagrama de actividad del bloque movimiento del algoritmo de localización. Inicio en espera de los datos de muestreo de las narices electrónicas procedentes del bloque sensores. Procesado de decisiones y tipo de movimiento según el protocolo utilizado. Posterior conexión externa con los bloques de código *move* y *turn*. Condición de finalización del sistema enviada por la conexión con el bloque sensores confirmando localización de fuente.

7.3.3. Núcleo del bloque de sensores

En esta sección explicaremos en detalle algunas de las funciones clave para comprender la ejecución de las mismas. Conforman los puntos clave del algoritmo del sistema.

Modificación de Temperatura

El conjunto de funciones que modifican la temperatura media representan la función de adaptación del sistema al entorno, a través del control de temperatura podemos ejercer control sobre la sensibilidad del sistema. Absorber las perturbaciones del sistema y proveer de protección contra sobreexposiciones son sus dos tareas principales. Estas funciones y la evolución de la estrategia queda reflejado en la sección 7.2.
La sensibilidad, como se ha demostrado en la sección 4.3.1, no es a gran escala una función lineal con la temperatura. Es por esto que utilizamos un controlador PID(PI) que proporciona un control dinámico dado un objetivo concreto: maximizar la señal diferencia de los sensores.

La figura 7.10 representa los subprocesos por los cuales se modifica la temperatura del sistema:

- **Procesado.** Bloque de entrada, representa los parámetros necesarios para conformar los valores del proceso.
- **Controlador PID.** A través del ajuste de un controlador PID(PI) ejercemos control sobre la magnitud del sistema. El valor de error e_p se define como la diferencia entre incrementos inmediatos entre ambos sensores que conforman un eje 7.1. El valor de n es la ultima muestra diferencial tomada ya que el ajuste se realiza en tiempo de ejecución. Este factor aumentara su valor con relación a las perturbaciones que sucedan en cada sensor independientemente. El valor e_i es un valor atenuador, se opone a la modificación de temperatura con el valor acumulativo de e_p, esto provoca que ante inestabilidades del sistema consecutivas el valor de e_i disminuirá el cambio para no provocar más inestabilidad debida al cambio brusco de temperatura.

$$e_p = (\overline{Sensor1[n]} - \overline{Sensor1[n-1]}) - (\overline{Sensor2[n]} - \overline{Sensor2[n-1]}) \tag{7.1}$$

$$e_i = \Sigma e_p[n] \tag{7.2}$$

- **Parámetros de signo.** Los parámetros del signo, como su propio nombre indica conformarán el signo del incremento de temperatura. El objetivo del conjunto es el de maximizar el signo que mayor cambio a provocado en la dirección tentativa del eje.
  - Signo sensor referencia. Es el signo del diferencial de las señales promedio, este signo se obtiene del inicio del algoritmo y puede ser cambiado por el encuentro de signos consecutivos parametrizado en el algoritmo. Es decir, al inicio de la iteración se fija el signo según diferencia del promedio, ese será el signo del sensor que se motivará hasta que no se encuentre consecutivamente un número de veces parametrizado el signo contrario.
  - Signo diferencial. Es el signo de la diferencia de incrementos. Análogo al valor e_p de la ecuación 7.1 se obtiene el signo del incremental. Si en el ultimo ciclo ha incrementado el sensor marcado como líder se continuara realizando la misma modificación, si ha disminuido su diferencia, se cambiara el signo del incremento de temperatura.
  - Signo actual. Memoria del sistema, signo anterior de la modificación de temperatura. En función de los otros parámetros este signo marcará si se debe de continuar incrementando o disminuyendo el valor de la temperatura.

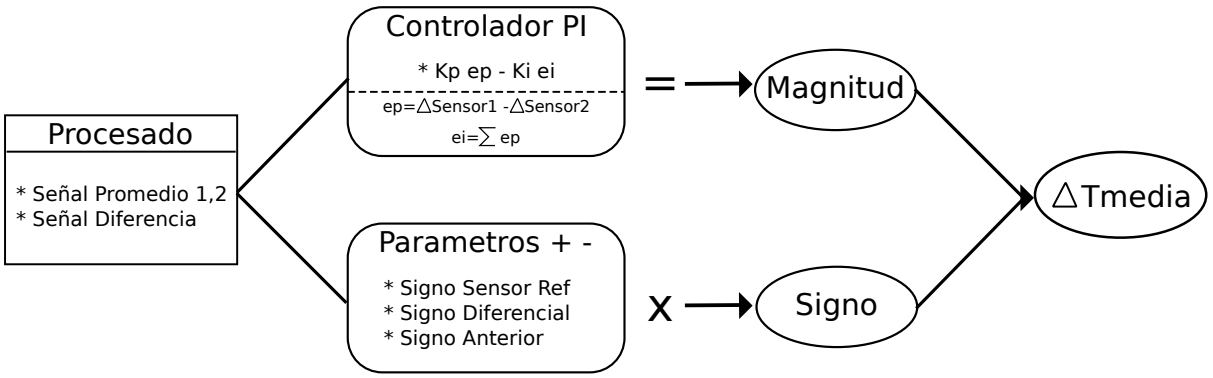


Figura 7.10: Esquema de la función que modifica la temperatura promedio de la señal del calentador. El incremento en temperatura media esta compuesto de dos factores: magnitud y signo. La magnitud se calcula mediante un controlador PID(PI) con respecto los valores de la señal diferencia 7.1. El cálculo del sentido esta compuesto por tres signos del sistema: signo sensor referencia, cada sensor tiene asignado un signo identificado según el cálculo del diferencial($Sensor1 - Sensor2$); signo diferencial, es el signo identificativo de la influencia de la anterior modificación; signo anterior, es el valor almacenado del signo con el que se incremento el pasado ciclo la temperatura promedio.

El objetivo de la función es maximizar la diferencia entre ambos ejes. Es intuitivo pensar que ante sobreexposiciones la modificación de temperatura ayude a diferenciar estas señales, sin embargo el cambio brusco en la temperatura provocado por la diferencia de incrementos ha resultado ser beneficiosa disminuyendo la sensibilidad independientemente del signo de la temperatura. El cambio directo en la sensibilidad del sistema únicamente sucede a incrementos pequeños.

Toma de decisión

La toma de decisión de la dirección de movimiento ha evolucionado a lo largo de los experimentos. La toma de decisión de nuestro sistema estará vinculada con el tipo de movimiento del bloque plataforma de la sección 7.3.2 que detallaremos en el protocolo de experimentación de la sección 7.4.2.

- **Decisión Promedio.** Política de decisión utilizada en el sistema Olus, se calcula el promedio de las diferencias medidas en cada eje que conforman dos sensores eliminando las primeras que introducían ruido producido por el estado estacionario anterior al muestreo. El promedio obtenido dirigirá la plataforma hacia el sentido del sensor con mayor señal. En este tipo de decisión el ajuste inicial del parámetro *offset* entre las señales de los sensores a temperatura ambiente es una parte frágil, un mal ajuste de offset puede beneficiar un sentido con respecto el otro. Este comportamiento queda resaltado en la sección 7.4.3.

- **Decisión Rotada.** Política de decisión según el diferencial de las muestras rotadas. El protocolo del valor de decisión es el siguiente:

  1. Muestreo y reinicio de variables. Se reiniciará la variable que controla el sensor referencia tomado en la modificación de la temperatura del eje. Tras la obtención de este nuevo sensor según las primeras muestras obtenidas, se ajustará la señal de *offset* entre las señales de concentración de los sensores.
     Esta señal de *offset* era la variable que permanecía inmutable durante la ejecución del algoritmo con el otro tipo de decisión.
  2. Cálculo diferencial. Toma muestral y cálculo de diferencia de promedios equivalente a la decisión de tipo promedio.

3. Rotación de ejes. En el bloque movimiento se identificará este como el ciclo 1 del algoritmo y rotará la plataforma 180 grados dejando los sensores en sentido inverso al inicio.
4. Muestreo. Ejecución de otro ciclo de muestreo sin modificar las variables de control del sistema anteriormente citadas en el paso 1.
5. Cálculo de rotación. Ahora se tomará la decisión en función de la señal promedio actual y la señal promedio anterior.

El movimiento se dirigirá hacia el sentido que marque el diferencial entre el promedio de la primera toma muestral y su correspondiente toma rotada. Es decir, hacia el sentido en el que se ha incrementado la diferencia entre sensores tras la rotación de la plataforma.
Expondremos un ejemplo práctico para clarificar este comportamiento:

- Reseteamos las señales de offset y de sensor referencia. Tomamos las señales diferencia de la toma muestral y realizamos el promedio como en el caso de la decisión promedio. En este caso practico obtendremos un valor promedio de -3 puntos de concentración diferenciales.
- Rotamos la plataforma y sin modificar las variables de control volvemos a realizar el muestreo de las narices electrónicas. En este caso, en el promedio de los diferenciales hemos obtenido un valor de -1 punto de concentración diferencial.
- Comparativamente entre las tomas de decisión expuestas, los valores negativos de los promedios decidirían dos veces de forma consecutiva hacia la dirección del $Sensor2$ ($Sensor1 - Sensor2$), decisión en (-3) y en (-1). Sin embargo, el cálculo de la decisión rotada obtendría un valor de $+2$ $((-3)-(-1))$ y por lo tanto decidiría el movimiento hacia el $Sensor1$.
  Esta comparativa no es real ya que el protocolo promedio no esta vinculado con el reseteo inicial de los parámetros y por tanto no obtendría los mismos valores.

Como comentamos anteriormente si la muestra rotada se encuentra dentro de un rango de incertidumbre tomaremos la decisión basándonos en el segundo criterio. Para ello, realizaremos el mismo cálculo rotado que hemos explicado antes, pero en este caso se tendrá en cuenta el parámetro de la tendencia y no del promedio. El parámetro de la tendencia se calcula con el valor diferencia entre el promedio final y el promedio total de la señal muestral.

Si estas dos decisiones se encuentran dentro del rango de incertidumbre se tomara la decisión de no movimiento a la espera de otra iteración del mismo procedimiento.
Una muestra visual de como afecta la toma de decisión en el algoritmo puede observarse en la figura 7.11 que define el protocolo de experimentación con los sistemas.

7.4. Experimentos del sistema

7.4.1. Introducción

Los experimentos de esta sección conforman la introducción de los algoritmos desarrollados en el entorno real de localización de odorantes con la integración total de los componentes de los sistemas artificiales olfativos desarrollados.

Inicialmente se reserva una sección para explicar con detalle los protocolos que conforman la experimentación con nuestros sistemas, posteriormente se explicarán los experimentos realizados con ambos sistemas y los resultados obtenidos en la tarea de localización.

7.4.2. Protocolo de experimentación

Debido a la dimensionalidad de los factores externos al sistema que pueden modificar los resultados, enfatizaremos el protocolo de experimentación seguido con objetivo de mantener los resultados en el mayor grado de reproducibilidad posible. El protocolo de experimentación queda expuesto gráficamente en la figura 7.11.

- Todos los experimentos que expondremos están realizados en el mismo lugar de experimentación, el descrito en la sección 4.4.2.
- El protocolo de inicio descrito en la sección 6.4.1 se ejecutará a modo de calentamiento de los sensores en el lugar de explotación del algoritmo sin la presencia de odorante. Este protocolo mantendrá las narices electrónicas en una situación estable antes de la exposición del odorante.
- La distancia a la que se sitúa la plataforma robótica inicialmente se encuentra en el rango de los 100cm y los 150cm de la fuente de odorante.
- La representación de la fuente de odorante puede verse modificada debido a los distintos experimentos realizados según la guía de reactivos y concentraciones de la sección 4.4.3.
- La distancia que se recorre tras la toma de decisión ha sido modificada en los experimentos unidimensionales realizados con Olus se ha buscado la estabilidad de las señales y aumentar el número de decisiones por lo que se realizaron pasos de 15cm. En los experimentos bidimensionales del sistema integrado se ha pretendido realizar una búsqueda real con el menor tiempo posible en la localización con lo que se aumento el paso a 20cm.
- En la figura 7.11 observamos el protocolo de experimentación en su totalidad, tras el inicio de los sensores con el protocolo de inicio se explotará el algoritmo desarrollado de la sección 7.3, en los experimentos desarrollados hemos utilizado el protocolo promedio para el sistema Olus y el protocolo rotado para el sistema integrado excepto experimentos concretos como el de la sección 7.4.4. Podemos observar gráficamente que la toma de decisión en el protocolo rotado ocupa dos tomas muestrales de las narices electrónicas.

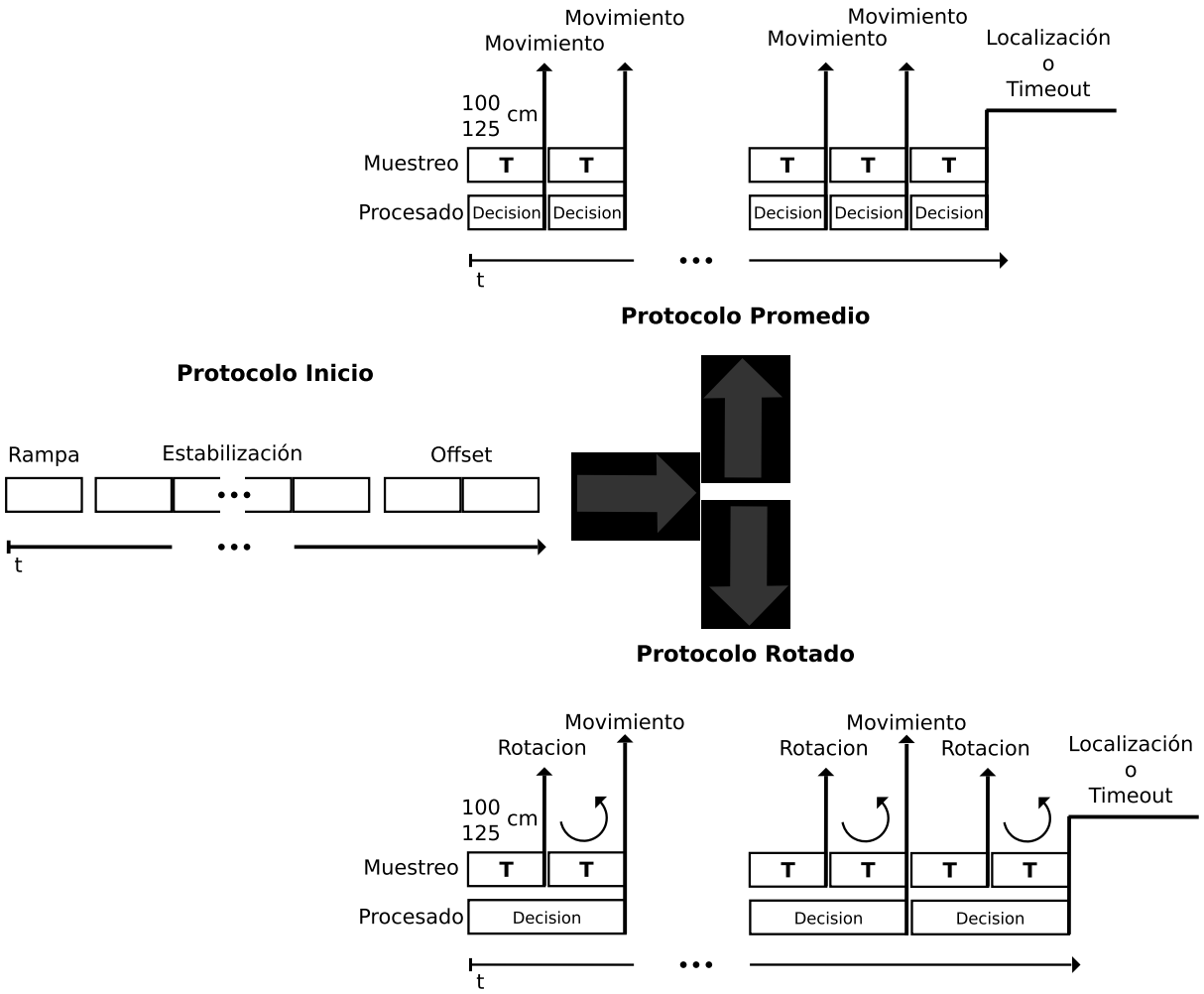


Figura 7.11: Esquema del protocolo de experimentación utilizado para la explotación del algoritmo desarrollado. Se ha utilizado el protocolo promedio para el caso de Olus y el protocolo rotado en el sistema integrado.

7.4.3. Sistema Olus

Los experimentos del sistema Olus únicamente han sido realizados en una dimensión debido a los inconvenientes del sistema respecto la movilidad del mismo. Para la experimentación Olus y debido al estado inicial del movimiento en la plataforma robótica, se decidió colocar una cuerda a modo de guía para asegurar el movimiento unidimensional de la plataforma. En este punto de la experimentación se realizaron experimentos que validaran el algoritmo mediante la simulación de diferentes situaciones reales de un sistema completo de localización.

La medida comparativa entre experimentos será el porcentaje de decisiones acertadas tomadas por el algoritmo. Aunque cada comportamiento representado componga un reto diferente, se expondrán las variantes de experimentación antes de exponer los resultados obtenidos.

Las narices electrónicas que tendremos en cuenta para la decisión del movimiento se situarán por tanto en el eje longitudinal y paralelo a la dirección del movimiento de la plataforma por mayor facilidad en la ejecución del movimiento, queda representado este eje en la figura 7.12.

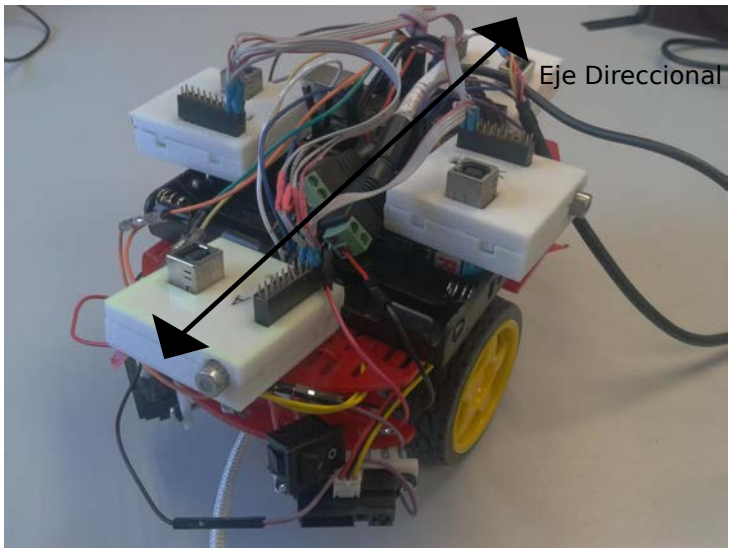
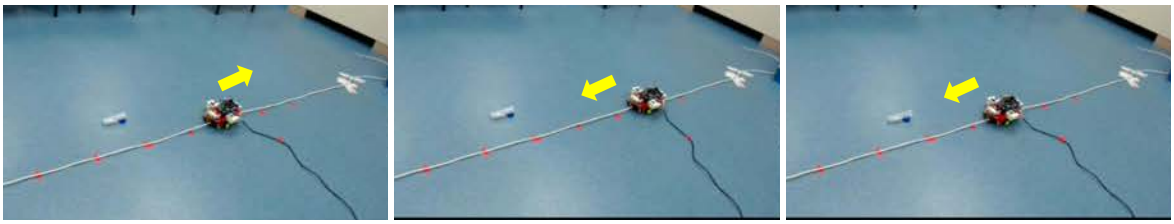


Figura 7.12: Imagen del sistema Olus determinando el eje direccional sobre el que se ejecuta el movimiento que utilizamos en los experimentos de una dimensión.

Búsqueda errónea y ajuste de parámetros

Queremos resaltar con esta sección la importancia del protocolo inicial de calentamiento explicado en la sección 7.4.2. En él ademas de minimizar los efectos de histéresis observados en los sensores realizamos una serie de muestreos con modificación de temperatura cuyo fin es la de obtener las temperaturas de inicio a las cuales el offset entre señales es constante.
Este offset referido a la medición diferencial de la concentración con los sensores situados en el mismo estado, será un parámetro fundamental en el algoritmo para la toma de decisión según decisión promedio a la que nos referimos en la sección 7.3.3.

En la realización de los distintos experimentos se ha observado que un ajuste erróneo del *offset* puede ocasionar un mal comportamiento de la toma de decisión puesto que se conservará este ajuste para el resto de la localización, como es característico del protocolo promedio 7.4.2. En la figura 7.13 observamos como aunque la plataforma ha alcanzado distancia cercanas a la fuente las decisiones son tomadas de forma alternativa moviendo de forma oscilante la plataforma.

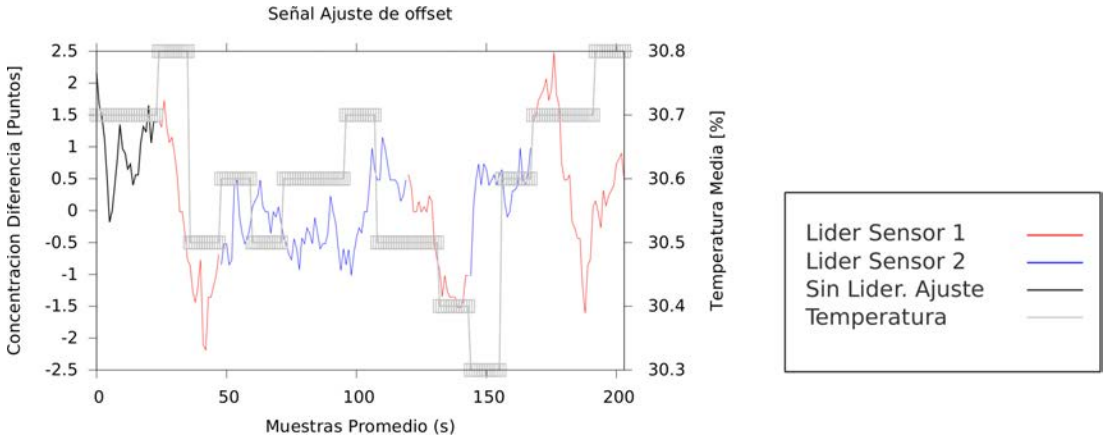


(a) (b) (c)

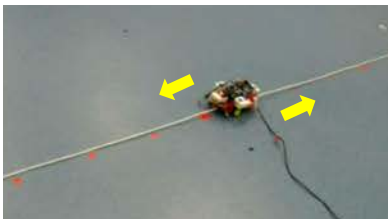
Figura 7.13: Experimento en una dimensión perteneciente al sistema Olus. La plataforma alcanza distancias cercanas a la fuente sin embargo se mantiene oscilante tomando decisiones erróneas en un alto porcentaje. La hipótesis de este comportamiento anómalo es que el ajuste de offset inicial de la plataforma debía de ser erróneo.

El ajuste de este parámetro es por tanto una parte fundamental del algoritmo por toma de decisión promedio. Para saber con certeza que el parámetro está bien ajustado, nos fijamos en que la señal diferencia obtenida tiene un caracter oscilante entre los valores positivos y negativos que caracterizan la decisión en el eje. La representación de la figura 7.14a muestra un ejemplo de la señal en este estado.

En el estudio de este parámetro realizamos un experimento en el que no exponemos al sistema ninguna fuente de odorante, este experimento es observado en la imagen de la figura 7.14b. Las decisiones son consecutivamente en un sentido y en el otro permaneciendo la plataforma en el lugar inicial como es característico de un buen ajuste de offset.



(a)



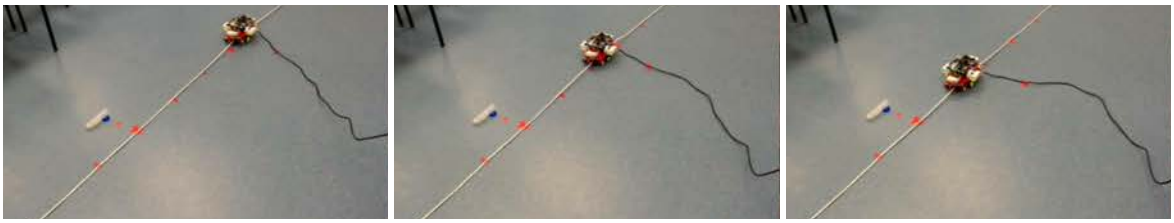
(b)

Figura 7.14: Experimento en una dimensión. En la figura a mostramos un ejemplo de la señal diferencia cuyo ajuste de offset muestra señales entre sensores que oscilan entre la dirección de un sensor y el otro. En la figura b mostramos el experimento del sistema en el cual eliminamos el estímulo de la fuente, la plataforma se mantiene oscilante en sus decisiones.

Búsqueda de odorante

Experimentación realizada sobre un único eje de la plataforma dirigiendo la fuente en linea directa con la dispersión del odorante. Experimento básico realizado para validar la correcta obtención de los vectores de movimiento obtenidos.

En el caso del sistema Olus, se realizo con la fuente de odorante tipo tubo de la sección 4.4.3, que dispersa el odorante de forma más lineal, concentración de 10 % y volumen de 10 centilitros de una disolución de etanol en agua. La figura 7.15 contiene las imágenes de este tipo de experimento.

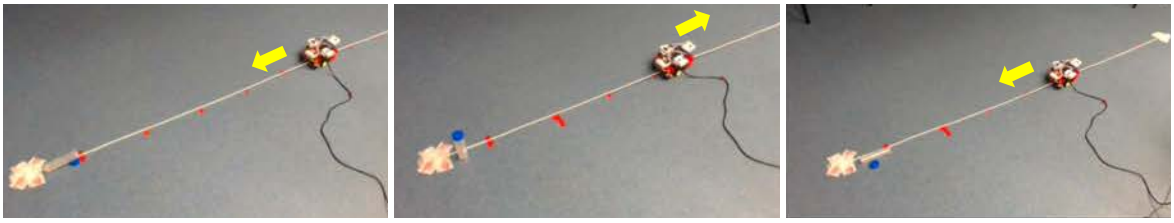


(a) (b) (c)

Figura 7.15: Experimento en una dimensión perteneciente al sistema Olus. La fuente, cuya concentración de etanol es de 10 %, permanece abierta en todo el experimento. La plataforma robótica se mueve hacia la fuente de odorante.

Búsqueda discontinua

En este tipo de experimento se elimina el estímulo de odorante para observar el comportamiento del algoritmo.
La figura 7.16 muestra el comportamiento de la plataforma robótica con respecto la apertura y cierre de la fuente de odorante. En el primer caso las narices muestras una dirección de movimiento acorde a la fuente de odorante, al cerrarse la fuente en la figura 7.16b la señal de las narices electrónicas toma una decisión opuesta al movimiento, esto es producido a un posible efecto rebote de la señal.



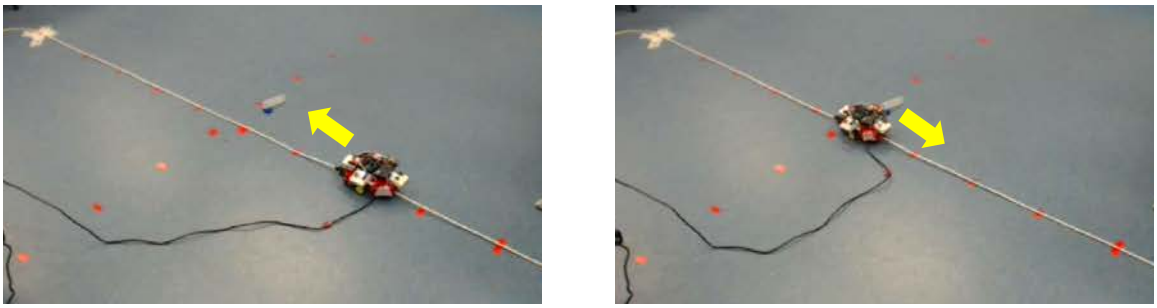
(a) Fuente abierta (b) Fuente cerrada (c) Fuente abierta

Figura 7.16: Experimento en una dimensión perteneciente al sistema Olus. La fuente se abre y se cierra provocando un movimiento de avance y retroceso de la plataforma robótica. La flecha amarilla determinada el sentido del próximo movimiento realizado.

Este tipo de búsqueda representa la falta de información sensorial representativa de entornos donde la perturbación domina el sistema. El efecto de cambio de dirección podría ser debido a la deriva ocasionada por el efecto de histeresis de los sensores. Cabe destacar que el algoritmo no está preparado para distinguir entre estados con estímulo y sin estímulo olfativo como representa el comportamiento propio de las búsquedas bioinspiradas de la sección 2.3.

Búsqueda perpendicular

Tipo de búsqueda realizada con variación de la posición de la dispersión de la fuente de odorante en la linea perpendicular respecto la linea que representa el eje direccional del sistema. Este tipo de búsqueda nos ofrece información futura para la implementación de la búsqueda en dos dimensiones.
La figura 7.17 muestra como la señal del sistema Olus dirige el sentido correctamente aún cuando la dispersión de la fuente se dirige en perpendicular con el mismo. Al llegar a la fuente se queda oscilando.



(a) (b)

Figura 7.17: Experimento en una dimensión perteneciente al sistema Olus. La dispersión de la fuente se sitúa en sentido perpendicular respecto la dirección del eje del movimiento. La flecha amarilla determinada el sentido del próximo movimiento realizado. El experimento concluye determinando la posibilidad de incluir una segunda dimension en este sistema.

A la vista de este experimento podemos concluir que el movimiento bidimensional podría ser posible en este sistema.

Resultados

La tabla 7.3 muestra los resultados en el conjunto de los experimentos realizados en el sistema Olus.
Los campos de los que se compone la tabla son los siguientes:

- La fuente de odorante ha sido en todos los casos la fuente lineal debido al caracter unidimensional de los experimentos.
- Las modalidades de experimentación se han dividido en dos agrupando los experimentos por sus resultados, en búsqueda en linea se tienen en cuenta los experimentos de búsqueda de odorante y búsqueda discontinua, búsqueda perpendicular estará compuesta de los experimentos realizados con la fuente en esta determinada posición.
- El campo de localización es referido al porcentaje del total que la plataforma robótica alcanza satisfactoriamente la fuente de odorante.
- El campo decisiones se refiere al porcentaje de decisiones acertadas respecto el total.
- El campo totales define el numero de decisiones tomadas en todos los experimentos, se seleccionó este parámetro debido a la variabilidad entre número de decisiones de las que se componen los experimentos.

Dimensión	**Fuente**	**Tipo Experimento**	**Localización**	**Decisiones**	**Totales**
1D	Lineal	Búsqueda en Linea	100 %	75 %	128
1D	Lineal	Búsqueda Perpendicular	100 %	72 %	35

Cuadro 7.3: Tabla de resultados de experimentación en Olus. La búsqueda en linea conglomera los experimentos de búsqueda de odorante y búsqueda discontinua. Decisiones acertadas se caracterizan por dirigir el movimiento del sistema a la fuente. La localización es el porcentajes del total de experimentos que alcanza finalmente la fuente. Este tipo de experimentación es muy sensible al ajuste inicial con lo que el protocolo inicial debe de cumplirse exhaustivamente.

El porcentaje obtenido entre la búsqueda con la fuente en posición perpendicular y en la linea paralela es muy similar si bien el número de experimentos llevado a cabo en el caso de la búsqueda perpendicular fue muy inferior a la búsqueda lineal como se infiere del número de decisiones totales.

Como habíamos adelantado con anterioridad la toma de decisión por promedio llevada a cabo por el sistema Olus es muy sensible a errores de ajuste de offset con lo que debíamos de ser muy exhaustivos en el proceso de inicio, el porcentaje de acierto cae drásticamente cuando realizamos un mal ajuste de offset.

Cabe destacar que el protocolo de experimentación total para la realización de los experimentos es objeto de optimización debido a que el tiempo total para la localización de la fuente es aún muy alto.

El sistema Olus ha obtenido resultados muy buenos en la consecución del objetivo inicial planteado de búsqueda de odorante. El porcentaje de acierto del protocolo experimental en este sistema sobrepasa las expectativas planteadas inicialmente.
La búsqueda bidimensional fue planteada como objeto de estudio del sistema integrado.

Se ha editado un vídeo con los experimentos demostrativos de la localización del sistema Olus en una dimension [60].

7.4.4. Transición entre sistemas

El sistema integrado fue diseñado con el objetivo de convertirse en el sistema de localización de fuentes de odorante caracterizado por una serie de ventajas con relación al sistema Olus: mayor movilidad de conexión a través del adaptador USB-wifi, flexibilidad de comunicación entre sus componentes no vinculada a protocolos y eficiencia de medios gracias a la integración del sistema en un único microprocesador y a un único código que dirige todo el sistema. La sección 6.3 refleja la motivación principal del sistema integrado respecto al sistema Olus.
Esta sección tendrá como objetivo principal comparar los dos sistemas desarrollados. Para ello se integrarán ambos sistemas y se realizarán mediciones simultaneas.

En este experimento se utilizara la toma de decisión de tipo promedio 7.3.3 y el tipo de movimiento vinculado con el mismo que definen el anterior sistema Olus y que han demostrado ser efectivos en el mismo.
Las pruebas realizadas con este tipo de protocolo en el sistema integrado configurado con los parámetros del sistema Olus han tenido resultados notablemente diferentes respecto el porcentaje de acierto. Este experimento nos ayudaran a explicar las diferencias entre sistemas que ocasionan un descenso de acierto con el protocolo desarrollado en el sistema Olus.

En el experimento expondremos la plataforma de la imagen 7.18 con ambos sistemas integrados a la fuente de odorante de tipo circular con 30cl de una disolución de etanol al 50 %. La toma de decisión se realizará por promedio de la señal diferencia como es característico del sistema Olus, el procesado de la señal promedio se realizará en ventana deslizante para tener mayor número de puntos. Se tomaran medidas empezando en la distancia 100cm de la fuente de odorante y la posición negativa del diferencial siguiendo la nomenclatura estipulada en la sección 7.2.2.

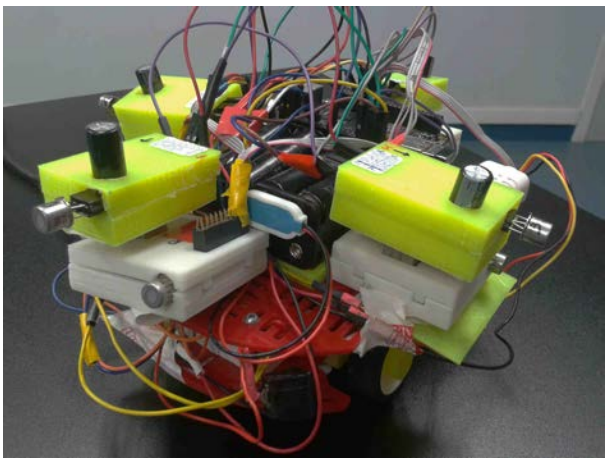


Figura 7.18: Imagen de la plataforma robótica en la cual se integran los dos sistemas desarrollados. Se tomaran muestras simultáneamente, se expondrá el sistema a la fuente de tipo circular con 30cl de una disolución de etanol al 50 %.

La representación de la figura 7.19 de las señales diferencia en ambos sistemas se ha realizado en la misma escala pudiendo realizarse una comparativa visual de lo sucedido.

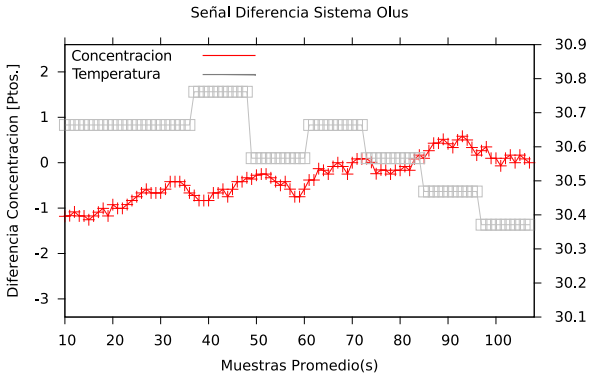
Podemos obtener las siguientes conclusiones de la comparativa de la figura:

- **Sensibilidad.** Podemos observar en todas las distancias de la figura 7.19 como las señales del sistema integrado son más sensibles que las señales diferencia obtenidas del sistema Olus. Mientras que las señales Olus de la figura 7.19c varían entorno a un punto diferencial, las análogas del sistema integrado de la figura 7.19d lo hacen en más de 5 puntos diferenciales. La curva a 25 centímetros de la figura 7.19f alcanza un pico de valor el doble que su señal análoga en el sistema Olus de la figura 7.19e.
- **Variabilidad.** Las curvas de las imágenes 7.19a y 7.19b tienen la misma tendencia hacia valores positivos, sin embargo, las obtenidas en el sistema integrado tienen una variabilidad mucho mayor, los ciclos de subida y bajada de la señal son mucho mayores y por lo tanto los cambios de temperatura también lo son.
- **Offset.** Al observar la evolución de las señales en las figuras 7.19b y 7.19d se puede apreciar como mantener una señal de *offset* constante es una estrategia que induce errores en el sistema. Partiendo de la base que la dinámica de temperatura tiene un comportamiento equivalente para ambas ventanas, la ventana d presenta una desviación de eje de un punto debido al protocolo por promedio que mantiene el *offset* inicial del sistema. Mientras que la toma de decisión de Olus permanecería en el rango de la señal negativa, la evolución de la señal equivalente del sistema integrado presentaría incertidumbre entre la toma de decisión hacia el sentido **-** de la figura 7.19b y la toma de decisión hacia el sentido + de la figura 7.19d.

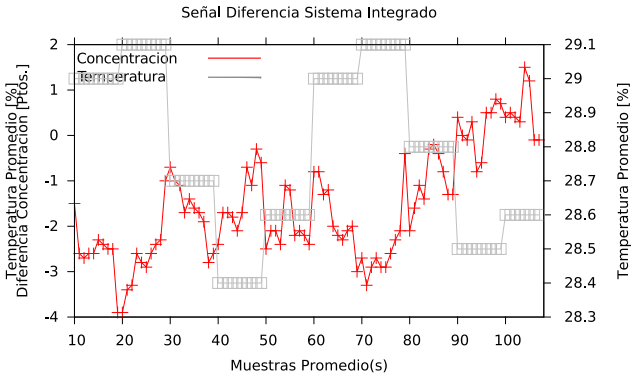
Ambos sistemas comparten el mismo modelo de sensor TGS2611 del fabricante Figaro[9]. Las narices electrónicas del sistema integrado están compuestas por sensores nuevos mientras que los sensores de los encapsulados del sistema Olus han estado expuestos a condiciones extremas durante periodos muy largos. En la sección 4.4.1 está el detalle de este encapsulado.

La exposición de estos resultados es coherente con la hipótesis del envejecimiento de los sensores. Los sensores envejecidos presentan una sensibilidad menor ante el estímulo olfativo y ante la modulación de temperatura. Ante estos resultados, se ha cambiado el protocolo de toma decisión a la toma de decisión rotada y al tipo de movimiento vinculado con el mismo.

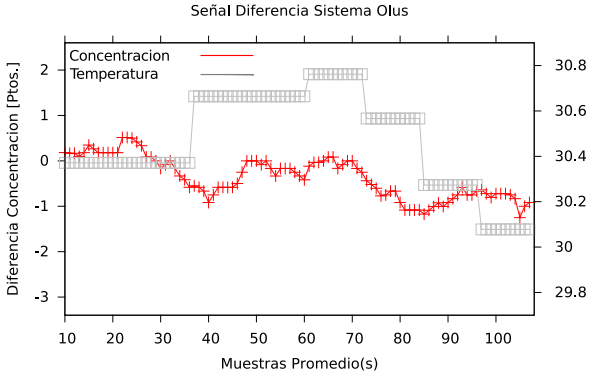
Por lo tanto, las decisiones del sistema integrado serán tomadas tras obtener dos tomas muestrales por cada una de las distancias a la fuente como describimos en la sección de protocolo de experimentación 7.4.2.



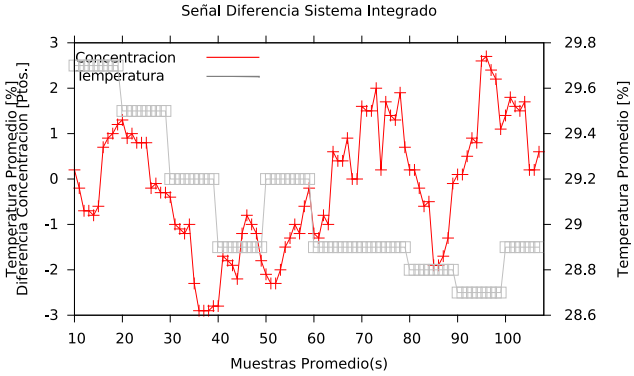
(a) Olus. 100cm posición de fuente -



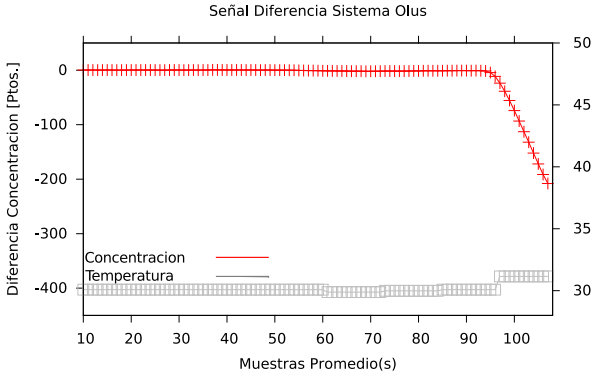
(b) Integrado. 100cm posición de fuente -



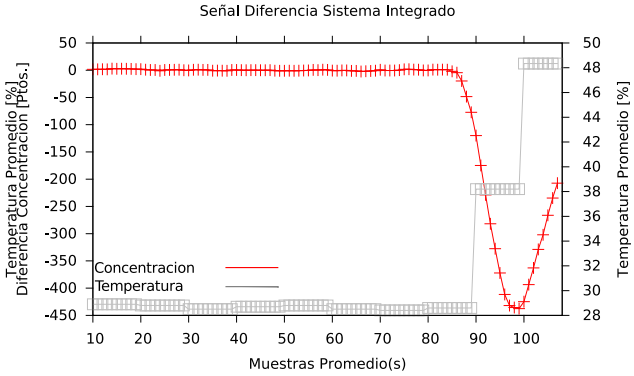
(c) Olus. 50cm posición de fuente -



(d) Integrado. 50cm posición de fuente -



(e) Olus. 25cm posición de fuente -



(f) Integrado. 25cm posición de fuente -

Figura 7.19: Comparativa entre sistemas muestreando simultáneamente espacial y temporalmente. Fuente situada en el sentido - de la diferencia $Sensor1 - Sensor2$ en todo el experimento. Experimento realizado 4 veces de manera independiente con resultados que favorecen la reproducibilidad. Las señales han sido representadas preservando la escala de forma que puedan ser fácilmente comparables. El sistema Olus representado en la columna de la izquierda presenta señales más suaves, con menor variabilidad, menor sensibilidad y señal diferencia más estable en comparación con las señales procedentes del sistema integrado. Este efecto podría ser ocasionado por el envejecimiento de los sensores del sistema Olus los cuales han estado expuestos a condiciones extremas.

7.4.5. Sistema Integrado

Los experimentos del sistema integrado están caracterizados por dos factores fundamentales que lo diferencian del sistema Olus anteriormente expuesto, la modificación en el protocolo de movimiento y decisión y la utilización principalmente de la fuente de dispersión circular.

Este sistema se encuentra en fase de desarrollo con respecto el ajuste de parámetros. El experimento mostrado en la transición entre sistemas de la sección 7.4.4 demuestra que los sensores de cada sistema tienen propiedades diferentes por lo que los parámetros que en Olus han obtenido resultados satisfactorios no serán apropiados para el sistema integrado.
El ajuste de parámetros es un procedimiento experimentalmente complejo debido a la complejidad misma de la dispersión de odorantes.

El nuevo protocolo denominado como rotado y observable en la sección 7.3.3 posee dos diferencias principales con el análogo del sistema Olus que crean algunas modificaciones generales:

- Realizamos una rotación por cada distancia en la que se sitúa la plataforma, esto crea que los sensores no tomen derivas debidas a la sobreexposición continuada de odorante a cortas distancias.
- Reseteamos las señales de offset y de la variable de control que mide el signo del sensor que beneficiamos en la modificación de temperatura. Esto nos permite aumentar el número de experimentos por cada protocolo inicial de calentamiento, en el sistema Olus la señal *offset* era un parámetro que se ajusta en cada inicio de algoritmo y por lo tanto por cada experimentos debíamos de realizar el protocolo previo de experimentación de la sección 7.4.2. Puesto que la señal de offset es variable y no guía la decisión final del movimiento podemos realizar varios experimentos en cascada sin que esto afecte el correcto funcionamiento del algoritmo.

La experimentación de este sistema se ha centrado en la búsqueda de la bidimensionalidad tomando la unidimensional como experimento inicial.
La linea de los sensores que forman el eje direccional es el marcado en la imagen de la figura 7.20

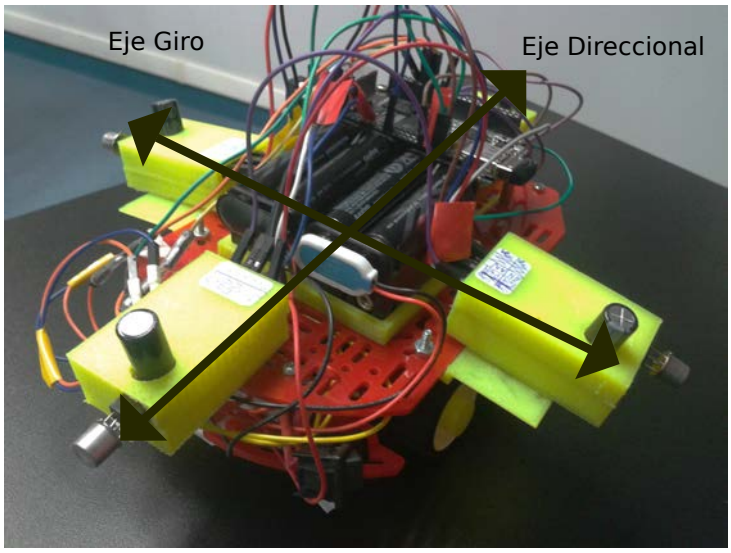


Figura 7.20: Imagen del sistema integrado determinando el eje direccional sobre el que se ejecuta el movimiento que utilizamos en los experimentos de una dimensión.

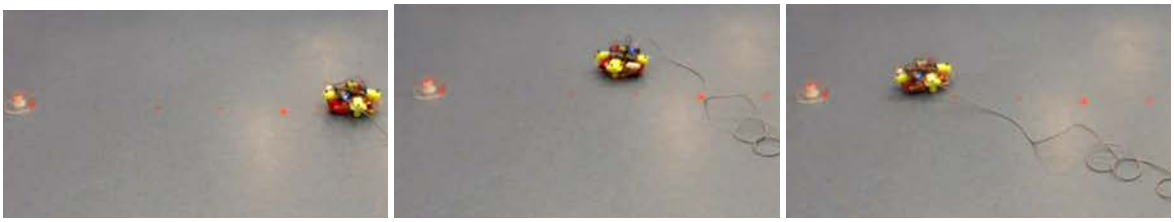
Búsqueda unidimensional

La experimentación llevada a cabo con el sistema integrado en una dimensión fue realizada con ambos tipos de fuente para tener un nexo de unión con los anteriores experimentos: la fuente de dispersión circular de la sección 4.4.3, con una concentración de 50 % de etanol en agua y volumen de 30 centilitros, y la utilizada en los experimento del sistema Olus y denominada como lineal con disolución de 10cl al 10 % de etanol en agua. El cambio de tipo de fuente se implemento para la posterior caracterización bidimensional del algoritmo.

La figura 7.21 muestra los experimentos realizados con la fuente lineal, la figura 7.22 hace referencia a los experimentos realizados con la fuente de dispersión circular.

(a) (b) (c)

Figura 7.21: Experimento en una dimensión perteneciente al sistema integrado. La fuente dispersa de forma lineal, concentración 10 % de disolución de etanol, permanece abierta en todo el experimento. La plataforma robótica se mueve hacia la fuente de odorante.



(a) (b) (c)

Figura 7.22: Experimento en una dimensión perteneciente al sistema integrado. La fuente dispersa de forma homogénea en todas direcciones, concentración 50 % de disolución de etanol, permanece abierta en todo el experimento. La plataforma robótica se mueve hacia la fuente de odorante.

El algoritmo toma decisiones hacia la fuente de odorante hasta la localización de la misma. El porcentaje de decisiones acertadas disminuye en el caso de la fuente de dispersión circular, si bien la diferencia es poco apreciable posteriormente en la sección 7.4.5 de resultados presentaremos los porcentajes.

Búsqueda bidimensional

Los experimentos realizados en dos dimensiones han sido únicamente llevados a cabo por el sistema integrado. Los experimentos previos en el sistema integrado bidimensional con la fuente lineal llevaron al cambio de la fuente de odorante por un tipo de fuente cuya dispersión es homogénea en los 360 grados. Tras la experimentación con diferentes técnicas y modificaciones de la dispersión se seleccionó la fuente circular descrita en la sección 4.4.3.

El setup es idéntico a los anteriores casos y se utilizará de la misma forma el protocolo rotado descrito en la sección 7.4.2. La decisión es tomada de forma independiente en ambos ejes de coordenadas y será en el bloque del algoritmo de la plataforma donde se realizará el procesado del giro.

El procesado del giro en función de las decisiones independientes es el siguiente:

1. Normalización de ejes. Las decisiones tomadas están cuantificadas por el incremento del promedio, parámetro característico de la toma de decisión rotada de la sección 7.3.3.
   A los valores de ambos ejes se les normalizará con el valor del eje direccional que guiará al sistema.

2. Obtención de porcentaje intereje. Con los valores normalizados se obtiene el porcentaje que representa la decisión en el eje de giro con respecto el eje direccional.

3. Cálculo de giro. El porcentaje de giro intereje es aplicado al giro máximo permitido por el sistema, al encontrarse el sistema en fase de pruebas hemos delimitado un giro máximo de 45 grados.

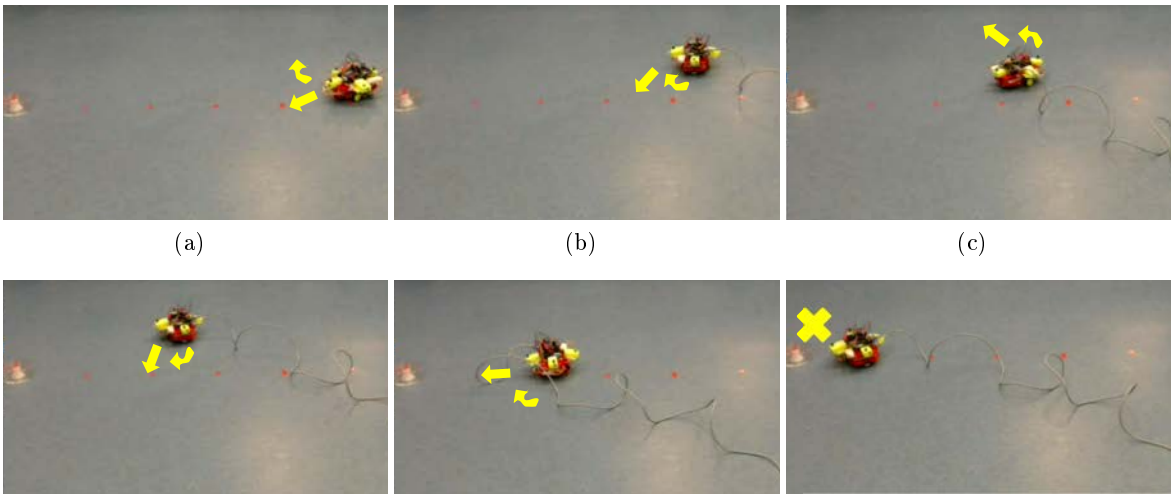


Figura 7.23: Experimento en dos dimensión realizado por el sistema integrado. La fuente dispersa de forma homogénea en todas direcciones, concentración 50 % de disolución de etanol, permanece abierta en todo el experimento. La plataforma robótica se mueve hacia la fuente de odorante en dos dimensiones. Las decisiones tomadas realizan una trayectoria satisfactoria en la localización pero poco eficiente.

La trayectoria típica es la realizada por la plataforma robótica en la figura 7.23 en la cual finalmente localiza la fuente de odorante. La trayectoria recorrida toma más pasos de los necesarios debida a la función que ecualización intereje.

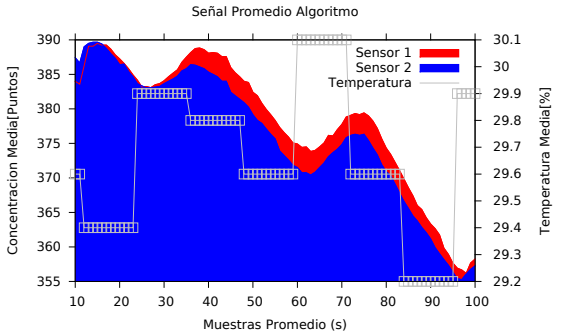
Este sistema se encuentra todavía en desarrollo por lo que es todavía altamente optimizable, es sabido que el valor de las decisiones de ambos ejes es función no lineal con la temperatura a la que situamos los mismos. Al ser estas temperaturas diferentes para cada eje necesitaríamos aplicar un valor de suavizado con respecto la temperatura para poder comparar ambos ejes entre si como lo hacemos en estos experimentos. En trabajo futuro queda esta optimización del sistema para mejorar los porcentajes de acierto, con particular detalle en el caso bidimensional.

Búsqueda sin control de temperatura

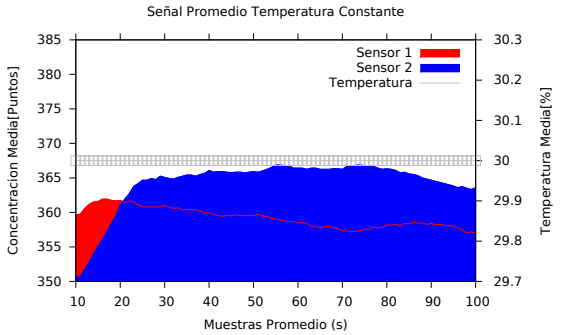
Finalmente se pretendió comparar la mejora obtenida con la modificación de temperatura del algoritmo de localización. Para ello, realizamos un conjunto de experimentos en las mismas condiciones previas y experimentales que los realizados en experimentación bidimensional del sistema integrado.

En la figura 7.24 se puede observar el resultado de la comparación entre las señales promedio y diferencia obtenidas para el algoritmo desarrollado y el equivalente mantenido a temperatura constante.
Las señales obtenidas muestran claramente como afecta el control de la temperatura en las señales que posteriormente formarán la decisión de movimiento. En las señales obtenidas se observa un control de las mismas a través de la modificación de la temperatura, esta modificación modula las señales otorgando estabilidad a la señal diferencia de la figura 7.24c respecto de su análoga a temperatura constante de la figura 7.24d.

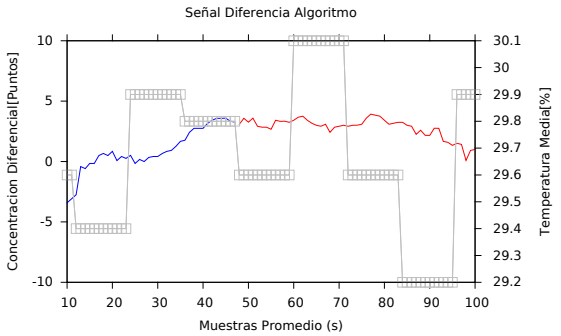
El resultado de la realización de estas pruebas está cuantificado en la tabla 7.4 con los resultados del sistema completo.



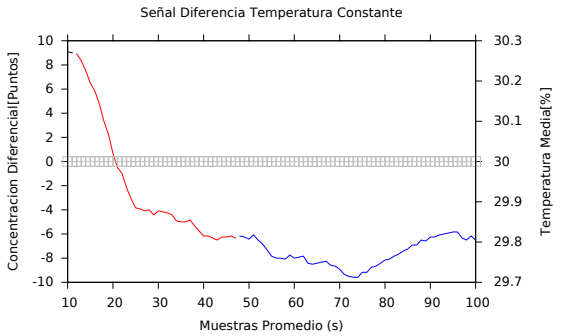
(a) Algoritmo 100cm. Promedios



(b) Temperatura constante 100cm. Promedios



(c) Algoritmo 100cm. Diferencial



(d) Temperatura constante 100cm. Diferencial

Figura 7.24: Comparativa del algoritmo de control de la temperatura respecto el mismo sistema con los sensores a temperatura constante y 100 cm de la fuente de odorante en las mismas circunstancias. Ventanas a y c correspondientes al sistema con modificación de la temperatura, ventanas b y d al sistema con temperatura constante. Los colores de las señales diferenciales representan el sensor que maximiza la modificación de temperatura, en este caso no aplicable en la ventana d. Las señales son altamente inestables como observamos en las ventanas diferenciales que mantenemos en la misma escala, esta inestabilidad provoca la oscilación y la toma de decisión con poca efectividad como se observa en la tabla de resultados 7.4.

Resultados

La tabla 7.4 muestra los resultados en el conjunto de los experimentos realizados en el sistema integrado.
Los campos de los que se compone la tabla son los siguientes:

- La fuente de odorante ha sido en un caso lineal en comparación de la fuente circular que caracteriza los experimentos bidimensionales.
- Las modalidades de experimentación son las denominadas: búsqueda en linea, equivalente a los experimentos unidimensionales del sistema Olus; búsqueda sin control, caracterizado por eliminar el control de temperatura en comparación con el algoritmo diseñado; y búsqueda de odorante, siendo este tipo de búsqueda el sistema completo bidimensional.
- El campo de localización es referido al porcentaje del total que la plataforma robótica alcanza satisfactoriamente la fuente de odorante.
- El campo decisiones se refiere al porcentaje de decisiones acertadas respecto el total.
- El campo totales define el numero de decisiones tomadas en todos los experimentos, se seleccionó este parámetro debido a la variabilidad entre número de decisiones de las que se componen los experimentos.

Dimensión	**Fuente**	**Tipo Experimento**	**Localización**	**Decisiones**	**Totales**
1D	Lineal	Búsqueda en Linea	73 %	69 %	95
1D	Circular	Búsqueda en Linea	67 %	76 %	47
2D	Circular	Búsqueda sin Control	0 %	46 %	21
2D	Circular	Búsqueda de Odorante	57 %	62 %	133

Cuadro 7.4: Tabla de resultados de experimentación realizada en sistema integrado. La búsqueda sin control está referida a la búsqueda a temperatura constante de la sección 7.4.5. Las decisiones acertadas se caracterizan por dirigir el movimiento del sistema a la fuente. La localización es el porcentajes del total de experimentos que alcanza finalmente la fuente. Se observa como el cambio de tipo de fuente no afecta al sistema y como la explotación del algoritmo sin uso de la modificación de temperatura no localiza la fuente de odorante.

La comparativa de la tabla 7.4 muestra, en primer lugar, como el cambio de fuente de odorante no repercute en una disminución del porcentaje de acierto unidimensional, mientras en el caso bidimensional la fuente lineal ha quedado fuera de esta comparativa debido a su falta de efectividad.

El porcentaje de localización total del problema bidimensional es menor de lo esperado, la complejidad de la toma de decisión bidimensional hace que el sistema integrado tenga aún mucho margen de mejora. Como puntualizamos anteriormente este sistema se encuentra en desarrollo y optimización de parámetros del algoritmo.

Finalmente la comparativa realizada entre el algoritmo de localización y el análogo sin control de temperatura junto con las figuras comparativas 7.24 demuestra como las estrategias adaptativas de modificación de la temperatura son efectivas para el problema de la localización de odorante.

Se ha editado un vídeo con los experimentos demostrativos de la localización del sistema integrado en dos dimensiones [61].

8 Conclusiones y trabajo futuro

En este capítulo se expondrán las conclusiones que se han obtenido de la realización de este proyecto con relación a los objetivos inicialmente planteados de la sección 1.2, así como la consecución de los mismos a lo largo de este trabajo.
Posteriormente se expondrá el trabajo futuro fruto de las reflexiones que han surgido en el transcurso y que podrán ser desarrolladas posteriormente partiendo con este proyecto como base.

8.1. Conclusiones y resumen de resultados

8.1.1. Conclusiones

Este proyecto está enfocado en un entorno multidisciplinar, las tareas que se han realizado se engloban en los campos de la robótica, la instrumentación electrónica, la teoría de control y la computación adaptativa.

La integración de sistemas olfativos en plataformas robóticas para el uso de algoritmos de localización era el reto inicial con el que se planteó el proyecto, desde el punto de vista de los algoritmos se han estudiado diferentes técnicas del actual estado del arte que ponen de manifiesto la complejidad del problema que se obtiene al aplicar el análisis de la dispersión de fluidos en los diferentes entornos. Este último factor es debido a la cantidad de parámetros externos que los definen y a la multitud de cambios que ocurren en el ambiente.

Como investigación básica se introdujo el problema desde la perspectiva sensorial del sistema Olus, obteniendo pleno control sobre la plataforma se ha conseguido el conocimiento básico para descifrar la perspectiva de la dispersión de odorantes en entornos controlados sin presencia de obstáculos. Los sensores tipo MOS son un tipo de sensor de bajo coste pero con elevado efecto de histéresis, discriminar los estados por los que pasan las señales de odorante en este tipo de sensor como hemos demostrado en la memoria es un problema altamente complejo.

La selección de la plataforma robótica ha beneficiado la facilidad de integración de otro tipo de sistemas y la utilización de un entorno libre, estas características nos permiten en la mayoría de las situaciones adaptar el sistema para la aplicación en diversos tipos de experimentación tanto de algoritmos de localización como tareas más complejas relacionadas con el campo de la algoritmia.

La integración de la plataforma con las narices finales diseñadas a lo largo del capítulo 5 resultó más compleja de lo esperado en el sentido electrónico debido a las particularidades del sistema con respecto la modulación de temperatura. Físicamente, la posibilidad de integrar piezas diseñadas con impresora 3D resultó un recurso muy útil. En algunos casos y debido al caracter de prototipo del sistema se han tenido que reparar piezas o se han adaptado otras como en el ejemplo de la reducción de diámetro de las ruedas mejorando la resolución de los codificadores de rueda.

El microcontrolador de la plataforma nos ha permitido la creación de un sistema complejo ante la posibilidad de integración con cualquier otro tipo de sensores creando una plataforma multisensorial. Los microcontroladores con sistema embebido ofrecen mejores características y flexibilidad para el tipo de problemas del ámbito multidisciplinar.

El estudio de las técnicas de la teoría de control junto con la aplicación de las estrategias de calefacción de los sensores se introdujeron en el ámbito experimental generando una serie de estrategias que concluyeron en el algoritmo de localización. En el desarrollo de los experimentos ha quedado demostrada que la modificación de la sensibilidad de los sensores en bucle cerrado puede aplicarse en la localización para obtener adaptabilidad del sistema al entorno.

El sistema Olus con el algoritmo desarrollado ha obtenido resultados muy positivos en la localización unidimensional. Debido a la complejidad de la comunicación con los encapsulados y a la ineficiencia de la utilización de dos microprocesadores se decidió diseñar el sistema integrado, éste se encuentra en fase de desarrollo debido a la complejidad del ajuste de parámetros internos del controlador PID (PI) que es el mecanismo por el que obtenemos la adaptación al entorno de dispersión de odorantes.
Los parámetros del sistema Olus no son aplicables en el sistema integrado como inicialmente se esperaba, aunque el sistema integrado desarrollado ha demostrado valores confiables, el envejecimiento de los sensores del sistema Olus confiere capacidades fisicoquímicas alteradas respecto los sensores nuevos como queda reflejado en el experimento conjunto de la sección 7.4.4. Este efecto es debido a que los sensores del sistema Olus han sido anteriormente expuestos por un periodo del orden de 1 año a temperatura y humedad extrema, en particular, se situaron al aire libre en un pantano.

El desarrollo del protocolo de decisión con rotación de los sensores resultó ser un protocolo más flexible debido a la independencia de la decisión final con respecto los valores de *offset* entre sensores. Esta independencia repercute en que hemos podido realizar varios experimentos consecutivos con un único protocolo inicial puesto que en anteriores ocasiones el efecto de histéresis modificaba la estabilidad y por consiguiente el ajuste de *offset* entre sensores. Por otro lado, este protocolo ha repercutido en decisiones más lentas debido al mayor número de muestras y su funcionamiento puede quedar sesgado por errores en la ejecución de la rotación de la plataforma.

Es importante destacar que el enfoque de un algoritmo de localización basado en el control de las propiedades de los sensores es un tipo de estrategia hasta ahora poco estudiada. Las decisiones del algoritmo están basadas en la información obtenida de los sensores y por lo tanto es posible una compatibilidad con otro tipo de estrategias de decisión.
El diseño híbrido de algoritmos integrando otro tipo de estrategias de localización como *infotaxis* o los métodos de aprendizaje automático aplicados de forma simultanea con las estrategias actuales de explotación de información de los sensores ya conforman algunas de las ideas de trabajo futuro respecto la linea de investigación de los algoritmos de localización.

8.1.2. Resumen de objetivos y resultados

A continuación expondremos un resumen de los resultados obtenidos con relación a los objetivos planteados en el capítulo 1.

1. *Objetivo: Estudio y análisis del estado del arte respecto algoritmos de búsqueda de odorantes, estrategias bioinspiradas y entornos de dispersión de fluidos.*

   El desarrollo de este objetivo queda expuesto en el capítulo 2.
   El estudio de los sensores y el análisis de la dispersión de fuentes han sido el bloque inicial sobre el que se inicio el desarrollo de este proyecto, las técnicas aplicadas y los resultados de los experimentos previos son resultado de estos estudios y finalizaron con la parametrización del sistema del capítulo 4.

   La selección de los sensores de odorantes, el apoyo de sensores de otro tipo y las estrategias de localización por gradiente han resultado los factores determinantes del diseño de nuestro sistema.

2. *Objetivo: Utilización de narices olfativas Olus, proporcionadas por el GNB, para la aplicación de estrategias de captación, modulación y parametrización de los odorantes según percepción sensorial de la nariz.*

   El desarrollo de este objetivo queda reflejado en el capítulo 4 en el cual se obtienen los parámetros óptimos del sistema para la aplicación de las técnicas modulatorias y la aplicación en los entornos reales de búsqueda de odorantes.
   El estudio de los protocolos de comunicación de los encapsulados se explica con detalle en la sección 6.2.1 y junto con la aplicación de las librerías externas proporcionadas, conformamos el algoritmo inicial de muestreo de la sección 6.2.3. El sistema final de localización basado en estos encapsulados está condicionado por la comunicación y el uso de los microprocesadores del encapsulado siendo un sistema poco eficiente en la integración con plataformas robóticas.

3. *Objetivo: Estudio de la dispersión de odorantes y parámetros a tener en cuenta en el diseño de los algoritmos de búsqueda.*

   La aplicación de la teoría de control ha resultado en el controlador PID (PI) que modifica la temperatura en ciclo cerrado para adaptarse a los medios de dispersión, se explica su comportamiento en la sección 2.4.1 y su aplicación en el algoritmo en la sección 7.3.
   El problema de la localización de odorantes es muy complejo debido al numero de factores que lo influencian y condicionan, tanto parámetros internos respecto al sistema como externos referidos a la dispersión. Por consiguiente, el numero de factores exploratorios del sistema es muy alto y su exploración costosa.
   Hemos seleccionado el subconjunto de estos factores de estudio por medio de la intuición y análisis de los resultados. La experimentación ha tenido en cuenta las fases de la dispersión, el análisis de los experimentos previos se han realizado a diferentes distancias y se han realizado experimentos con los diferentes tipos de fuente de la sección 4.4.3.

4. *Objetivo: Integración en un plataforma robótica apropiada para la creación de un sistema completo de localización.*

   Se ha descrito con detalle la plataforma robótica seleccionada y sus componentes en el capítulo 3.
   Se han integrado piezas externas diseñadas con una impresora 3D, sección 3.3.2. Se adaptó la teoría de control en el movimiento de la plataforma robótica para la mejora en el recorrido de largas distancias. La plataforma BBB ha resultado ser un microcontrolador con alta capacidad de integración multisensor, como resultado, se ha añadido un sensor de humedad y temperatura observable en la sección 6.3.1.
   Los experimentos conjuntos del sistema de plataforma y unidad sensorial han podido realizarse correctamente en una y dos dimensiones como se expone en la sección de experimentación 7.4.

5. *Objetivo: Acondicionamiento de sensores comerciales modelo TGS [9] para aplicación de técnicas modulatorias de temperatura de calefacción en placas microcontroladoras y diseño de algoritmos de localización basados en el mismo.*

   El diseño y evolución de las narices electrónicas tiene reservado el capítulo 5. En él finalmente queda caracterizado el sistema diseñado para la utilización de las técnicas modulatorias expuestas en la sección 4.3 para las que no está inicialmente preparado el sensor que integramos TGS [9]. De las tecnologías utilizadas en el campo de la percepción olfativa artificial, este sensor se caracteriza por ser de bajo coste y tamaño, este tipo de sensores son poco exactos en sus mediciones pero son teóricamente de fácil acondicionamiento e integración en sistemas olfativos artificiales.
   Se ha implementado un sistema final de localización autónomo, móvil e integrado en el microcontrolador de la plataforma robótica. Los experimentos de localización con la plataforma han sido realizados de forma correcta, pueden observarse en la sección 7.4.5.

6. *Objetivo: Diseño de algoritmos de localización basados en técnicas de control y utilización de varias narices electrónicas simultáneamente en una única plataforma robótica.*

   Se han desarrollado algoritmos de localización basados en la modificación de temperatura en la sección 7.2 que han evolucionado hacia el algoritmo de localización de la sección 7.3 que explotamos en los sistemas olfativos artificiales. El algoritmo desarrollado tiene una clara componente de adaptación al entorno adquiriendo automáticamente dinámicas que protegen a los sensores, filtran plumas esporádicas del entorno y maximizan los vectores de movimiento que dirigen la plataforma robótica.
   El diseño de algoritmos de localización para la explotación en plataformas móviles tienen una complejidad añadida debido a las perturbaciones inducidas por la misma plataforma y a los errores del sistema de control.

7. *Objetivo: Experimentos de localización.*

   Finalmente se han diseñado los protocolos de experimentación para el uso de los sistemas y algoritmos desarrollados en la localización de las fuentes de odorantes en la sección 7.4. Se ha demostrado que la aplicación de métodos de modificación de temperatura en ciclo cerrado son efectivos para el problema de la localización de fuentes de odorante cuya comparativa de la sección 7.4.5 hace referencia. Estos métodos están vinculados con la estabilidad de la señal concentración procedente de los sensores debido a la adaptabilidad al entorno que muestran.

8.2. Trabajo Futuro

Este proyecto es el primer paso en la aplicación de los algoritmos de localización de odorantes, fruto de la creación de una plataforma de experimentación y de los estudios realizados en el campo podemos enumerar una serie de ideas que pueden desarrollarse en trabajos futuros. Hemos querido organizar estos trabajos para su correcta interpretación en: mejoras del sistema desarrollado, trabajo con relación a los sensores y temas relacionados con los algoritmos y los usos del sistema.

8.2.1. Mejoras del sistema desarrollado

- Mejora del movimiento en la plataforma. Mejora del movimiento de la plataforma para que sea un movimiento continuo y controlado gracias al controlador PID(PD) integrado en el movimiento del robot. El parámetro diferencial quedaría ajustado de forma dinámica a través de un sistema de coordenadas preestablecido.
- Caracterización dinámica del paso. Con objeto de la optimización del número de decisiones por búsqueda y por lo tanto el tiempo, se podría parametrizar de forma dinámica la distancia del paso en la plataforma robótica en función de la certeza de las señales obtenidas.
- Array de sensores. Integración de array de sensores de odorantes y de otros tipos en las estrategias de localización.
- Interfaz de usuario. Interfaz de usuario para la comunicación con el sistema. Cambio de parámetros del algoritmo o modificación del posicionamiento para una experimentación más flexible.
- Dinámica en narices electrónicas. Introducción de un servo de posicionamiento para las narices electrónicas que sustituya el actual giro de ruedas con el consiguiente aumento de las turbulencias en el entorno.

8.2.2. Sensores

- Reciclado de sensores antiguos. Debido al descubrimiento del estado de los sensores integrados en el encapsulado Olus, puede iniciarse un estudio más amplio acerca de la reutilización de este tipo de sensores en sistemas de bajo coste aprovechando las derivas del sistema.
- Diseño de placa PCB para las narices electrónicas desarrolladas en el sistema integrado.
- Optimización parámetros de modulación. Ajuste de los parámetros de modulación para la mejora de los algoritmos en el sistema integrado.

8.2.3. Algoritmos y usos

- Aprendizaje automático. Aplicación de metodologías de aprendizaje automático para la obtención de parámetros en localización como la longitud a la fuente o verificación de coherencia en la toma de decisión.
- Integración de sistemas de localización híbridos. Introducción de las estrategias de localización del estado del arte en los algoritmos desarrollados a nivel de sensores para la mejora de la toma de decisión en la localización de odorantes.
- Análisis de dispersión de fluidos. Uso de la plataforma de cuatro sensores como unidad básica para la monitorización y estudio de la dispersión de odorantes en diferentes entornos.

Glosario de acrónimos

- **BBB**: BeagleBone Black
- **GNB**: Grupo de Neurocomputación Biológica
- **MOS**: Metal Oxide Semiconductors
- **PWM**: Pulse Wide Modulation
- **PID**: Proporcional Integrativo Derivativo

Bibliografía

[1] David J Yáñez, Adolfo Toledano, Eduardo Serrano, Ana M Martín de Rosales, Francisco B Rodríguez, and Pablo Varona. Characterization of a clinical olfactory test with an artificial nose. *Frontiers in neuroengineering*, 5, 2012.

[2] Fernando Herrero-Carrón, David J Yáñez, Francisco de Borja Rodríguez, and Pablo Varona. An active, inverse temperature modulation strategy for single sensor odorant classification. *Sensors and Actuators B: Chemical*, 2014.

[3] Gerald Coley of BeagleBoard.org. Extraido de beaglebone black system reference manual. creative commons attribution-share alike 3.0 unported., 2012.

[4] Hiroshi Ishida, Yuta Wada, and Haruka Matsukura. Chemical sensing in robotic applications: A review. *Sensors Journal, IEEE*, 12(11):3163–3173, 2012.

[5] Mirbek Turduev, Gonçalo Cabrita, Murat Kırtay, Veysel Gazi, and Lino Marques. Experimental studies on chemical concentration map building by a multi-robot system using bio-inspired algorithms. *Autonomous agents and multi-agent systems*, 28(1):72–100, 2014.

[6] B Lorena Villarreal, Gustavo Olague, and JL Gordillo. Odor plume tracking algorithm inspired on evolution. In *Pattern Recognition*, pages 321–330. Springer, 2014.

[7] Victor Hernandez Bennetts, Achim J Lilienthal, Patrick P Neumann, and Marco Trincavelli. Mobile robots for localizing gas emission sources on landfill sites: is bio-inspiration the way to go? *Frontiers in neuroengineering*, 4, 2011.

[8] David J. Yáñez Villarreal. Estrategias bioinspiradas para la adquisición de olores en narices artificiales. Master's thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid., 2009.

[9] FIGARO GAS SENSOR TGS. Tgs2620/tgs2611 figaro engineering inc. *Japan (Nov. 1, 1978)*, 1978.

[10] Carlos García-Saura, Francisco de Borja Rodríguez, and Pablo Varona. Design principles for cooperative robots with uncertainty-aware and resource-wise adaptive behavior. In *Biomimetic and Biohybrid Systems*, pages 108–117. Springer, 2014.

[11] Tomas Vazquez Rubio. Integración de una nariz electrónica ultra-portátil en un robot modular para el control de su movimiento a través de los odorantes recibidos. Master's thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid., 2013.

[12] Carlos Garcia-Saura. Cooperative strategies for the detection and localization of odorants with robots and artificial noses. Master's thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid., 2014.

[13] Frank W Grasso, Thomas R Consi, David C Mountain, and Jelle Atema. Biomimetic robot lobster performs chemo-orientation in turbulence using a pair of spatially separated sensors: Progress and challenges. *Robotics and Autonomous Systems*, 30(1):115–131, 2000.

[14] Massimo Vergassola, Emmanuel Villermaux, and Boris I Shraiman. ïnfotaxis.ªs a strategy for searching without gradients. *Nature*, 445(7126):406–409, 2007.

[15] Javier Gonzalez-Jimenez, Javier G Monroy, and Jose Luis Blanco. The multi-chamber electronic nose?an improved olfaction sensor for mobile robotics. *Sensors*, 11(6):6145–6164, 2011.

[16] Gideon Kowadlo and R Andrew Russell. Robot odor localization: a taxonomy and survey. *The International Journal of Robotics Research*, 27(8):869–894, 2008.

[17] Ahmet Kuzu, Seta Bogosyan, and Metin Gokasan. Survey: detection, recognition and source localization of odor. *WSEAS Transactions on Systems*, 7(6):611–621, 2008.

[18] Tim C Pearce, Susan S Schiffman, H Troy Nagle, and Julian W Gardner. *Handbook of machine olfaction: electronic nose technology*. John Wiley & Sons, 2006.

[19] Figaro. Operating principle in mos type sensors. Technical report, Figaro. http://www.figarosensor.com/technicalinfo/principle/mos-type.html.

[20] Edward Lansing Cussler. *Diffusion: mass transfer in fluid systems*. Cambridge university press, 2009.

[21] Marcel Lesieur. *Turbulence in fluids*, volume 84. Springer Science & Business Media, 2008.

[22] Mark T Stacey, Edwin A Cowen, Thomas M Powell, Elizabeth Dobbins, Stephen G Monismith, and Jeffrey R Koseff. Plume dispersion in a stratified, near-coastal flow: measurements and modeling. *Continental Shelf Research*, 20(6):637–663, 2000.

[23] DR Webster and MJ Weissburg. Chemosensory guidance cues in a turbulent chemical odor plume. *Limnology and Oceanography*, 46(5):1034–1047, 2001.

[24] DR Webster, KY Volyanskyy, and MJ Weissburg. Bioinspired algorithm for autonomous sensor-driven guidance in turbulent chemical plumes. *Bioinspiration & biomimetics*, 7(3):036023, 2012.

[25] Jelle Atema. Eddy chemotaxis and odor landscapes: exploration of nature with animal sensors. *Biological Bulletin*, pages 129–138, 1996.

[26] JE Fackrell and AG Robins. Concentration fluctuations and fluxes in plumes from point sources in a turbulent boundary layer. *Journal of Fluid Mechanics*, 117:1–26, 1982.

[27] Ramon Huerta and Thomas Nowotny. Bio-inspired solutions to the challenges of chemical sensing. *Frontiers in neuroengineering*, 5, 2012.

[28] Mirbek Turduev, Murat Kirtay, Pedro Sousa, Veysel Gazi, and Lino Marques. Chemical concentration map building through bacterial foraging optimization based search algorithm by mobile robots. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 3242–3249. IEEE, 2010.

[29] Tien-Fu Lu. Indoor odour source localisation using robot: Initial location and surge distance matter? *Robotics and Autonomous Systems*, 61(6):637–647, 2013.

[30] Mohamed Awadalla, Tien-Fu Lu, Zhao F Tian, Bassam Dally, and Zhenzhang Liu. 3d framework combining cfd and matlab techniques for plume source localization research. *Building and Environment*, 70:10–19, 2013.

[31] Riaan Brits, Andries P Engelbrecht, and F Van den Bergh. A niching particle swarm optimizer. In *Proceedings of the 4th Asia-Pacific conference on simulated evolution and learning*, volume 2, pages 692–696. Singapore: Orchid Country Club, 2002.

[32] Andrew M Hein and Scott A McKinley. Sensing and decision-making in random search. *Proceedings of the National Academy of Sciences*, 109(30):12070–12074, 2012.

[33] Simon Benhamou. How many animals really do the levy walk? *Ecology*, 88(8):1962–1969, 2007.

[34] Frederic Bartumeus, Jordi Catalan, UL Fulco, ML Lyra, and GM Viswanathan. Optimizing the encounter rate in biological interactions: Lévy versus brownian strategies. *Physical Review Letters*, 88(9):097901, 2002.

[35] R Andrew Russell, Alireza Bab-Hadiashar, Rod L Shepherd, and Gordon G Wallace. A comparison of reactive robot chemotaxis algorithms. *Robotics and Autonomous Systems*, 45(2):83–97, 2003.

[36] Eduardo Martin Moraud and Dominique Martinez. Effectiveness and robustness of robot infotaxis for searching in dilute conditions. *Frontiers in neurorobotics*, 4, 2010.

[37] Carlo Barbieri, Simona Cocco, and Rémi Monasson. On the trajectories and performance of infotaxis, an information-based greedy search algorithm. *EPL (Europhysics Letters)*, 94(2):20005, 2011.

[38] Dominique Martinez, Lotfi Arhidi, Elodie Demondion, Jean-Baptiste Masson, and Philippe Lucas. Using insect electroantennogram sensors on autonomous robots for olfactory searches. *JoVE (Journal of Visualized Experiments)*, (90):e51704–e51704, 2014.

[39] David J Harvey, Tien-Fu Lu, and Michael A Keller. Effectiveness of insect-inspired chemical plume-tracking algorithms in a shifting wind field. *Robotics, IEEE Transactions on*, 24(1):196–201, 2008.

[40] Wei Li, Carl Bloomquist, Mohamoud M Elgassier, and Karan Srivasta. Multisensor integration for declaring the odor source of a plume in turbulent fluid-advected environments. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5534–5539. IEEE, 2006.

[41] Michihisa Iida, Donghyeon Kang, Mitsuru Taniwaki, Mutsumi Tanaka, and Mikio Umeda. Localization of co 2 source by a hexapod robot equipped with an anemoscope and a gas sensor. *computers and electronics in agriculture*, 63(1):73–80, 2008.

[42] Ji-Gong Li, Qing-Hao Meng, Yang Wang, and Ming Zeng. Odor source localization using a mobile robot in outdoor airflow environments with a particle filter algorithm. *Autonomous Robots*, 30(3):281–292, 2011.

[43] Hiroshi Ishida, Gouki Nakayama, Takamichi Nakamoto, and Toyosaka Moriizumi. Controlling a gas/odor plume-tracking robot based on transient responses of gas sensors. *Sensors Journal, IEEE*, 5(3):537–545, 2005.

[44] Montep Kiatweerasakul and TJ Stonham. Odour plume tracking robot using semiconductor gas sensors. In *Control, Automation, Robotics and Vision, 2002. ICARCV 2002. 7th International Conference on*, volume 2, pages 815–819. IEEE, 2002.

[45] GCHE De Croon, LM O'connor, C Nicol, and Dario Izzo. Evolutionary robotics approach to odor source localization. *Neurocomputing*, 121:481–497, 2013.

[46] B Lorena Villarreal, Christian Hassard, and José Luis Gordillo. Finding the direction of an odor source by using biologically inspired smell system. In *Advances in Artificial Intelligence–IBERAMIA 2012*, pages 551–560. Springer, 2012.

[47] Katsuhiko Ogata and Yanjuan Yang. Modern control engineering. fifth edition. 1970.

[48] TravTigerEE. Source: https://commons.wikimedia.org. creative commons attribution-share alike 3.0 unported, 2013.

[49] John G Ziegler and Nathaniel B Nichols. Optimum settings for automatic controllers. *trans. ASME*, 64(11), 1942.

[50] Rowland O'Flaherty. O'botics. http://o-botics.org/. code: https://github.com/o-botics, 2012.

[51] The arduino.cc team. Source: http://arduino.cc/it/tutorial/pwm. creative commons attribution-share alike 3.0 unported, 2012.

[52] Adafruit team. Link: https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/installation., 2012.

[53] *Nanostructured Tin Dioxide Materials for Gas Sensor Applications*, chapter 30. Functional Nanomaterials, 2006.

[54] R Gutierrez-Osuna, A Gutierrez-Galvez, and N Powar. Transient response analysis for temperature-modulated chemoresistors. *Sensors and Actuators B: Chemical*, 93(1):57–66, 2003.

[55] R Chutia and M Bhuyan. Study of temperature modulated tin oxide gas sensor and identification of chemicals. In *Computational Intelligence and Signal Processing (CISP), 2012 2nd National Conference on*, pages 181–184. IEEE, 2012.

[56] Andrew P Lee and Brian J Reedy. Temperature modulation in semiconductor gas sensing. *Sensors and Actuators B: Chemical*, 60(1):35–42, 1999.

[57] Eugenio Martinelli, Davide Polese, Alexandro Catini, Arnaldo D?Amico, and Corrado Di Natale. Self-adapted temperature modulation in metal-oxide semiconductor gas sensors. *Sensors and Actuators B: Chemical*, 161(1):534–541, 2012.

[58] Kal Mustafa. Filtering techniques: Isolating analog and digital powersupplies in ti's pll-based cdc devices. Technical report, Texas Instruments. http://www.ti.com/lit/an/scaa048/scaa048.pdf, 2001.

[59] Boost libraries for c++. http://www.boost.org/.

[60] Video de localización unidimensional, sistema olus. https://youtu.be/Z-omVyKD5Yg.

[61] Video de localización bidimensional, sistema integrado. http://youtu.be/-R-OyCITK30.

A

Esquematicos

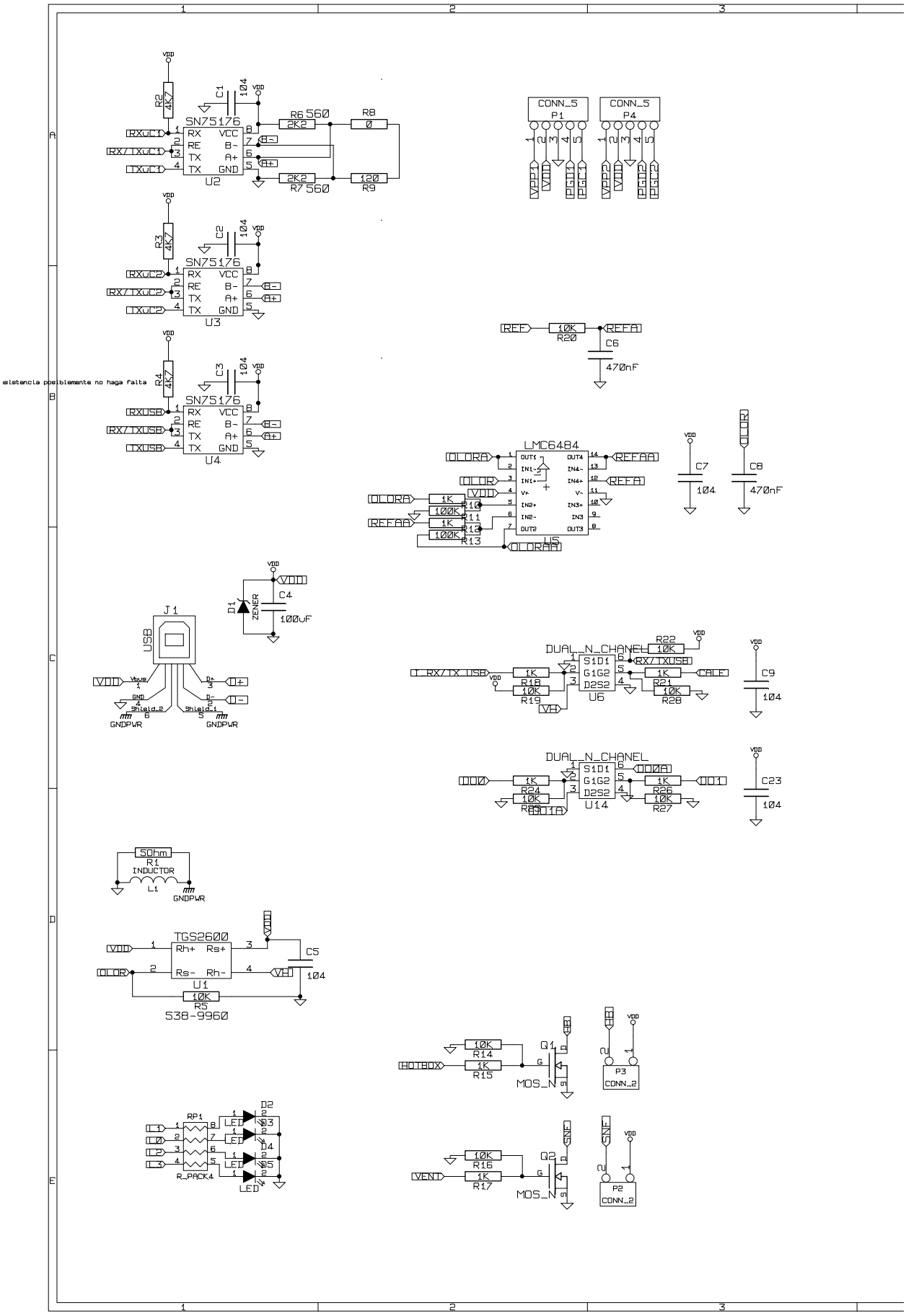


Figura A.1: Esquemático Olus Parte 1

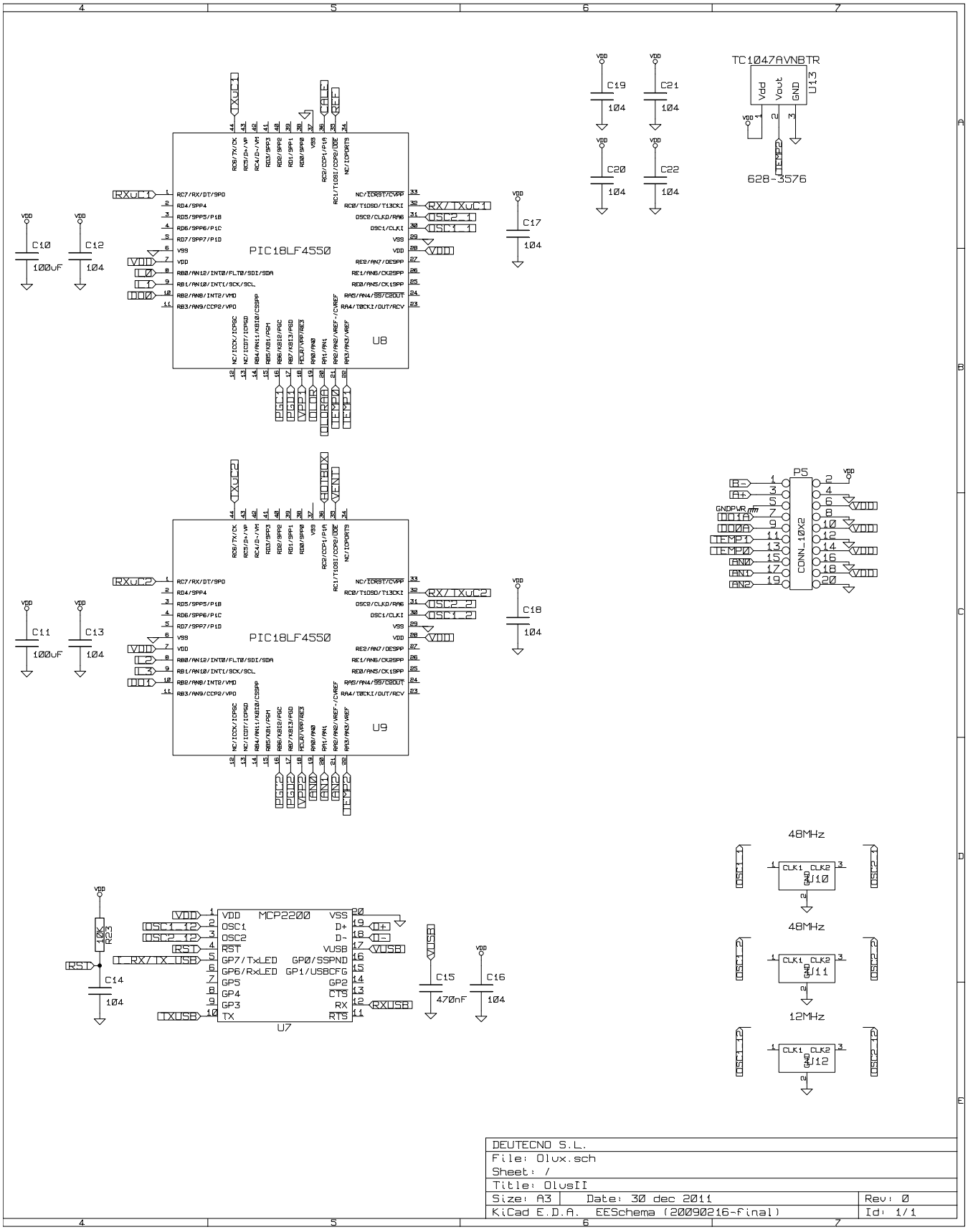


Figura A.2: Esquemático Olus Parte 2

B
Plataforma robótica

B.1. Listado de componentes

Parte	Nombre	Cantidad
Cable Conexion Prototipado M/M	Breadboard jumper wire M/M	30
Cable Conexion Prototipado M/F	Jumper Wires Premium 6"M/F	10
Adaptador Wifi	Compact USB Wi-Fi Adapter with 4.^ntenna	1
Regulador de tension 5V	LDO Voltage Regulators 5.0V 3.0A Positive - LD1085V50	1
Condensador 100uF	Electrolytic Capacitors - Leaded 100uF 25V	1
Condensador 10uF	Electrolytic Capacitors - Leaded 10uF 16V	1
Condensador 0.1uF	Ceramic Capacitors - Leaded 0.1uF 50V	1
Diodo	Rectifier diode 1N4007	1
Resistencia 20K	Resistors - Through hole 1/4W 20k ohm	7
Resistencia 10K	Resistors - Through hole 1/4W 10k ohm	7
Microcontrolador	BeagleBone Black	1
Adaptador Jack	DC Barrel Jack Adapter - Female	2
Conector Baterias	Battery Holder - 4xAA Square	2
Chasis del Robot	Magician Chassis	1
Codificador de rueda	RedBot Wheel Encoder	1
Placa Prototipado Mini	Breadboard - Mini Modular (White)	1
Placa Prototipado	Breadboard - Self-Adhesive (White)	1
Tornillo (Largo)	Screw - Phillips Head (1/2", 4-40, 10 pack)	2
Tornillo Largo (Corto)	Screw - Phillips Head (1/4", 4-40, 10 pack)	2
Tuercas	Nut - Metal (4-40, 10 pack)	2
Driver de potencia	H-Bridge Motor Driver 1A	1
Disipador Termico	Heatsink TO-220	1
Interruptor	Rocker Switch - SPST (right-angle)	1
Baterias Recargables	AA Ni-MH Rechargeable Batteries	8

Cuadro B.1: Lista de componentes de la plataforma robótica necesarios para su construcción y conexionado.

B.2. BBB. Pines de conexión y puertos de expansión

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3
1,2	GND					
3	R9	GPIO1-6	gpmc-ad6	mmc1-dat6		
4	T9	GPIO1-7	gpmc-ad7	mmc1-dat7		
5	R8	GPIO1-2	gpmc-ad2	mmc1-dat2		
6	T8	GPIO1-3	gpmc-ad3	mmc1-dat3		
7	R7	TIMER4	gpmc-advn-ale		timer4	
8	T7	TIMER7	gpmc-oen-ren		timer7	
9	T6	TIMER5	gpmc-be0n-cle		timer5	
10	U6	TIMER6	gpmc-wen		timer6	
11	R12	GPIO1-13	gpmc-ad13	lcd-data18	mmc1-dat5	mmc2-dat1
12	T12	GPIO1-12	gpmc-ad12	Lcd-data19	mmc1-dat4	Mmc2-dat0
13	T10	EHRPWM2B	gpmc-ad9	lcd-data22	mmc1-dat1	mmc2-dat5
14	T11	GPIO0-26	gpmc-ad10	lcd-data21	mmc1-dat2	mmc2-dat6
15	U13	GPIO1-15	gpmc-ad15	lcd-data16	mmc1-dat7	mmc2-dat3
16	V13	GPIO1-14	gpmc-ad14	lcd-data17	mmc1-dat6	mmc2-dat2
17	U12	GPIO0-27	gpmc-ad11	lcd-data20	mmc1-dat3	mmc2-dat7
18	V12	GPIO2-1	gpmc-clk-mux0	lcd-memory-clk	gpmc-wait1	mmc2-clk
19	U10	EHRPWM2A	gpmc-ad8	lcd-data23	mmc1-dat0	mmc2-dat4
20	V9	GPIO1-31	gpmc-csn2	gpmc-be1n	mmc1-cmd	
21	U9	GPIO1-30	gpmc-csn1	gpmc-clk	mmc1-clk	
22	V8	GPIO1-5	gpmc-ad5	mmc1-dat5		
23	U8	GPIO1-4	gpmc-ad4	mmc1-dat4		
24	V7	GPIO1-1	gpmc-ad1	mmc1-dat1		
25	U7	GPIO1-0	gpmc-ad0	mmc1-dat0		
26	V6	GPIO1-29	gpmc-csn0			
27	U5	GPIO2-22	lcd-vsync	gpmc-a8		
28	V5	GPIO2-24	lcd-pclk	gpmc-a10		
29	R5	GPIO2-23	lcd-hsync	gpmc-a9		
30	R6	GPIO2-25	lcd-ac-bias-en	gpmc-a11		
31	V4	UART5-CTSN	lcd-data14	gpmc-a18	eQEP1-index	mcasp0-axr1
32	T5	UART5-RTSN	lcd-data15	gpmc-a19	eQEP1-strobe	mcasp0-ahclkx
33	V3	UART4-RTSN	lcd-data13	gpmc-a17	eQEP1B-in	mcasp0-fsr
34	U4	UART3-RTSN	lcd-data11	gpmc-a15	ehrpwm1B	mcasp0-ahclkr
35	V2	UART4-CTSN	lcd-data12	gpmc-a16	eQEP1A-in	mcasp0-aclkr
36	U3	UART3-CTSN	lcd-data10	gpmc-a14	ehrpwm1A	mcasp0-axr0
37	U1	UART5-TXD	lcd-data8	gpmc-a12	ehrpwm1-tripzone-in	mcasp0-aclkx
38	U2	UART5-RXD	lcd-data9	gpmc-a13	ehrpwm0-synco	mcasp0-fsx
39	T3	GPIO2-12	lcd-data6	gpmc-a6		eQEP2-index
40	T4	GPIO2-13	lcd-data7	gpmc-a7		eQEP2-strobe
41	T1	GPIO2-10	lcd-data4	gpmc-a4		eQEP2A-in
42	T2	GPIO2-11	lcd-data5	gpmc-a5		eQEP2B-in
43	R3	GPIO2-8	lcd-data2	gpmc-a2		ehrpwm2-tripzone-in
44	R4	GPIO2-9	lcd-data3	gpmc-a3		ehrpwm0-synco
45	R1	GPIO2-6	lcd-data0	gpmc-a0		ehrpwm2A
46	R2	GPIO2-7	lcd-data1	gpmc-a1		ehrpwm2B

Cuadro B.2: Pines de conexión. Modos de funcionamiento Puerto 8 (**P8**). Parte 1.

PIN	MODE4	MODE5	MODE6	MODE7
1,2	GND			
3				gpio1[6]
4				gpio1[7]
5				gpio1[2]
6				gpio1[3]
7				gpio2[2]
8				gpio2[3]
9				gpio2[5]
10				gpio2[4]
11	eQEP2B-in		pr1-pru0-pru-r30-15	gpio1[13]
12	Eqep2a-in		pr1-pru0-pru-r30-14	gpio1[12]
13	ehrpwm2B			gpio0[23]
14	ehrpwm2-tripzone-in			gpio0[26]
15	eQEP2-strobe		pr1-pru0-pru-r31-15	gpio1[15]
16	eQEP2-index		pr1-pru0-pru-r31-14	gpio1[14]
17	ehrpwm0-synco			gpio0[27]
18			mcasp0-fsr	gpio2[1]
19	ehrpwm2A			gpio0[22]
20		pr1-pru1-pru-r30-13	pr1-pru1-pru-r31-13	gpio1[31]
21		pr1-pru1-pru-r30-12	pr1-pru1-pru-r31-12	gpio1[30]
22				gpio1[5]
23				gpio1[4]
24				gpio1[1]
25				gpio1[0]
26				gpio1[29]
27		pr1-pru1-pru-r30-8	pr1-pru1-pru-r31-8	gpio2[22]
28		pr1-pru1-pru-r30-10	pr1-pru1-pru-r31-10	gpio2[24]
29		pr1-pru1-pru-r30-9	pr1-pru1-pru-r31-9	gpio2[23]
30				gpio2[25]
31	uart5-rxd		uart5-ctsn	gpio0[10]
32	mcasp0-axr3		uart5-rtsn	gpio0[11]
33	mcasp0-axr3		uart4-rtsn	gpio0[9]
34	mcasp0-axr2		uart3-rtsn	gpio2[17]
35	mcasp0-axr2		uart4-ctsn	gpio0[8]
36			uart3-ctsn	gpio2[16]
37	uart5-txd		uart2-ctsn	gpio2[14]
38	uart5-rxd		uart2-rtsn	gpio2[15]
39		pr1-pru1-pru-r30-6	pr1-pru1-pru-r31-6	gpio2[12]
40	pr1-edio-data-out7	pr1-pru1-pru-r30-7	pr1-pru1-pru-r31-7	gpio2[13]
41		pr1-pru1-pru-r30-4	pr1-pru1-pru-r31-4	gpio2[10]
42		pr1-pru1-pru-r30-5	pr1-pru1-pru-r31-5	gpio2[11]
43		pr1-pru1-pru-r30-2	pr1-pru1-pru-r31-2	gpio2[8]
44		pr1-pru1-pru-r30-3	pr1-pru1-pru-r31-3	gpio2[9]
45		pr1-pru1-pru-r30-0	pr1-pru1-pru-r31-0	gpio2[6]
46		pr1-pru1-pru-r30-1	pr1-pru1-pru-r31-1	gpio2[7]

Cuadro B.3: Pines de conexión. Modos de funcionamiento Puerto 8 (**P8**). Parte 2.

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3
1,2						GND
3,4						DC-3.3V
5,6						VDD-5V
7,8						SYS-5V
9						PWR-BUT
10	A10					SYS-RESETnRESET-OUT
11	T17	UART4-RXD	gpmc-wait0	mii2-crs	gpmc-csn4	rmii2-crs-dv
12	U18	GPIO1-28	gpmc-be1n	mii2-col	gpmc-csn6	mmc2-dat3
13	U17	UART4-TXD	gpmc-wpn	mii2-rxerr	gpmc-csn5	rmii2-rxerr
14	U14	EHRPWM1A	gpmc-a2	mii2-txd3	rgmii2-td3	mmc2-dat1
15	R13	GPIO1-16	gpmc-a0	gmii2-txen	rmii2-tctl	mii2-txen
16	T14	EHRPWM1B	gpmc-a3	mii2-txd2	rgmii2-td2	mmc2-dat2
17	A16	I2C1-SCL	spi0-cs0	mmc2-sdwp	I2C1-SCL	ehrpwm0-synci
18	B16	I2C1-SDA	spi0-d1	mmc1-sdwp	I2C1-SDA	ehrpwm0-tripzone
19	D17	I2C2-SCL	uart1-rtsn	timer5	dcan0-rx	I2C2-SCL
20	D18	I2C2-SDA	uart1-ctsn	timer6	dcan0-tx	I2C2-SDA
21	B17	UART2-TXD	spi0-d0	uart2-txd	I2C2-SCL	ehrpwm0B
22	A17	UART2-RXD	spi0-sclk	uart2-rxd	I2C2-SDA	ehrpwm0A
23	V14	GPIO1-17	gpmc-a1	gmii2-rxdv	rgmii2-rxdv	mmc2-dat0
24	D15	UART1-TXD	uart1-txd	mmc2-sdwp	dcan1-rx	I2C1-SCL
25	A14	GPIO3-21*	mcasp0-ahclkx	eQEP0-strobe	mcasp0-axr3	mcasp1-axr1
26	D16	UART1-RXD	uart1-rxd	mmc1-sdwp	dcan1-tx	I2C1-SDA
27	C13	GPIO3-19	mcasp0-fsr	eQEP0B-in	mcasp0-axr3	mcasp1-fsx
28	C12	SPI1-CS0	mcasp0-ahclkr	ehrpwm0-synci	mcasp0-axr2	spi1-cs0
29	B13	SPI1-D0	mcasp0-fsx	ehrpwm0B		spi1-d0
30	D12	SPI1-D1	mcasp0-axr0	ehrpwm0-tripzone		spi1-d1
31	A13	SPI1-SCLK	mcasp0-aclkx	ehrpwm0A		spi1-sclk
32						VADC
33	C8					AIN4
34						AGND
35	A8					AIN6
36	B8					AIN5
37	B7					AIN2
38	A7					AIN3
39	B6					AIN0
40	C7					AIN1
41#	D14	CLKOUT2	xdma-event-intr1		tclkin	clkout2
	D13	GPIO3-20	mcasp0-axr1	eQEP0-index		Mcasp1-axr0
42	C18	GPIO0-7	eCAP0-in-PWM0-out	uart3-txd	spi1-cs1	pr1-ecap0-ecap-capin-apwm-o
	B12	GPIO3-18	Mcasp0-aclkr	eQEP0A-in	Mcaspo-axr2	Mcasp1-aclkx
43-46						GND

Cuadro B.4: Pines de conexión. Modos de funcionamiento Puerto 9 (**P9**). Parte 1.

PIN	MODE4	MODE5	MODE6	MODE7
1,2	GND			
3,4	DC-3.3V			
5,6	VDD-5V			
7,8	SYS-5V			
9	PWR-BUT			
10	SYS-RESETnRESET-OUT			
11	mmc1-sdcd		uart4-rxd-mux2	gpio0[30]
12	gpmc-dir		mcasp0-aclkr-mux3	gpio1[28]
13	mmc2-sdcd		uart4-txd-mux2	gpio0[31]
14	gpmc-a18		ehrpwm1A-mux1	gpio1[18]
15	gpmc-a16		ehrpwm1-tripzone-input	gpio1[16]
16	gpmc-a19		ehrpwm1B-mux1	gpio1[19]
17	pr1-uart0-txd			gpio0[5]
18	pr1-uart0-rxd			gpio0[4]
19	spi1-cs1	pr1-uart0-rts-n		gpio0[13]
20	spi1-cs0	pr1-uart0-cts-n		gpio0[12]
21	pr1-uart0-rts-n		EMU3-mux1	gpio0[3]
22	pr1-uart0-cts-n		EMU2-mux1	gpio0[2]
23	gpmc-a17		ehrpwm0-synco	gpio1[17]
24		pr1-uart0-txd	pr1-pru0-pru-r31-16	gpio0[15]
25	EMU4-mux2	pr1-pru0-pru-r30-7	pr1-pru0-pru-r31-7	gpio3[21]
26		pr1-uart0-rxd	pr1-pru1-pru-r31-16	gpio0[14]
27	EMU2-mux2	pr1-pru0-pru-r30-5	pr1-pru0-pru-r31-5	gpio3[19]
28	eCAP2-in-PWM2-out	pr1-pru0-pru-r30-3	pr1-pru0-pru-r31-3	gpio3[17]
29	mmc1-sdcd-mux1	pr1-pru0-pru-r30-1	pr1-pru0-pru-r31-1	gpio3[15]
30	mmc2-sdcd-mux1	pr1-pru0-pru-r30-2	pr1-pru0-pru-r31-2	gpio3[16]
31	mmc0-sdcd-mux1	pr1-pru0-pru-r30-0	pr1-pru0-pru-r31-0	gpio3[14]
32	VADC			
33	AIN4			
34	AGND			
35	AIN6			
36	AIN5			
37	AIN2			
38	AIN3			
39	AIN0			
40	AIN1			
41#	timer7-mux1	pr1-pru0-pru-r31-16	EMU3-mux0	gpio0[20]
	emu3	pr1-pru0-pru-r30-6	pr1-pru0-pru-r31-6	gpio3[20]
42@	spi1-sclk	mmc0-sdwp	xdma-event-intr2	gpio0[7]
		pr1-pru0-pru-r30-4	pr1-pru0-pru-r31-4	gpio3[18]
43-46	GND			

Cuadro B.5: Pines de conexión. Modos de funcionamiento Puerto 9 (**P9**). Parte 2.

C

Codigos de Ejecución

C.1. Codigo movimiento de la plataforma robótica

C.1.1. moveRobot.py

Código de movimiento de la plataforma robótica en linea recta.

```
 #!/usr/bin/python

import Adafruit_BBIO.ADC as ADC
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.PWM as PWM
import time
import threading
import sys

# Move Params
VELOCITY = [70,70]
minPWMThreshold = [67, 67]
maxPWMThreshold = [90, 90]
#Distance Odometry Error
DERIVAFL=20                     #Probabilistic adjust of the odometry
DERIVAFR=10                     #       F: Forward Move
DERIVABL=30                     #       B: Backward Move
DERIVABR=10                     #       L: Left Wheel
                                #       R: Right Wheel
#Velocity Error
ADJUSTVFL=0                     #Fixes Velocity adjust in Manual Mode
ADJUSTVFR=0
ADJUSTVBL=0
ADJUSTVBR=0

SAMPLESPOSTMOVE=1000            #Samples to measure post-finish move


DIFID=0                         #Objetive Difference between wheels for
    curve moves.(TESTING)
MANUALMODE=1
KP=0.12
```

```
KI=0
KD=0.008
ADJUSTSLEEP=0.05

# Constants
LEFT = 0
RIGHT = 1
UMBRAL = 0.6
UMBREP = 2              #Umbral de repeticion para contar tics
ticWheel = 14 #14               #65mm Diameter of the Wheel Wheel 16 tic
distPerRev= 110#204#110         #mm Distance per Revolution (pi*65)= 204,20

# Motor Pins -- (LEFT, RIGHT)
dir1Pin = ('P8_12', 'P8_14')
dir2Pin = ('P8_10', 'P8_16')
pwmPin = ('P9_14', 'P9_16')
encPin = ('P9_35','P9_36')

#Threading
ADCLock = threading.Lock()
ExitLock= threading.Lock()

#Globals
distance=[0,0]
exitFlag=0
DEBUG = 1

#Define threads of our system
class myThread (threading.Thread):
        def __init__(self, threadID, name, param1):
                threading.Thread.__init__(self)
                self.threadID = threadID
                self.name = name
                self.param1 = param1
        def run(self):
                # Get lock to synchronize threads
                print "Starting " + self.name
                # Free lock to release next thread
                if self.name == 'Tics':
                        measureDistance()
                elif self.name == 'Speed':
                        adjustSpeed(self.param1)

#Setup Pin Modes
def setup():
        ADC.setup()
        GPIO.setup(dir1Pin[LEFT], GPIO.OUT)
        GPIO.setup(dir2Pin[LEFT], GPIO.OUT)
        GPIO.setup(dir1Pin[RIGHT], GPIO.OUT)
        GPIO.setup(dir2Pin[RIGHT], GPIO.OUT)
        PWM.start(pwmPin[LEFT], 0)
        PWM.start(pwmPin[RIGHT], 0)

#Apply the value to the pwm on the wheels
def move(pwm):
        if pwm[LEFT] > 0:
            GPIO.output(dir1Pin[LEFT], GPIO.LOW)
```

```
89              GPIO.output(dir2Pin[LEFT], GPIO.HIGH)
90              PWM.set_duty_cycle(pwmPin[LEFT], abs(pwm[LEFT]))
91          elif pwm[LEFT] < 0:
92              GPIO.output(dir1Pin[LEFT], GPIO.HIGH)
93              GPIO.output(dir2Pin[LEFT], GPIO.LOW)
94              PWM.set_duty_cycle(pwmPin[LEFT], abs(pwm[LEFT]))
95          else:
96              GPIO.output(dir1Pin[LEFT], GPIO.LOW)
97              GPIO.output(dir2Pin[LEFT], GPIO.LOW)
98              PWM.set_duty_cycle(pwmPin[LEFT], 0)
99           # Right motor
100         if pwm[RIGHT] > 0:
101             GPIO.output(dir1Pin[RIGHT], GPIO.LOW)
102             GPIO.output(dir2Pin[RIGHT], GPIO.HIGH)
103             PWM.set_duty_cycle(pwmPin[RIGHT],abs(pwm[RIGHT]))
104         elif pwm[RIGHT] < 0:
105             GPIO.output(dir1Pin[RIGHT], GPIO.HIGH)
106             GPIO.output(dir2Pin[RIGHT], GPIO.LOW)
107             PWM.set_duty_cycle(pwmPin[RIGHT],abs(pwm[RIGHT]))
108         else:
109             GPIO.output(dir1Pin[RIGHT], GPIO.LOW)
110             GPIO.output(dir2Pin[RIGHT], GPIO.LOW)
111             PWM.set_duty_cycle(pwmPin[RIGHT], 0)
112
113 #Clean up and close connection with system pins
114 def cleanup():
115         print "Clean up"
116         move([0, 0])
117         GPIO.cleanup()
118         PWM.cleanup()
119
120 #Function to measure how the wheel has moved
121 def countTics(tic,state,value,repeticion):
122         #Left Wheel
123         if value[LEFT]>UMBRAL:
124                 state[LEFT]=1
125                 repeticion[LEFT]+=1
126         elif value[LEFT]<UMBRAL:
127                 if state[LEFT]==1 and repeticion[LEFT]>=UMBREP:
128                         tic[LEFT]+=1
129                 repeticion[LEFT]=0
130                 state[LEFT]=0
131
132         #Right Wheel
133         if value[RIGHT]>UMBRAL:
134                 state[RIGHT]=1
135                 repeticion[RIGHT]+=1
136         elif value[RIGHT]<UMBRAL:
137                 if state[RIGHT]==1 and repeticion[RIGHT]>=UMBREP:
138                         tic[RIGHT]+=1
139                 repeticion[RIGHT]=0
140                 state[RIGHT]=0
141         return (tic,state,repeticion)
142
143 #Function block to accumulate the total distance of the system
144 def measureDistance():
145         tic=[0,0]
146         state=[1,1]
147         repeticion=[0,0]
```

```
148             count=0
149             global distance
150             global exitFlag
151 
152             #fo = open("Tics.txt", "wb")
153 
154             ExitLock.acquire()
155             while not exitFlag and count<SAMPLESPOSTMOVE:
156                     ExitLock.release()
157 
158                     #Read Sensor
159                     value=[ADC.read(encPin[LEFT]),ADC.read(encPin[RIGHT])]
160                     [tic,state,repeticion]=countTics(tic,state,value,
                            repeticion)
161                     #Write Distance
162                     ADCLock.acquire()
163                     distance=[((float(tic[LEFT])/float(ticWheel))*
                            distPerRev),((float(tic[RIGHT])/float(ticWheel))*
                            distPerRev)]
164                     print 'LocalDistance:',distance[LEFT],distance[RIGHT]
165                     ADCLock.release()
166 
167                     ExitLock.acquire()
168                     if exitFlag:
169                             count+=1
170             ExitLock.release()
171 
172     #Adjust the speed with the distances and a PD control
173     def adjustSpeed(limit):
174             global distance
175             global exitFlag
176             if MANUALMODE==0:
177                     Kp=0
178                     Ki=0
179                     Kd=0
180             else:
181                     Kp=KP
182                     Ki=KI
183                     Kd=KD
184             erPOld=0
185             erI=0
186             gap=round(distPerRev/ticWheel)
187 
188             loop = 1
189 
190             if limit<0: #BackWard Move
191                     velocity=[-VELOCITY[LEFT]-ADJUSTVBL,-VELOCITY[RIGHT]-
                            ADJUSTVBR]
192             else:        #Forward Move
193                     velocity=[VELOCITY[LEFT]+ADJUSTVFL,VELOCITY[RIGHT]+
                            ADJUSTVFR]
194             velL=velocity[LEFT]
195             velR=velocity[RIGHT]
196             if velL>0 and velR>0:
197                     derivaL=DERIVAFL
198                     derivaR=DERIVAFR
199             elif velL<0 and velR<0:
200                     derivaL=DERIVABL
201                     derivaR=DERIVABR
```

```
202         else:
203                 derivaL=0
204                 derivaR=0
205
206         while loop:     #Adjust Speed Loop
207                 move([velL,velR])
208                 time.sleep(ADJUSTSLEEP)
209                 ADCLock.acquire()
210                 localDistance=[distance[LEFT]+derivaL,distance[RIGHT]+
                        derivaR]
211                 ADCLock.release()
212                 #Change de Speed
213                 if velL>=0 and velR>=0:
214                         difReal=localDistance[LEFT]-localDistance[RIGHT
                            ]
215                 elif velL<0 and velR<0:
216                         difReal=localDistance[RIGHT]-localDistance[LEFT
                            ]
217                 erP=difReal-DIFID
218                 erI+=erP
219                 erD=erP-erPOld
220                 erPOld=erP
221                 adjust=round(Kp*erP + Ki*erI - Kd*erD)
222                 print 'PWM:',velL,velR,adjust
223                 if localDistance[LEFT] > abs(limit) and localDistance[
                        RIGHT] > abs(limit) :
224                                 cleanup()
225                                 ExitLock.acquire()
226                                 exitFlag=1
227                                 ExitLock.release()
228                                 break #Or loop=0
229                 #Straight Line
230                 elif adjust>=0:
231                         if abs(velL-adjust)<minPWMThreshold[LEFT]:
232                                 velR+=adjust
233                         else:
234                                 velL-=adjust
235                 elif adjust<0:
236                         if abs(velR+adjust)<minPWMThreshold[RIGHT]:
237                                 velL-=adjust
238                         else:
239                                 velR+=adjust
240
241                 #Motor Protection
242                 if limit>0:
243                         velL=min(max(velL, minPWMThreshold[LEFT]),
                                maxPWMThreshold[LEFT])
244                         velR=min(max(velR, minPWMThreshold[RIGHT]),
                                maxPWMThreshold[RIGHT])
245                 elif limit<0:
246                         velL=max(min(velL, -minPWMThreshold[LEFT]), -
                                maxPWMThreshold[LEFT])
247                         velR=max(min(velR, -minPWMThreshold[RIGHT]), -
                                maxPWMThreshold[RIGHT])
248
249 def main():
250         # Program Ini Variables
251         threads = []
252
```

```
253         if(len(sys.argv)<2):
254                 print 'Error define parameter Distance[mm]'
255                 return 0
256         limitD=int(sys.argv[1])
257         setup()
258         #Create new Threads
259         thread1 = myThread(1, "Tics",'x')
260         thread2 = myThread(1, "Speed",limitD)
261         # Start new Threads
262         thread1.start()
263         thread2.start()
264         # Add threads to thread list
265         threads.append(thread1)
266         threads.append(thread2)
267         #Waitting until end
268         for t in threads:
269                 t.join()
270         print "Exiting Main Thread"
271         return 0
272
273 if __name__ == '__main__':
274     main()
```

C.1.2. turnRobot.py

Código de rotación de la plataforma robótica.
Valor de entrada numero de grados de la rotación
+: En contrasentido a las agujas del reloj
-: En sentido a las agujas del reloj

```
1
2   #!/usr/bin/python
3
4  import Adafruit_BBIO.ADC as ADC
5  import Adafruit_BBIO.GPIO as GPIO
6  import Adafruit_BBIO.PWM as PWM
7  import time
8  import threading
9  import sys
10
11 # Move Params
12 VELOCITY = [75,75]              #Velocity of wheels
13 minPWMThreshold = [75, 75]      #Minimun velocity
14 maxPWMThreshold = [100, 100]    #Maximun velocity
15 ADJUST=1                       #Test value. Adjusting velocity.
16
17 # Constants
18 LEFT = 0                #Index for Pins
19 RIGHT = 1               #       LEFT; RIGHT
20 UMBRAL = 0.6            #Edge of infrared measures in order to
       distinguish tics
21 UMBREP = 2              #Number of repetitions measures in order to
       count tics
22 ticWheel = 16           #65mm Diameter of the Wheel Wheel 16 tic
23 distPerRev=110          #mm Distance per Revolution (pi*65)= 204,20
24 distPerRound=345        #D=110mm => 345.  D*pi=314+ERROR
25
26 # Motor Pins -- (LEFT, RIGHT)
27 dir1Pin = ('P8_12', 'P8_14')
28 dir2Pin = ('P8_10', 'P8_16')
29 pwmPin = ('P9_14', 'P9_16')
30 encPin = ('P9_36','P9_35')
31
32 #Threading
33 ADCLock = threading.Lock()
34 ExitLock= threading.Lock()
35
36 #Globals
37 distance=[0,0]
38 exitFlag=0
39 DEBUG = 1
40
41 #Define threads of our system
42 class myThread (threading.Thread):
43         def __init__(self, threadID, name, param1,param2):
44                 threading.Thread.__init__(self)
45                 self.threadID = threadID
46                 self.name = name
47                 self.param1 = param1
48                 self.param2 = param2
49         def run(self):
50                 # Get lock to synchronize threads
```

```
51                  print "Starting " + self.name
52                  if self.name == 'Tics':
53                          measureDistance()
54                  elif self.name == 'Speed':
55                          adjustSpeed(self.param1,self.param2)
56
57  #Setup Pin Modes
58  def setup():
59          ADC.setup()
60          GPIO.setup(dir1Pin[LEFT], GPIO.OUT)
61          GPIO.setup(dir2Pin[LEFT], GPIO.OUT)
62          GPIO.setup(dir1Pin[RIGHT], GPIO.OUT)
63          GPIO.setup(dir2Pin[RIGHT], GPIO.OUT)
64          PWM.start(pwmPin[LEFT], 0)
65          PWM.start(pwmPin[RIGHT], 0)
66
67  #Apply the value to the pwm on the wheels
68  def move(pwm):
69          # Left motor
70          if pwm[LEFT] > 0:
71              GPIO.output(dir1Pin[LEFT], GPIO.LOW)
72              GPIO.output(dir2Pin[LEFT], GPIO.HIGH)
73              PWM.set_duty_cycle(pwmPin[LEFT], abs(pwm[LEFT]))
74          elif pwm[LEFT] < 0:
75              GPIO.output(dir1Pin[LEFT], GPIO.HIGH)
76              GPIO.output(dir2Pin[LEFT], GPIO.LOW)
77              PWM.set_duty_cycle(pwmPin[LEFT], abs(pwm[LEFT]))
78          else:
79              GPIO.output(dir1Pin[LEFT], GPIO.LOW)
80              GPIO.output(dir2Pin[LEFT], GPIO.LOW)
81              PWM.set_duty_cycle(pwmPin[LEFT], 0)
82
83          # Right motor
84          if pwm[RIGHT] > 0:
85              GPIO.output(dir1Pin[RIGHT], GPIO.LOW)
86              GPIO.output(dir2Pin[RIGHT], GPIO.HIGH)
87              PWM.set_duty_cycle(pwmPin[RIGHT],abs(pwm[RIGHT]))
88          elif pwm[RIGHT] < 0:
89              GPIO.output(dir1Pin[RIGHT], GPIO.HIGH)
90              GPIO.output(dir2Pin[RIGHT], GPIO.LOW)
91              PWM.set_duty_cycle(pwmPin[RIGHT],abs(pwm[RIGHT]))
92          else:
93              GPIO.output(dir1Pin[RIGHT], GPIO.LOW)
94              GPIO.output(dir2Pin[RIGHT], GPIO.LOW)
95              PWM.set_duty_cycle(pwmPin[RIGHT], 0)
96
97  #Clean up and close connection with system pins
98  def cleanup():
99          print "Clean up"
100         move([0, 0])
101         GPIO.cleanup()
102         PWM.cleanup()
103
104 #Function to measure how the wheel has moved
105 def countTics(tic,state,value,repeticion):
106         #Left Wheel
107         if value[LEFT]>UMBRAL:
108                 state[LEFT]=1
109                 repeticion[LEFT]+=1
```

```
110         elif value[LEFT]<UMBRAL:
111                 if state[LEFT]==1 and repeticion[LEFT]>=UMBREP:
112                         tic[LEFT]+=1
113                 repeticion[LEFT]=0
114                 state[LEFT]=0
115
116         #Right Wheel
117         if value[RIGHT]>UMBRAL:
118                 state[RIGHT]=1
119                 repeticion[RIGHT]+=1
120         elif value[RIGHT]<UMBRAL:
121                 if state[RIGHT]==1 and repeticion[RIGHT]>=UMBREP:
122                         tic[RIGHT]+=1
123                 repeticion[RIGHT]=0
124                 state[RIGHT]=0
125         return (tic,state,repeticion)
126
127 #Function block to accumulate the total distance of the system
128 def measureDistance():
129         tic=[0,0]
130         state=[1,1]
131         repeticion=[0,0]
132         global distance
133         global exitFlag
134         fo = open("Tics.txt", "wb")
135         ExitLock.acquire()
136         while not exitFlag:
137                 ExitLock.release()
138                 #Read Sensor
139                 value=[ADC.read(encPin[LEFT]),ADC.read(encPin[RIGHT])]
140                 lineValue='     '+str(value[LEFT])+'    '+str(value[
                        RIGHT])+'\n'
141                 fo.writelines(lineValue)
142                 [tic,state,repeticion]=countTics(tic,state,value,
                        repeticion)
143                 #Write Distance
144                 ADCLock.acquire()
145                 distance=[((float(tic[LEFT])/float(ticWheel))*
                        distPerRev),((float(tic[RIGHT])/float(ticWheel))*
                        distPerRev)]
146                 ADCLock.release()
147                 ExitLock.acquire()
148         ExitLock.release()
149         print "Exit Distance"
150         fo.close()
151         if DEBUG:
152                 print 'Total Length [',(float(tic[LEFT])/float(ticWheel
                        ))*distPerRev,',',(float(tic[RIGHT])/float(ticWheel)
                        )*distPerRev,']'
153
154 #Adjust the speed with the distances and a PD control
155 def adjustSpeed(limit,adjust):
156         global distance
157         global exitFlag
158         gap=round(distPerRev/ticWheel)
159         loop = 1
160         count=0
161         if limit<0:
162                 velocity=[-VELOCITY[LEFT],VELOCITY[RIGHT]]
```

```
163         else:
164                 velocity=[VELOCITY[LEFT],-VELOCITY[RIGHT]]
165         velL=velocity[LEFT]
166         velR=velocity[RIGHT]
167         while loop:
168                 move([velL,velR])
169                 ADCLock.acquire()
170                 localDistance=[distance[LEFT],distance[RIGHT]]
171                 ADCLock.release()
172                 if (localDistance[LEFT] > abs(limit) or localDistance[
                        RIGHT] > abs(limit)):
173                       cleanup()
174                       ExitLock.acquire()
175                       exitFlag=1
176                       ExitLock.release()
177                       break
178                 print 'LocalDistance:',localDistance[LEFT],
                        localDistance[RIGHT],'PWM:',velL,velR,adjust
179         print "Exit Speed"
180
181 def main():
182         # Program Ini Variables
183         threads = []
184
185         if(len(sys.argv)<2):
186                 print 'Error define parameter Angle[grad]'
187                 return 0
188         limitA=float(sys.argv[1])/2
189         distA=float(limitA/360)*distPerRound
190         print distA
191         #Create new Threads
192         thread1 = myThread(1, "Tics",'x','y')
193         thread2 = myThread(1, "Speed",distA,ADJUST)
194         setup()
195         # Start new Threads
196         thread1.start()
197         thread2.start()
198         # Add threads to thread list
199         threads.append(thread1)
200         threads.append(thread2)
201         #Waitting until end
202         for t in threads:
203                 t.join()
204         print "Exiting Main Thread"
205         return 0
206
207 if __name__ == '__main__':
208     main()
```

C.2. Codigo sistema Olus

C.2.1. multinose .cpp

```
1
2  #include "MultiNose.h"
3  #include "exceptions.h"
4
5  #include <boost/array.hpp>
6  #include <boost/exception/exception.hpp>
7  #include <boost/exception/errinfo_file_name.hpp>
8  #include <boost/exception/info.hpp>
9
10 #include <iomanip>
11 #include <sstream>
12 #include <string>
13
14 namespace b_asio = boost::asio;
15
16 using namespace boost;
17 using namespace std;
18
19 namespace libnose
20 {
21
22 MultiNose::MultiNose(const std::string path, unsigned int baud_rate) :
       Device(path, baud_rate)
23 {
24 }
25
26 MultiNose::~MultiNose() noexcept
27 {
28 }
29
30 void MultiNose::set_caudal(unsigned int direccion,unsigned int caudal)
31 {
32     char set_caudal_frame[] = ":006060000";
33
34     ostringstream new_caudal_stream;
35
36     new_caudal_stream << setfill('0') << uppercase << hex << setw(4) <<
           caudal;
37
38     ostringstream new_direccion;
39
40     new_direccion << setfill('0') << setw(2) << direccion;
41
42     const string &new_caudal_str = new_caudal_stream.str();
43
44     const string &new_direccion_str = new_direccion.str();
45
46     copy(new_caudal_str.begin(), new_caudal_str.end(), set_caudal_frame
           + 6);
47
48     copy(new_direccion_str.begin(), new_direccion_str.end(),
          set_caudal_frame + 1);
49
50     try
```

```
51      {
52          b_asio::write(_port, b_asio::buffer(set_caudal_frame, 10));
53
54          char receive_buffer[10];
55
56          b_asio::read(_port, b_asio::buffer(receive_buffer));
57      }
58          catch(boost::system::system_error &e)
59          {
60                  BOOST_THROW_EXCEPTION(operation_error() << what_string(
                        e.what()) << errinfo_file_name(_path));
61          }
62  }
63
64  void MultiNose::set_temcal(unsigned int direccion,unsigned int temcal)
65  {
66          char set_temcal_frame[] = ":006050000";  /*char
                set_temcal_frame[] = ":006020000";*/
67
68      ostringstream new_temcal_stream;
69
70      new_temcal_stream << setfill('0') << uppercase << hex << setw(4) <<
            temcal;
71
72      ostringstream new_direccion;
73
74      new_direccion << setfill('0') << setw(2) << direccion;
75
76      const string &new_temcal_str = new_temcal_stream.str();
77
78      const string &new_direccion_str = new_direccion.str();
79
80      copy(new_temcal_str.begin(), new_temcal_str.end(), set_temcal_frame
            + 6);
81
82      copy(new_direccion_str.begin(), new_direccion_str.end(),
           set_temcal_frame + 1);
83
84      try
85      {
86          b_asio::write(_port, b_asio::buffer(set_temcal_frame, 10));
87
88          boost::array<char, 10> receive_buffer;
89
90  //      char receive_buffer[10];
91
92          b_asio::read(_port, b_asio::buffer(receive_buffer));
93      }
94          catch(boost::system::system_error &e)
95          {
96                  BOOST_THROW_EXCEPTION(operation_error() << what_string(
                        e.what()) << errinfo_file_name(_path));
97          }
98  }
99
100 unsigned int MultiNose::get_caudal(unsigned int direccion)
101 {
102         char get_sensor_frame[] = ":003010000";
103
```

```
104         boost::array<char, 10> receive_buffer;
105
106         ostringstream new_direccion;
107
108         new_direccion << setfill('0') << setw(2) << direccion;
109
110         const string &new_direccion_str = new_direccion.str();
111
112         copy(new_direccion_str.begin(), new_direccion_str.end(),
                get_sensor_frame + 1);
113
114         try
115         {
116                 b_asio::write(_port, b_asio::buffer(get_sensor_frame,
                        10));
117
118                 b_asio::read(_port, b_asio::buffer(receive_buffer));
119         }
120         catch(boost::system::system_error &e)
121         {
122                 BOOST_THROW_EXCEPTION(operation_error() << what_string(
                        e.what()) << errinfo_file_name(_path));
123         }
124
125     string sensor_value(receive_buffer.begin() + 6, receive_buffer.end
            ());
126
127     istringstream sensor_stream(sensor_value);
128
129     unsigned int result = 0;
130
131     sensor_stream >> hex >> result;
132
133     return result;
134 }
135
136 unsigned int MultiNose::get_temcal(unsigned int direccion)
137 {
138         char get_sensor_frame[] = ":003050000";
139
140         boost::array<char, 10> receive_buffer;
141
142         ostringstream new_direccion;
143
144         new_direccion << setfill('0') << setw(2) << direccion;
145
146         const string &new_direccion_str = new_direccion.str();
147
148         copy(new_direccion_str.begin(), new_direccion_str.end(),
                get_sensor_frame + 1);
149
150         try
151         {
152                 b_asio::write(_port, b_asio::buffer(get_sensor_frame,
                        10));
153
154                 b_asio::read(_port, b_asio::buffer(receive_buffer));
155         }
156         catch(boost::system::system_error &e)
```

```
157         {
158                 BOOST_THROW_EXCEPTION(operation_error() << what_string(
                       e.what()) << errinfo_file_name(_path));
159         }
160     string sensor_value(receive_buffer.begin() + 6, receive_buffer.end
           ());
161
162     istringstream sensor_stream(sensor_value);
163
164     unsigned int result = 0;
165
166     sensor_stream >> hex >> result;
167
168     return result;
169 }
170
171 unsigned int MultiNose::get_sensor(unsigned int direccion,unsigned int
       sensor)
172 {
173         char get_sensor_frame[] = ":033000000"; /*char get_sensor_frame
               [] = ":003000000";*/
174
175     //The MultiNose requires adding 3 to the sensor address
176         get_sensor_frame[5] = '0' + sensor;
177
178         boost::array<char, 10> receive_buffer;
179
180         ostringstream new_direccion;
181
182         new_direccion << setfill('0') << setw(2) << direccion;
183
184         const string &new_direccion_str = new_direccion.str();
185
186         copy(new_direccion_str.begin(), new_direccion_str.end(),
               get_sensor_frame + 1);
187     try
188     {
189         b_asio::write(_port, b_asio::buffer(get_sensor_frame, 10));
190
191         b_asio::read(_port, b_asio::buffer(receive_buffer));
192     }
193     catch(boost::system::system_error &e)
194     {
195         BOOST_THROW_EXCEPTION(operation_error() << what_string(e.what()
               ) << errinfo_file_name(_path));
196     }
197     string sensor_value(receive_buffer.begin() + 6, receive_buffer.end
           ());
198
199     istringstream sensor_stream(sensor_value);
200
201     unsigned int result = 0;
202
203     sensor_stream >> hex >> result;
204
205     return result;
206 }
207 }
```

C.2.2. main .cpp

```
/*
    ********************************************************************************

 *      Sampling nose sensors for Olus system
 *      OUT: Files .txt with measures and averages
 *                      Olus3   Olus11  Olus9   Olus7   AverageFile
 *      @ Alejandro Pequeno
 *      @ GNB. Universidad Autonoma de Madrid
 *      Libraries @ Fernando Herrero
 *
 ********************************************************************************
    */

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstring>
#include <string>
#include <time.h>
#include <math.h>
#include <LibNose/MultiNose.h>

#define PI 3.14159265

#define DIRECTORIO      "muestras/Muestras_Calentamiento/"
#define PUERTO          "/dev/ttyUSB0"
#define BAUDIOS         115200

/*System variables*/
#define ITERACIONES      200 //Samples
#define PERIODO          1   //Period between samples 1/frequency
#define SLEEP            20  //Numero de pasos de estabilizacion

/*Temperature Variables*/
#define T1              340
#define T2              340
#define T3              340
#define T4              340

/*Temperature variables*/
#define NP              10 //Number of Points
#define VAR             10 //Maximun Variation

using namespace std;
using namespace libnose;


int main()
{

        int contador=1,concentration3=0,concentration11=0,
           concentration9=0,concentration7=0;

        double temperaturaExt3=0,temperaturaExt11=0,temperaturaExt9=0,
           temperaturaExt7=0;

```

```
52          int temperatura3=0,temperatura11=0,temperatura9=0,temperatura7
               =0,ciclos=0,estado=0;
53
54          int *pendiente3;
55
56          double ta1=0,ta2=0,ta3=0,ta4=0;
57
58          double av3=0,av9=0,av11=0,av7=0;
59
60          double xav3=0,xav9=0,xav11=0,xav7=0;
61
62          double tav3=0,tav9=0,tav11=0,tav7=0;
63
64          ciclos=floor(ITERACIONES/NP);
65
66          double av3T[ciclos],av9T[ciclos],av11T[ciclos],av7T[ciclos];
67
68          /*Cuantification variables for external temperature measure*/
69          double convert= 5000/1023;            //mv per point
70
71          double pasoT=floor(1023/ITERACIONES);
72
73          char timestamp[80];
74
75          ofstream myfile3,myfile11,myfile9,myfile7,myAveragefile;
76
77          stringstream name1,name2,name3,name4,name5;
78
79          string sName1,sName2,sName3,sName4,sName5;
80
81          //Timestamp variables for files
82          time_t rawtime;
83
84          struct tm * timeinfo;
85
86          rawtime =time(0);
87          timeinfo = localtime (&rawtime);
88          strftime (timestamp,80,"%d%b%y[%a]_%H:%M_",timeinfo);
89
90          name1 << DIRECTORIO <<"Ta" << T1 << "+-" << VAR << "_" <<
               timestamp << "Olus3.txt";
91          name2 << DIRECTORIO <<"Ta" << T2 << "+-" << VAR << "_" <<
               timestamp <<  "Olus11.txt";
92          name3 << DIRECTORIO <<"Ta" << T3 << "+-" << VAR << "_" <<
               timestamp <<  "Olus9.txt";
93          name4 << DIRECTORIO <<"Ta" << T4 << "+-" << VAR << "_" <<
               timestamp <<  "Olus7.txt";
94          name5 << DIRECTORIO << "Ta" << T1 << "+-" << VAR << "_" <<
               timestamp <<  "AverageFile.txt";
95
96          name1 >> sName1;
97          name2 >> sName2;
98          name3 >> sName3;
99          name4 >> sName4;
100         name5 >> sName5;
101
102         myfile3.open(sName1.c_str());
103         myfile11.open(sName2.c_str());
104         myfile9.open(sName3.c_str());
```

```
105         myfile7.open(sName4.c_str());
106         myAveragefile.open(sName5.c_str());
107 
108         MultiNose my_nose(PUERTO,BAUDIOS);
109 
110         //Format of Files
111         myfile3  << "Muestra (t) \t"  << "Concentracion\t" << "
               Temperatura Exterior: "<< ((my_nose.get_sensor(4,4)*convert)
               -500)/10 <<"\n";
112         myfile11 << "Muestra (t)\t"   << "Concentracion\t" << "
               Temperatura Exterior: "<< ((my_nose.get_sensor(12,4)*convert
               )-500)/10 <<"\n";
113         myfile9  << "Muestra (t) \t"  << "Concentracion\t" << "
               Temperatura Exterior: "<< ((my_nose.get_sensor(10,4)*convert
               )-500)/10 <<"\n";
114         myfile7  << "Muestra (t) \t"  << "Concentracion\t" << "
               Temperatura Exterior: "<< ((my_nose.get_sensor(8,4)*convert)
               -500)/10 <<"\n";
115         myAveragefile << "Ciclo  \t"  << "Average 3 \t" << "Average 11
               \t" << "Average 9 \t"<< "Average 7 \n";
116 
117         /*Firsts steps heated sensors */
118         for(int j=0; j<SLEEP;j++){
119                 ta1=T1+VAR*cos(2*PI*j/NP);
120                 ta2=T2+VAR*cos(2*PI*j/NP);
121                 ta3=T3+VAR*cos(2*PI*j/NP);
122                 ta4=T4+VAR*cos(2*PI*j/NP);
123                 my_nose.set_temcal(3,ta1);
124                 my_nose.set_temcal(11,ta2);
125                 my_nose.set_temcal(9,ta3);
126                 my_nose.set_temcal(7,ta4);
127                 sleep(PERIODO);
128         }
129 
130         while(contador <= ITERACIONES)
131         {
132                 /*Sensors Data*/
133                 concentration3=my_nose.get_sensor(3,1);
134                 concentration11=my_nose.get_sensor(11,1);
135                 concentration9=my_nose.get_sensor(9,1);
136                 concentration7=my_nose.get_sensor(7,1);
137 
138                 temperatura3=my_nose.get_temcal(3);
139                 temperatura11=my_nose.get_temcal(11);
140                 temperatura9=my_nose.get_temcal(9);
141                 temperatura7=my_nose.get_temcal(7);
142 
143                 temperaturaExt3=((my_nose.get_sensor(4,4)*convert)-500)
                    /10;
144                 temperaturaExt11=((my_nose.get_sensor(12,4)*convert)
                    -500)/10;
145                 temperaturaExt9=((my_nose.get_sensor(10,4)*convert)
                    -500)/10;
146                 temperaturaExt7=((my_nose.get_sensor(8,4)*convert)-500)
                    /10;
147 
148                 /*Debug through terminal*/
149                 cout << "Muestra (t) = " << contador*PERIODO<<"\n";
```

```
150                     cout << "Nariz 1\t\t\tNariz 2\t\t\tNariz 3\t\t\tNariz
                           4\n";
151                     cout << concentration3 << ' ' << temperatura3 << ' ' <<
                            temperaturaExt3 <<"\t\t";
152                     cout << concentration11 << ' ' << temperatura11 << ' '
                           << temperaturaExt11 <<"\t\t";
153                     cout << concentration9 << ' ' << temperatura9 << ' ' <<
                            temperaturaExt9 <<"\t\t";
154                     cout << concentration7 << ' ' << temperatura7 << ' ' <<
                            temperaturaExt7 << "\n";
155
156                     /*Write individual File */
157                     myfile3 << contador << "\t\t"  << concentration3 << "\t
                           \t" << temperatura3 << "\t\t" << temperaturaExt3 <<"
                           \n" ;
158                     myfile11 << contador << "\t\t"<< concentration11  << "\
                           t\t"<< temperatura11 << "\t\t" << temperaturaExt11
                           <<"\n" ;
159                     myfile9 << contador << "\t\t" << concentration9 << "\t\
                           t" << temperatura9 << "\t\t" << temperaturaExt9 << "
                           \n" ;
160                     myfile7 << contador << "\t\t" << concentration7 << "\t\
                           t" << temperatura7 << "\t\t" << temperaturaExt7 << "
                           \n" ;
161
162                     contador++;
163
164                     /*Update Temperatures*/
165                     ta1=T1+ceil(VAR*cos(2*PI*(contador-1)/NP));
166                     ta2=T2+ceil(VAR*cos(2*PI*(contador-1)/NP));
167                     ta3=T3+ceil(VAR*cos(2*PI*(contador-1)/NP));
168                     ta4=T4+ceil(VAR*cos(2*PI*(contador-1)/NP));
169                     my_nose.set_temcal(3,ta1);
170                     my_nose.set_temcal(11,ta2);
171                     my_nose.set_temcal(9,ta3);
172                     my_nose.set_temcal(7,ta4);
173
174                     /*Average*/
175                     av3+=concentration3;
176                     av11+=concentration11;
177                     av9+=concentration9;
178                     av7+=concentration7;
179
180                     tav3+=temperaturaExt3;
181                     tav11+=temperaturaExt11;
182                     tav9+=temperaturaExt9;
183                     tav7+=temperaturaExt7;
184
185                     /*Average for cycle*/
186                     xav3+=(double)concentration3/NP;
187                     xav9+=(double)concentration9/NP;
188                     xav11+=(double)concentration11/NP;
189                     xav7+=(double)concentration7/NP;
190
191                     if(contador%NP==0&&contador!=0){
192                             av3T[estado]=xav3;
193                             xav3=0;
194                             av11T[estado]=xav11;
195                             xav11=0;
```

```
196                             av9T[estado]=xav9;
197                             xav9=0;
198                             av7T[estado]=xav7;
199                             xav7=0;
200                             estado++;
201                     }
202                     sleep(PERIODO);
203             }
204
205             myfile3.close();
206             myfile11.close();
207             myfile9.close();
208             myfile7.close();
209             /*Write Average File*/
210             for(int i=0; i<ciclos;i++){
211                     printf ("Ciclo%d    Average3:%f  Average11:%f  Average9
                            :%f  Average7:%f\n",i,av3T[i],av11T[i],av9T[i],av7T[
                            i]);
212                     myAveragefile << i << "\t\t" << av3T[i] << "\t\t" <<
                            av11T[i] << "\t\t" << av9T[i]<<"\t\t" << av7T[i]<<"\
                            t\t" << T1<< "\n";
213             }
214             myAveragefile << tav3/ITERACIONES << "\t\t" << tav11/
                    ITERACIONES << "\t\t" << tav9/ITERACIONES<<"\t\t" << tav7/
                    ITERACIONES<< "\n";
215             myAveragefile.close();
216             /*Debug in terminal*/
217             printf ("\nTotal_Average3:%f\nAverage11:%f\nAverage9:%f\
                    nAverage7:%f\n",av3/ITERACIONES,av11/ITERACIONES,av9/
                    ITERACIONES,av7/ITERACIONES);
218             printf ("\nTemperatura_Average3:%f\nAverage11:%f\nAverage9:%f\
                    nAverage7:%f\n",tav3/ITERACIONES,tav11/ITERACIONES,tav9/
                    ITERACIONES,tav7/ITERACIONES);
219
220             return 0;
221     }
```

C.2.3. main diferencial basico ventana solapante .cpp

```
/*
    ********************************************************************************

 *      Sampling nose sensors for Olus system .
 *              -Calculate Average in sliding window
 *              -Take Decission base on differential averages
 *      OUT: Files .txt with measures and averages
 *                      Olus3   Olus11  Olus9   Olus7   AverageFile
 *      @ Alejandro Pequeno
 *      @ GNB. Universidad Autonoma de Madrid
 *      Libraries @ Fernando Herrero
 *
 ********************************************************************************
    */

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstring>
#include <string>
#include <time.h>
#include <math.h>
#include <LibNose/MultiNose.h>

#define PI 3.14159265

#define DIRECTORIO      "muestras/Muestras_Test/"

#define PUERTO          "/dev/ttyUSB0"
#define BAUDIOS         115200

/*System variables*/
#define ITERACIONES      200 //Samples
#define PERIODO          1   //Period between samples 1/frequency
#define SLEEP            20  //Numero de pasos de estabilizacion

/*Temperature Variables*/
#define T1              307
#define T2              307
#define T3              307
#define T4              307

/*Variacion de Tł*/
#define NP              12
#define AV              30
#define LIMITSOURCE     550
#define VAR             10 // Maximun variation

using namespace std;
using namespace libnose;


int main(int argc, char *argv[])
{

```

```
53          int contador=0,concentration3=0,concentration11=0,
               concentration9=0,concentration7=0;
54
55          double temperaturaExt3=0,temperaturaExt11=0,temperaturaExt9=0,
               temperaturaExt7=0;
56
57          int temperatura3=0,temperatura11=0,temperatura9=0,temperatura7
               =0;
58
59          double ta1=0,ta2=0,ta3=0,ta4=0;
60
61          int index;
62
63          double av3=0,av9=0,av11=0,av7=0;
64
65          double xav3=0,xav9=0,xav11=0,xav7=0;
66
67          double av3T[NP],av9T[NP],av11T[NP],av7T[NP];
68
69          double  offsetTan, offsetPar;
70
71          int decisionIndex=ITERACIONES-AV;
72
73          double decision3=0, decision11=0,decision9=0, decision7=0;
74
75          double decisionAv3=0, decisionAv11=0,decisionAv9=0, decisionAv7
               =0;
76
77          double decisionAvTan=0, decisionAvPar=0;
78
79          /*Cuantification variables for external temperature measure*/
80          double convert= 5000/1023; //mv per point
81
82          double pasoT=floor(1023/ITERACIONES);
83
84          char timestamp[80];
85
86          ofstream myfile3,myfile11,myfile9,myfile7,myAveragefile,
               myOffsetfile,myDecision;
87
88          stringstream name1,name2,name3,name4,name5,name6,name8;
89
90          string sName1,sName2,sName3,sName4,sName5,sName6,sName8;
91
92          //Timestamp variables
93          time_t rawtime;
94
95          struct tm * timeinfo;
96
97          rawtime =time(0);
98          timeinfo = localtime (&rawtime);
99          strftime (timestamp,80,"%d%b%y[%a]_%H:%M_",timeinfo);
100
101         //Control of Arguments
102         if (argc != 3){
103                 cout << "Uso: ./test_nariz_diferencial offsetParalelo
                           offsetTangente \n";
104                 return 0;
105         }
```

```
106         else{
107                 offsetPar = std::stol(argv[1],nullptr,0);
108                 offsetTan = std::stol(argv[2],nullptr,0);
109         }
110
111         name1 << DIRECTORIO <<"Ta" << T1 << "+-" << VAR << "_" <<
               timestamp << "Olus3.txt";
112         name2 << DIRECTORIO << "Ta" << T2 << "+-" << VAR << "_" <<
               timestamp <<  "Olus11.txt";
113         name3 << DIRECTORIO <<"Ta" << T3 << "+-" << VAR << "_" <<
               timestamp <<  "Olus9.txt";
114         name4 << DIRECTORIO <<"Ta" << T4 << "+-" << VAR << "_" <<
               timestamp <<  "Olus7.txt";
115         name5 << DIRECTORIO <<"Average.txt";
116         name6 << DIRECTORIO << "offset.txt";
117         name8 << "decission.txt";
118
119         name1 >> sName1;
120         name2 >> sName2;
121         name3 >> sName3;
122         name4 >> sName4;
123         name5 >> sName5;
124         name6 >> sName6;
125         name8 >> sName8;
126
127         myfile3.open(sName1.c_str());
128         myfile11.open(sName2.c_str());
129         myfile9.open(sName3.c_str());
130         myfile7.open(sName4.c_str());
131         myAveragefile.open(sName5.c_str());
132
133         MultiNose my_nose(PUERTO,BAUDIOS);
134
135         /*Parameters*/
136         myfile3  << "Muestra (t) \t"  << "Concentracion\t" << "
               Temperatura Exterior: "<< ((my_nose.get_sensor(4,4)*convert)
               -500)/10 <<"\n";
137         myfile11 << "Muestra (t)\t"   << "Concentracion\t" << "
               Temperatura Exterior: "<< ((my_nose.get_sensor(12,4)*convert
               )-500)/10 <<"\n";
138         myfile9  << "Muestra (t) \t"  << "Concentracion\t" << "
               Temperatura Exterior: "<< ((my_nose.get_sensor(10,4)*convert
               )-500)/10 <<"\n";
139         myfile7  << "Muestra (t) \t"  << "Concentracion\t" << "
               Temperatura Exterior: "<< ((my_nose.get_sensor(8,4)*convert)
               -500)/10 <<"\n";
140         myAveragefile << "Ciclo  \t"  << "Average 3 \t" << "Average 11
               \t" << "Average 9 \t"<< "Average 7 \n";
141
142         my_nose.set_temcal(3,T1);
143         my_nose.set_temcal(11,T2);
144         my_nose.set_temcal(9,T3);
145         my_nose.set_temcal(7,T4);
146         sleep(PERIODO);
147
148         /*Firsts steps heated sensors */
149         for(int i=0; i<NP;i++){
150                 av3T[i]=0;
151                 av11T[i]=0;
```

```
152                     av9T[i]=0;
153                     av7T[i]=0;
154             }
155
156             while(contador < ITERACIONES)
157             {
158                     /*Sensors Data*/
159                     concentration3=my_nose.get_sensor(3,1);
160                     concentration11=my_nose.get_sensor(11,1);
161                     concentration9=my_nose.get_sensor(9,1);
162                     concentration7=my_nose.get_sensor(7,1);
163
164                     temperatura3=my_nose.get_temcal(3);
165                     temperatura11=my_nose.get_temcal(11);
166                     temperatura9=my_nose.get_temcal(9);
167                     temperatura7=my_nose.get_temcal(7);
168
169                     temperaturaExt3=((my_nose.get_sensor(4,4)*convert)-500)
                           /10;
170                     temperaturaExt11=((my_nose.get_sensor(12,4)*convert)
                           -500)/10;
171                     temperaturaExt9=((my_nose.get_sensor(10,4)*convert)
                           -500)/10;
172                     temperaturaExt7=((my_nose.get_sensor(8,4)*convert)-500)
                           /10;
173
174                     /*Debug through terminal*/
175                     cout << "Muestra (t) = " << contador*PERIODO<<"\n";
176                     cout << "Nariz 1\t\t\tNariz 2\t\t\tNariz 3\t\t\tNariz
                           4\n";
177                     cout << concentration3 << ' ' << temperatura3 << ' ' <<
                            temperaturaExt3 <<"\t\t";
178                     cout << concentration11 << ' ' << temperatura11 << ' '
                           << temperaturaExt11 <<"\t\t";
179                     cout << concentration9 << ' ' << temperatura9 << ' ' <<
                            temperaturaExt9 <<"\t\t";
180                     cout << concentration7 << ' ' << temperatura7 << ' ' <<
                            temperaturaExt7 << "\n";
181
182                     /*Write individual File */
183                     myfile3 << contador << "\t\t"  << concentration3 << "\t
                           \t" << temperatura3 << "\t\t" << temperaturaExt3 <<"
                           \n" ;
184                     myfile11 << contador << "\t\t"<< concentration11  << "\
                           t\t"<< temperatura11 << "\t\t" << temperaturaExt11
                           <<"\n" ;
185                     myfile9 << contador << "\t\t" << concentration9 << "\t\
                           t" << temperatura9 << "\t\t" << temperaturaExt9 << "
                           \n" ;
186                     myfile7 << contador << "\t\t" << concentration7 << "\t\
                           t" << temperatura7 << "\t\t" << temperaturaExt7 << "
                           \n" ;
187
188
189                     /*Average calculate in sliding window*/
190                     index=contador%NP;
191                     cout << index << "\n";
192
193                     av3T[index]=concentration3;
```

```
194                     av9T[index]=concentration9;
195                     av11T[index]=concentration11;
196                     av7T[index]=concentration7;
197
198                     av3=0;
199                     av11=0;
200                     av9=0;
201                     av7=0;
202                     for(int j=0; j<NP;j++){
203                             av3+=av3T[j];
204                             av11+=av11T[j];
205                             av9+=av9T[j];
206                             av7+=av7T[j];
207                     }
208
209                     if (contador<NP){
210                             xav3=(double)av3/(contador+1);
211                             xav11=(double)av11/(contador+1);
212                             xav9=(double)av9/(contador+1);
213                             xav7=(double)av7/(contador+1);
214                     }
215                     else{
216                             xav3=(double)av3/NP;
217                             xav11=(double)av11/NP;
218                             xav9=(double)av9/NP;
219                             xav7=(double)av7/NP;
220                     }
221
222                     /*Offset representation*/
223                     if(contador==NP){
224                                     myOffsetfile.open(sName6.c_str());
225                                     myOffsetfile << offsetPar << "\t" <<
                                           offsetTan << "\n";
226                                     myOffsetfile.close();
227                     }
228                     printf ("Ciclo%d    Average3:%f  Average11:%f  Average9
                           :%f  Average7:%f\n",contador,xav3,xav11,xav9,xav7);
229                     myAveragefile << contador << "\t\t" << xav3 << "\t\t"
                           << xav11 << "\t\t" << xav9 <<"\t\t" << xav7<<"\t\t"
                           << T1 <<"\t\t"<< T2 <<"\t\t"<< T3<<"\t\t" << T4 << "
                           \n";
230
231                     /*Decission base on average */
232                     if(contador >= decisionIndex){
233                             decision3+=concentration3;
234                             decision9+=concentration9;
235                             decision11+=concentration11;
236                             decision7+=concentration7;
237                     }
238
239                     ta1=T1+ceil(VAR*sin(2*PI*(contador-1)/NP));
240                     ta2=T2+ceil(VAR*sin(2*PI*(contador-1)/NP));
241                     ta3=T3+ceil(VAR*sin(2*PI*(contador-1)/NP));
242                     ta4=T4+ceil(VAR*sin(2*PI*(contador-1)/NP));
243
244                     /*Update Temperature*/
245                     my_nose.set_temcal(3,ta1);
246                     my_nose.set_temcal(11,ta2);
247                     my_nose.set_temcal(9,ta3);
```

```
248                 my_nose.set_temcal(7,ta4);
249
250                 contador++;
251
252                 sleep(PERIODO);
253         }
254         /*Locate Source Condition*/
255         decisionAv3=decision3/AV;
256         decisionAv9=decision9/AV;
257         decisionAv11=decision11/AV;
258         decisionAv7=decision7/AV;
259
260         decisionAvPar+=decisionAv3-decisionAv9+offsetPar;
261         decisionAvTan+=decisionAv11-decisionAv7+offsetTan;
262         myDecision.open(sName8.c_str());
263         if(decisionAv3>LIMITSOURCE && decisionAv9>LIMITSOURCE){
264                 if(round(decisionAvPar)>=0)
265                                 myDecision << "XF" <<"\n";      //
                                    Source in Forward Location
266                 else if(round(decisionAvPar)<0)
267                                 myDecision << "XB" <<"\n";      //
                                    Source in Backward Location
268         }
269         else{
270                 if(round(decisionAvPar)>0)
271                                 myDecision << 'F' <<"\n";       //
                                    Forward
272                 else if(round(decisionAvPar)<0)
273                                 myDecision << 'B' <<"\n";       //
                                    Backward
274                 else
275                                 myDecision << 'U' <<"\n";       //
                                    Unknow
276         }
277         myDecision.close();
278         myfile3.close();
279         myfile11.close();
280         myfile9.close();
281         myfile7.close();
282         myAveragefile.close();
283         return 0;
284 }
```

C.2.4. algoritmo diferencial C .cpp

```
/*
    ********************************************************************************

 *      Sampling nose sensors for Olus system
 *              and take decissions for Robotic Platform
 *              -Take Decissions base on differential averages
 *      IN1: offsetParallel-> difference in a steady state  in order to
     compair sensors
 *      IN1: offsetPerpendicular-> difference in a steady state  in
    order to compair sensors
 *      IN2: input.txt file which contains the temperature of the
    sensors
 *      OUT1: Files .txt with measures and averages
 *              Olus3   Olus11  Olus9   Olus7   AverageFile
 *      OUT2: Files .txt which feed the decission script
 *              offset: differential between sensor in an axis in the
    steady state
 *                      file for figure representation
 *              decission: move direction
 *              input: final temperature and start point for the next
    sampling
 *      @ Alejandro Pequeno
 *      @ GNB. Universidad Autonoma de Madrid
 *      Libraries @ Fernando Herrero
 *
 ********************************************************************************
    */

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstring>
#include <string>
#include <time.h>
#include <math.h>
#include <LibNose/MultiNose.h>

#define PI 3.14159265

/*System variables*/
#define DIRECTORIO      "muestras/Muestras_10_24_Robot_Basico/" //Local
    Directory
#define NAME            "/home/lala/TestNariz/src/muestras/
   Muestras_10_24_Robot_Basico/input.txt" //Directory for input file
#define PUERTO          "/dev/ttyUSB0"
#define BAUDIOS         115200

#define ITERACIONES      100 //Samples
#define PERIODO          1  //Period between samples 1/frequency
#define SLEEP            20 //Numero de pasos iniciales no registrados

#define LIMITSOURCE     600 //Umbral limite que delimita la posicion de
    la fuente

/*Temperature Variables*/
#define T1              340
```

```
46  #define T2              340
47  #define T3              340
48  #define T4              340
49
50  /*Adjust Temperature variables*/
51  #define NP              10  // Steps of sine function
52  #define VAR             10  // Max variance
53  #define AV              30  // Final Samples taken in decission
54
55  /*PID parameters for temperature control*/
56  #define Kp              1.8     //Derivative parameter
57  #define Ki              0.54    //Integrative parameter
58
59  /*Algorithm parameters*/
60  #define PASO            5       // Start Gap
61  #define TMAX            400     // Maximun Temperature
62  #define TMIN            200     // Minimun Temperature
63
64  using namespace std;
65  using namespace libnose;
66
67
68  int main(int argc, char *argv[])
69  {
70          int contador=1;
71
72          int concentration3=0,concentration11=0,concentration9=0,
                concentration7=0;
73
74          int temperatura3=0,temperatura11=0,temperatura9=0,temperatura7
                =0;
75
76          int ciclos=floor(ITERACIONES/NP),estado=0;
77
78          double ta1=0,ta2=0,ta3=0,ta4=0;
79
80          double ti1=T1,ti2=T2,ti3=T3,ti4=T4;
81
82          int saltoTan=PASO, saltoPar=PASO, margenPar, margenTan;
83
84          double av3=0,av9=0,av11=0,av7=0;
85
86          double xav3=0,xav9=0,xav11=0,xav7=0;
87
88          double av3T[ciclos],av9T[ciclos],av11T[ciclos],av7T[ciclos];
89
90          double diferenciaM,diferenciaA,diferenciaV,diferenciaR,
                diferenciaTm=0, diferenciaTa=0,diferenciaTv=0, diferenciaTr
                =0, offsetTan, offsetPar;
91
92          double sumDecision;
93
94          double avgSource3, avgSource9;
95
96          double decisionIndex=ITERACIONES-AV;
97
98          int decisionAv=0;
99
100         double convert= 5000/1023; //mv per point converter
```

```
101
102         /*Timestamp of files*/
103         char timestamp[80];
104
105         ofstream myfile3,myfile11,myfile9,myfile7,myAveragefile,
               myOffsetfile,myInputfile,myDecision;
106
107         stringstream name1,name2,name3,name4,name5,name6,name7,name8;
108
109         string sName1,sName2,sName3,sName4,sName5,sName6,sName7,sName8;
110
111         time_t rawtime; //Timestamp variables
112
113         struct tm * timeinfo;
114
115         rawtime =time(0);
116         timeinfo = localtime (&rawtime);
117         strftime (timestamp,80,"%d%b%y[%a]_%H:%M_",timeinfo);
118
119         //Read Input File
120         char line[1000], *token;
121         FILE *f=NULL;
122
123         //Arguments control
124         if (argc != 3){
125                 cout << "Uso: ./test_nariz_diferencial offsetParalelo
                           offsetTangente \n";
126                 return 0;
127         }
128         else{
129                 offsetPar = std::stol(argv[1],nullptr,0);
130                 offsetTan = std::stol(argv[2],nullptr,0);
131         }
132
133         if ((f = fopen (NAME, "r")) == NULL)
134         {
135                 printf("\nCannot open the file:%s\n Set up default
                           values\n", NAME);
136         }
137         else
138         {
139                 fgets(line,1000,f);
140                 token = strtok(line, "\t");
141                 printf("\nToken Paralel, Input Olus 3&9:%s\n", token);
142                 if(token!=NULL){
143                         ti1=std::stoi(token,nullptr,0);
144
145                         ti3=std::stoi(token,nullptr,0);
146
147                 }
148                 token = strtok(NULL, "\n");
149                 printf("\nToken Tangent, Input Olus 11&7:%s\n", token);
150                 if(token!=NULL){
151
152                         ti2=std::stoi(token,nullptr,0);
153
154                         ti4=std::stoi(token,nullptr,0);
155                 }
156                 fclose(f);
```

```
157             }
158             /*Asignacion de nombre de ficheros*/
159             name1 << DIRECTORIO <<"Olus3.txt";
160             name2 << DIRECTORIO <<"Olus11.txt";
161             name3 << DIRECTORIO <<"Olus9.txt";
162             name4 << DIRECTORIO <<"Olus7.txt";
163             name5 << DIRECTORIO << "Average.txt";
164             name6 << DIRECTORIO << "offset.txt";
165             name7 << DIRECTORIO << "input.txt";
166             name8 << "decission.txt";
167
168             name1 >> sName1;
169             name2 >> sName2;
170             name3 >> sName3;
171             name4 >> sName4;
172             name5 >> sName5;
173             name6 >> sName6;
174             name7 >> sName7;
175             name8 >> sName8;
176
177             myfile3.open(sName1.c_str());
178             myfile11.open(sName2.c_str());
179             myfile9.open(sName3.c_str());
180             myfile7.open(sName4.c_str());
181             myAveragefile.open(sName5.c_str());
182
183             MultiNose my_nose(PUERTO,BAUDIOS);
184
185             /*Start params*/
186             myfile3  << "Muestra (t) \t"  << "Concentracion\t" << "
                    Temperatura: Ext["<< ((my_nose.get_sensor(4,4)*convert)-500)
                    /10 <<"]\n";
187             myfile11 << "Muestra (t)\t"   << "Concentracion\t" << "
                    Temperatura: Ext["<< ((my_nose.get_sensor(12,4)*convert)
                    -500)/10 <<"]\n";
188             myfile9  << "Muestra (t) \t"  << "Concentracion\t" << "
                    Temperatura: Ext["<< ((my_nose.get_sensor(10,4)*convert)
                    -500)/10 <<"]\n";
189             myfile7  << "Muestra (t) \t"  << "Concentracion\t" << "
                    Temperatura: Ext["<< ((my_nose.get_sensor(8,4)*convert)-500)
                    /10 <<"]\n";
190             myAveragefile << "Ciclo  \t"  << "Average 3 \t" << "Average 11
                    \t" << "Average 9 \t"<< "Average 7 \n";
191             /*Temperature*/
192             ta1=ti1+ceil(VAR*cos(2*PI*(contador)/NP));
193             ta2=ti2+ceil(VAR*cos(2*PI*(contador)/NP));
194             ta3=ti3+ceil(VAR*cos(2*PI*(contador)/NP));
195             ta4=ti4+ceil(VAR*cos(2*PI*(contador)/NP));
196             my_nose.set_temcal(3,ta1);
197             my_nose.set_temcal(11,ta2);
198             my_nose.set_temcal(9,ta3);
199             my_nose.set_temcal(7,ta4);
200             sleep(PERIODO);
201
202             /*Loop*/
203             while(contador <= ITERACIONES)
204             {
205                     concentration3=my_nose.get_sensor(3,1);
206                     concentration11=my_nose.get_sensor(11,1);
```

```
207                     concentration9=my_nose.get_sensor(9,1);
208                     concentration7=my_nose.get_sensor(7,1);
209
210                     temperatura3=my_nose.get_temcal(3);
211                     temperatura11=my_nose.get_temcal(11);
212                     temperatura9=my_nose.get_temcal(9);
213                     temperatura7=my_nose.get_temcal(7);
214
215                     /*Terminal Debug*/
216                     cout << "Muestra (t) = " << contador*PERIODO<<"\n";
217                     cout << "Nariz 1\t\tNariz 2\t\tNariz 3\t\tNariz 4\n";
218                     cout << concentration3 << ' ' << temperatura3  <<"\t\t"
                           ;
219                     cout << concentration11 << ' ' << temperatura11 <<"\t\t
                           ";
220                     cout << concentration9 << ' ' << temperatura9 <<"\t\t";
221                     cout << concentration7 << ' ' << temperatura7 << "\n";
222
223                     /*Sampling Write File*/
224                     myfile3 << contador << "\t\t"  << concentration3 << "\t
                           \t" << temperatura3 << "\n" ;
225                     myfile11 << contador << "\t\t"<< concentration11  << "\
                           t\t"<< temperatura11 <<"\n" ;
226                     myfile9 << contador << "\t\t" << concentration9 << "\t\
                           t" << temperatura9 << "\n" ;
227                     myfile7 << contador << "\t\t" << concentration7 << "\t\
                           t" << temperatura7 << "\n" ;
228
229                     /*Toral Average*/
230                     av3+=concentration3;
231                     av11+=concentration11;
232                     av9+=concentration9;
233                     av7+=concentration7;
234
235                     /*Average per cycle*/
236                     xav3+=(double)concentration3/NP;
237                     xav9+=(double)concentration9/NP;
238                     xav11+=(double)concentration11/NP;
239                     xav7+=(double)concentration7/NP;
240
241                     if(contador%NP==0&&contador!=0){
242                             av3T[estado]=xav3;
243                             av11T[estado]=xav11;
244                             av9T[estado]=xav9;
245                             av7T[estado]=xav7;
246
247                             /*Offset File*/
248                             if(estado==1){
249                                     //Hypothesis per axis
250                                     if(av3T[estado]-((av9T[estado])+
                                           offsetPar)<0)
251                                             margenPar=0;
252                                     else
253                                             margenPar=1;
254
255                                     if(av11T[estado]-((av7T[estado])+
                                           offsetPar)<0)
256                                             margenTan=0;
257                                     else
```

```
258                                     margenTan=1;
259
260                                 /*Offset file for plotting purpose*/
261                                 myOffsetfile.open(sName6.c_str());
262                                 myOffsetfile << offsetPar << "\t" <<
                                        offsetTan << "\n";
263                                 myOffsetfile.close();
264                             }
265                             /*Change Hypothesis, leader in direction of
                                    temperature change*/
266                             if(estado>1){
267                                 /*Value function of past differences*/
268                                 //Parallel
269                                 if((av3T[estado]-((av9T[estado])+
                                        offsetPar)>0) and (av3T[estado-1]-((
                                        av9T[estado-1])+offsetPar)>0))
270                                      margenPar=1;
271                                 else if((av3T[estado]-((av9T[estado])+
                                        offsetPar)<0) and (av3T[estado-1]-((
                                        av9T[estado-1])+offsetPar)<0))
272                                      margenPar=0;
273                                 //Perpendicular
274                                 if((av11T[estado]-((av7T[estado])+
                                        offsetTan)>0) and (av11T[estado
                                        -1]-((av7T[estado-1])+offsetTan)>0))
275                                      margenTan=1;
276                                 else if((av11T[estado]-((av7T[estado])+
                                        offsetTan)<0) and (av11T[estado
                                        -1]-((av7T[estado-1])+offsetTan)<0))
277                                      margenTan=0;
278
279                                 //Diferential values
280                                 diferenciaM=(av3T[estado]-(av3T[estado
                                        -1]))-(av9T[estado]-av9T[estado-1]);
281                                 diferenciaA=(av9T[estado]-av9T[estado
                                        -1])-(av3T[estado]-av3T[estado-1]);
282                                 diferenciaTm+=diferenciaM;
283                                 diferenciaTa+=diferenciaA;
284
285                                 diferenciaV=(av11T[estado]-(av11T[
                                        estado-1]))-(av7T[estado]-av7T[
                                        estado-1]);
286                                 diferenciaR=(av7T[estado]-av7T[estado
                                        -1])-(av11T[estado]-av11T[estado-1])
                                        ;
287                                 diferenciaTv+=diferenciaV;
288                                 diferenciaTr+=diferenciaR;
289
290                                 //Calculate Sign of the increment
291                                 //Parallel
292                                 if(margenPar==1){
293                                      saltoPar=abs(round(diferenciaM*
                                             Kp-diferenciaTm*Ki));
294                                      if(diferenciaM>0)
295                                              ;
296                                      else if(diferenciaM<0)
297                                              saltoPar=(-1)*saltoPar;
298                                 }
299                                 else if(margenPar==0){
```

```
300                                         saltoPar=abs(round(diferenciaA*
                                               Kp-diferenciaTa*Ki));
301                                         if(diferenciaA>0)
302                                                 ;
303                                         else if(diferenciaA<0)
304 \textsf{                                                       saltoPar=(-1)*
       saltoPar;
305
306                                 }
307                                 //Perpendicular
308                                 if(margenTan==1){
309                                         saltoTan=abs(round(diferenciaV*
                                               Kp-diferenciaTv*Ki));
310                                         if(diferenciaV>0)
311                                                 ;
312                                         else if(diferenciaV<0)
313                                                 saltoTan=(-1)*saltoTan;
314                                 }
315                                 else if(margenTan==0){
316                                         saltoTan=abs(round(diferenciaR*
                                               Kp-diferenciaTr*Ki));
317                                         if(diferenciaR>0)
318                                                 ;
319                                         else if(diferenciaR<0)
320                                                 saltoTan=(-1)*saltoTan;
321
322                                 }
323
324                                 /*Update Temperature*/
325                                 ti1+=saltoPar;
326                                 ti2+=saltoTan;
327                                 ti3+=saltoPar;
328                                 ti4+=saltoTan;
329
330                                 /*Temperature between limits*/
331                                 if(ti1>TMAX)
332                                         ti1=TMAX;
333                                 else if(ti1<TMIN)
334                                         ti1=TMIN;
335                                 if(ti2>TMAX)
336                                         ti2=TMAX;
337                                 else if(ti2<TMIN)
338                                         ti2=TMIN;
339                                 if(ti3>TMAX)
340                                         ti3=TMAX;
341                                 else if(ti3<TMIN)
342                                         ti3=TMIN;
343                                 if(ti4>TMAX)
344                                         ti4=TMAX;
345                                 else if(ti4<TMIN)
346                                         ti4=TMIN;
347                         }
348                         /*Write File*/
349                         myAveragefile << estado << "\t\t" << xav3 << "\
                               t\t" << xav11 << "\t\t" << xav9<<"\t\t" <<
                               xav7<<"\t\t" << ti1<<"\t\t" << ti2<<"\t\t"
                               << ti3<<"\t\t" << ti4<< "\n";
350
```

```
351                         cout << "MediaT(t) =" << xav3<<"\t\t"<<xav11<<"
                               \t\t"<<xav9<<"\t\t"<<xav7<<"\n";
352                         xav3=0;
353                         xav11=0;
354                         xav9=0;
355                         xav7=0;
356                         estado++;
357                         cout << "TMedia (t) =" << ti1<<"\t\t"<<ti2<<"\t
                               \t"<<ti3<<"\n";
358                 }
359                 /*Update Sine Temperature Cycle*/
360                 ta1=ti1+ceil(VAR*cos(2*PI*(contador+1)/NP));
361                 ta2=ti2+ceil(VAR*cos(2*PI*(contador+1)/NP));
362                 ta3=ti3+ceil(VAR*cos(2*PI*(contador+1)/NP));
363                 ta4=ti4+ceil(VAR*cos(2*PI*(contador+1)/NP));
364                 my_nose.set_temcal(3,ta1);
365                 my_nose.set_temcal(11,ta2);
366                 my_nose.set_temcal(9,ta3);
367                 my_nose.set_temcal(7,ta4);
368
369                 contador++;
370                 sleep(PERIODO);
371         }
372         /*Decission base on last average differences*/
373         sumDecision=(av3T[estado-1]-((av9T[estado-1])+offsetPar))+(av3T
               [estado-2]-((av9T[estado-2])+offsetPar))+(av3T[estado-3]-((
               av9T[estado-3])+offsetPar));
374         avgSource3=(av3T[estado-1]+av3T[estado-2]+av3T[estado-3])/3;
375         avgSource9=(av9T[estado-1]+av9T[estado-2]+av9T[estado-3])/3;
376
377         /*Source Location Condition*/
378         myDecision.open(sName8.c_str());
379         if(avgSource3>LIMITSOURCE && avgSource9>LIMITSOURCE){
380                 if(round(sumDecision)>=0)
381                                 myDecision << "XF" <<"\n";      //
                                       Source locate in Forward direction
382                 else if(round(sumDecision)<0)
383                                 myDecision << "XB" <<"\n";      //
                                       Source locate in Forward direction
384         }/*MOVE*/
385         else{
386                 if(round(sumDecision)>0)
387                                 myDecision << 'F' <<"\n";       //
                                       Forward
388                 else if(round(sumDecision)<0)
389                                 myDecision << 'B' <<"\n";       //
                                       Backward
390                 else
391                                 myDecision << 'U' <<"\n";       //
                                       Unknow
392         }
393         myDecision.close();
394         myfile3.close();
395         myfile11.close();
396         myfile9.close();
397         myfile7.close();
398         myAveragefile.close();
399
400         /*Write Input File*/
```

```
401             myInputfile.open(sName7.c_str());
402             myInputfile << ti1 <<"\t"<< ti2 <<"\n";
403             myInputfile.close();
404 
405             /*Terminal DEBUG*/
406             printf ("\n Suma Diferencias:%f\n",sumDecision);
407             for(int i=0; i<ciclos;i++){
408                     printf ("Ciclo%d    Average3:%f  Average11:%f  Average9
                        :%f  Average7:%f\n",i,av3T[i],av11T[i],av9T[i],av7T[
                        i]);
409             }
410             printf ("\nTotal_Average3:%f\nAverage11:%f\nAverage9:%f\
                    nAverage7:%f\n",av3/ITERACIONES,av11/ITERACIONES,av9/
                    ITERACIONES,av7/ITERACIONES);
411 
412             return 0;
413     }
```

C.3. Codigo representación gráfica. Gnuplot

C.3.1. plot hypothesis .gp

```
#!/usr/bin/gnuplot
#Need Hipothesis variable in File in order to choose a style color in differential lines


set term eps

set key off
set xlabel "Samples (s)"
set xrange [0:203]
set x2range [0:203]
set ylabel "Concentration[Points]"
set y2label "Temperature[%]"
set ytics nomirror
set y2tics
set tics out
set autoscale y
set autoscale y2

set style line 1 lt rgb "red" lw 1
set style line 2 lt rgb "green" lw 1
set style line 3 lt rgb "blue" lw 1
set style line 4 lt rgb "grey" lw 2
set style line 5 lt rgb "black" lw 2
set style line 6 lt rgb "pink" lw 1

#Plot signals of concentration from sensors
set output "Samples_0-1Pa.eps"

plot "0-1_Olus.txt" using 1:2  with filledcurve x1 ls 1 axis x1y1,"" using 1:3  with filledcurve x1 ls 3 axis x1y1,"" using 1:2  with lines ls 1 axis x1y1,"" using 1:3 with lines ls 3 axis x1y1,"" using 1:4 with lines ls 4 axis x2y2,"" using 1:5 with lines ls 4 axis x2y2

set output

#Plot averages of concentration from sensors
set output "Mean_Samples_0-1Pa.eps"
plot "0-1_Average.txt" using 1:2  with filledcurve x1 ls 1 axis x1y1,"" using 1:3  with filledcurve x1 ls 3 axis x1y1,"" using 1:2  with lines ls 1 axis x1y1,"" using 1:3 with lines ls 3 axis x1y1,"" using 1:4 with lines ls 4 axis x2y2,"" using 1:5 with lines ls 4 axis x2y2,"" using 1:4 with points ls 4 axis x2y2,"" using 1:4 with lines ls 4 axis x2y2

set output

#Plot difference between averages of concentration from sensors
set output "Diff0-1Pa.eps"      #Set colour in order of hypothesis parameter
plot "0-1_Average.txt" using 1:($6 > 0? ($2-$3):1/0) with line ls 1 axis x1y1 ,\
  "" using 1:($6 < 0? ($2-$3):1/0) with line ls 3 axis x1y1 ,\
```

```
43    "" using 1:($6 == 0? ($2-$3):1/0) with line ls 5 axis x1y1 ,"" using
          1:5  with lines ls 4 axis x2y2,"" using 1:4 with points ls 4 axis
          x2y2,"" using 1:4 with lines ls 4 axis x2y2
44
45  set output
46
47
48  #Plot signals of concentration from sensors
49  set output "Samples_2-3Pe.eps"
50
51  plot "2-3_Olus.txt"  using 1:2  with filledcurve x1 ls 2 axis x1y1,""
       using 1:3  with filledcurve x1 ls 6 axis x1y1,"" using 1:2  with
       lines ls 2 axis x1y1,"" using 1:3 with lines ls 6 axis x1y1,"" using
        1:4 with lines ls 4 axis x2y2,"" using 1:5 with lines ls 4 axis
       x2y2
52
53  set output
54
55  #Plot averages of concentration from sensors
56  set output "Mean_Samples_2-3Pa.eps"
57  plot "2-3_Average.txt" using 1:2  with filledcurve x1 ls 2 axis x1y1,""
        using 1:3  with filledcurve x1 ls 6 axis x1y1,"" using 1:2  with
       lines ls 2 axis x1y1,"" using 1:3 with lines ls 6 axis x1y1,"" using
        1:4 with lines ls 4 axis x2y2,"" using 1:5 with lines ls 4 axis
       x2y2,"" using 1:4 with points ls 4 axis x2y2,"" using 1:4 with lines
        ls 4 axis x2y2
58
59  set output
60
61  #Plot difference between averages of concentration from sensors
62  set output "Diff2-3Pe.eps"      #Set colour in order of hypothesis
       parameter
63  plot "2-3_Average.txt" using 1:($6 > 0? ($2-$3):1/0) with line ls 2
       axis x1y1 ,\
64    "" using 1:($6 < 0? ($2-$3):1/0) with line ls 6 axis x1y1 ,\
65    "" using 1:($6 == 0? ($2-$3):1/0) with line ls 5 axis x1y1 ,"" using
          1:5  with lines ls 4 axis x2y2,"" using 1:4 with points ls 4 axis
          x2y2,"" using 1:4 with lines ls 4 axis x2y2
66
67  set output
```

C.4. Codigo ensamblado. Scripting

C.4.1. robot loop

```
#!/bin/sh
# Script of odour localization
# Input Variables:
#                                    * Timeout Iteration Value
# @ Alejandro Pequeno
# @ GNB. Universidad Autonoma de Madrid
loop=0
count=0
distance=100
offsetPar=9      #//Valor marcado por la diferencia entre sensor Delantero(R) y Trasero(A)
offsetTan=43     #//Valor marcado por la diferencia entre sensor Derecho(V) e Izquierdo(M)

while [ $loop != 1 ]
do
        let count++

        #******************Sampling*************************#
        echo Sampling_Decission_$count
        ./sampling $offsetPar $offsetTan $count
        #*******************--------*************************#

        #******************Taking Decission*****************#
        while read line
        do
            echo -e "$line" && var=$line
        done < decission.txt   #Nombre fichero

        if [[ "$var" = "F" ]] ; then
                echo 'Forward'
                ./exec_moveRobot $distance
        elif [[ "$var" = 'B' ]] ; then
                echo 'Backward'
                ./exec_moveRobot -$distance
        elif [[ "$var" = 'U' ]] ; then
                echo 'Stay'
        elif [[ "$var" = "XF" ]] ; then
                echo 'Source At 20cm Forward'
                ./exec_moveRobot 100
                let loop=1
        elif [[ "$var" = "XB" ]] ; then
                echo 'Source At 20cm Backward'
                ./exec_moveRobot -100
                let loop=1
        else
                echo 'Nothing'
        fi

        for FILE in decission* ; do mv $FILE "$count"_$FILE ; done
        #*******************----------*************************#
done
```

C.4.2. sampling

```
#!/bin/bash
# Execute sampling program of noses

cd /home/TestNariz/src
./test_nariz_basico $1 $2
cd /home/TestNariz/src/muestras/Robot/ #carpeta destino del experimento
./script_gnuplot_Parametrizable
./scriptNombreParam $3
                  #parametro del nombre
```

C.4.3. exec moveRobot

```
 #!/bin/bash
# Execute remote code in BBB system

USERNAME=root
HOSTS="beaglebone.local"
SCRIPT="pwd; cd Desktop/BeaglePythonProjects ; python move.py "$1""

sshpass -p "*******" ssh -o StrictHostKeyChecking=no root@beaglebone.
   local "${SCRIPT}"
```

C.5. Código sensor humedad-temperatura am2302

```
#!/usr/bin/python
# Copyright (c) 2014 Adafruit Industries
# Author: Tony DiCola
# Modifications: Alejandro Pequeno

import Adafruit_DHT
import time
import os
import sys


SAMPLES=25
stime=2


nameDHT="DHT.txt" ;
fileString= nameDHT ;
if os.path.exists(fileString):
        localtime   = time.localtime()
        now = time.strftime("%HH%MM%SS_", localtime)
        dst= now+fileString
        os.rename(fileString, dst)

sensor = Adafruit_DHT.AM2302
pin = 'P9_41'


dht= open(fileString, "w+")
ite=0
wline='Ite'+'   '+'Temp(C)'+'   '+'Hum(%)'+'\n'
dht.writelines(wline)
localtime   = time.localtime()
print time.strftime("%HH%MM%SS_", localtime)

while ite<SAMPLES:
        humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
        if humidity is not None and temperature is not None:
                print 'Temp={0:0.1f}*C Humidity={1:0.1f}%'.format(
                   temperature, humidity)
                localtime   = time.localtime()
                print time.strftime("%HH%MM%SS_", localtime)
                sys.stdout.flush()
                temp=   round(temperature,1)
                hum=    round(humidity,1)
                wline=str(ite)+'        '+str(temp)+'   '+str(hum)+'\n'
        else:
                print 'Failed to get reading. Try again!'
                wline="Fail"
        dht.writelines(wline)
        ite+=1
localtime   = time.localtime()
print time.strftime("%HH%MM%SS_", localtime)
dht.close
```

C.6. Protocolo de inicio de las narices electrónicas

C.6.1. Script

```
 #!/bin/sh

# Initial Protocol
# Input parameters:
#                  $1 : number of repetitions of measure
#                  $2 : Temperature Parallel
#                  $3 : Temperature Perpend
#                  $4 : cleanUp1to1.py [0 Off / 1 On]
# @ Alejandro Pequeno
# @ GNB. Universidad Autonoma de Madrid

loop=0

if [ $# != 4 ]
then

        echo "bash loop numberOfExecutions TempParallel
           TempPerpendicular 0/1(clean-up)"
        exit
fi

cmdae="python cleanUp1to1.py 0 1 1"
cmdao="python cleanUp1to1.py 2 3 1"


cmdbe="python measure1to1.py $2 $2 0 1 1"
cmdbo="python measure1to1.py $3 $3 2 3 1"


cmdplot="gnuplot plot_basic.gp"
cmdscript="bash scriptNombreParam STA$2$3"

if [ $4 == 1 ]
then
        echo "cleanUp1to1 Phase"
        $cmdae & $cmdao
        echo "Exiting"
        wait
        $cmdplot
        $cmdscript
fi

while [ $loop != $1 ]
do
        echo $loop
        let loop++
        $cmdbe & $cmdbo
        echo "Exiting"
        echo $$
        wait
        $cmdplot
        $cmdscript
done
```

C.6.2. cleanUp1to1.py

```
 #!/usr/bin/python
"""
    Lineal increment protocol in order to clean up the sensor
    @ Alejandro Pequeno
    @ Grupo de Neurocomputacion Biologica
    @ Universidad Autonoma de Madrid

    Keyword arguments:
    index of sensor1 --------- int variable [0,1,2,3]
    index of sensor2 --------- int variable [0,1,2,3]
    stay Alive Temperature --- int flag indicates continue Temperature
       in pin when finish 0 or 1
"""

import Adafruit_BBIO.ADC as ADC
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.PWM as PWM
#from matplotlib import pyplot as plt
import time
import os
import sys
import threading
import math

SLEEP=1
SAMPLES=200
DEBUG=1
TINI=25
SALTO=0.375

# Motor Pins -- (LEFT, RIGHT)
senPin = ('P9_38','P9_33','P9_37','P9_39') #(Black, Green, Yellow, Red)
heatPin = ('P8_13','P8_19','P9_21','P9_22')

#Threading Locks
ADCLock = threading.Lock()
ExitLock= threading.Lock()

#Modulating Variables --> Lineal protocol
NP = 10
VAR = 1.66

#Temperature Working Range
TMIN=0
TMAX=100

def setup(start_temp1,start_temp2, ind1, ind2):
        ADC.setup()
        PWM.start(heatPin[ind1], start_temp1, 2000, 0)
        PWM.start(heatPin[ind2], start_temp2, 2000, 0)

def main():

        if(len(sys.argv)<4):
                print 'Error define parameter sensor1[0,1,2,3] sensor2
                   [0,1,2,3] stayAlive[0|1]'
```

```
55                  return 0
56          ind1=int(sys.argv[1])
57          ind2=int(sys.argv[2])
58          stayAlive=int(sys.argv[3])
59          #File
60          nameF="Olus.txt" ;      nameAv= "Average.txt"
61          fileString= str(ind1)+"-"+str(ind2)+"_"+nameF ; avString= str(
               ind1)+"-"+str(ind2)+"_"+nameAv
62          if os.path.exists(fileString):
63                  localtime   = time.localtime()
64                  now = time.strftime("%HH%MM%SS_", localtime)
65                  dst= now+fileString
66                  os.rename(fileString, dst)
67          if os.path.exists(avString):
68                  localtime   = time.localtime()
69                  now = time.strftime("%HH%MM%SS_", localtime)
70                  dst= now+avString
71                  os.rename(avString, dst)
72
73
74          # Open a file
75          fo = open(fileString, "w+")
76          count=0
77          x=[]
78          concent1=[];    concent2=[]
79          temp1=[];       temp2=[]
80          avList1=[];     avList2=[];     avTempList1=[]; avTempList2=[]
81          conList1=[];    conList2=[];
82          temperature=TINI
83          setup(temperature, temperature, ind1, ind2)
84          while count<SAMPLES:
85                  #Measure
86                  PWM.set_duty_cycle(heatPin[ind1], temperature)
87                  PWM.set_duty_cycle(heatPin[ind2], temperature)
88                  time.sleep(SLEEP)
89                  value1=ADC.read_raw(senPin[ind1])
90                  value2=ADC.read_raw(senPin[ind2])
91                  x.append(count)
92                  concent1.append(value1)
93                  concent2.append(value2)
94                  temp1.append(temperature)
95                  temp2.append(temperature)
96
97                  #Average Values
98                  conList1.append(value1)
99                  conList2.append(value2)
100                 average1=0; average2=0
101                 #Adding concentrate to calculate average
102                 for value1,value2 in zip(conList1,conList2) :
103                         average1+=value1
104                         average2+=value2
105
106                 #Loop iteracion
107                 count+=1
108                 temperature += SALTO
109                 #Keep between the working range
110                 if(temperature>TMAX):
111                                 temperature=TMAX
112                 elif(temperature<TMIN):
```

```
113                                     temperature=TMIN
114
115                     #Calculate average
116                     if (count)<NP :
117                             avList1.append(round(average1/(count),2))
118                             avList2.append(round(average2/(count),2))
119                             avTempList1.append(temperature)
120                             avTempList2.append(temperature)
121                     else :
122                             avList1.append(round(average1/NP,2))
123                             avList2.append(round(average2/NP,2))
124                             avTempList1.append(temperature)
125                             avTempList2.append(temperature)
126                             del conList1[0];
127                             del conList2[0];
128
129                     #Export Data
130                     wline=str(count-1)+'    '+str(value1)+' '+str(value2)+'
                            '+str(temperature)+'    '+str(temperature)+'\n'
131                     fo.writelines(wline)
132                     if DEBUG:
133                             print 'Muestra[',count-1,']     Value: ',value1
                                ,'----',value2,'  Temperature: ',temperature
                                ,'----',temperature
134             if not stayAlive :
135                     PWM.cleanup()
136                     print "Cleaning Up PWM"
137             fo.close()
138
139             avo= open(avString, "w+")
140             for index in range(len(avList1)) :
141                     wline=str(index)+'      '+str(avList1.pop(0))+' '+str(
                        avList2.pop(0))+' '+str(avTempList1.pop(0))+'     '+
                        str(avTempList2.pop(0))+'     '+'\n'
142                     avo.writelines(wline)
143             avo.close()
144             return 0
145
146     if __name__ == '__main__':
147         main()
```

C.6.3. measure1to1.py

```
 #!/usr/bin/python
"""
    Sensing concentration in two sensors with Modulation.
    @ Alejandro Pequeno
    @ Grupo de Neurocomputacion Biologica
    @ Universidad Autonoma de Madrid

    Keyword arguments:
    temperature of Sensor1 --- int variable between [0-100]
    temperature of Sensor2 --- int variable between [0-100]
    index of sensor1 --------- int variable [0,1,2,3]
    index of sensor2 --------- int variable [0,1,2,3]
    stay Alive Temperature --- int flag indicates continue Temperature
       in pin when finish 0 or 1
"""

import Adafruit_BBIO.ADC as ADC
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.PWM as PWM
import time
import os
import sys
import threading
import math

SLEEP=1
SAMPLES=204
DEBUG=1

#Threading Locks
ADCLock = threading.Lock()
ExitLock= threading.Lock()

#Modulating Variables
NP = 12
VAR = 0.98

#Temperature Working Range
TMIN=0
TMAX=100

def setup(start_temp1,start_temp2, ind1, ind2):
        ADC.setup()
        PWM.start(heatPin[ind1], start_temp1, 2000, 0)
        PWM.start(heatPin[ind2], start_temp2, 2000, 0)

def main():

        if(len(sys.argv)<6):
                print 'Error define parameter temperatureS1[0-100]
                   temperatureS2[0-100] sensor1[0,1,2,3] sensor2
                   [0,1,2,3] stayAlive[0|1]'
                return 0
        avTemp1=float(sys.argv[1])
        avTemp2=float(sys.argv[2])
        ind1=int(sys.argv[3])
```

```
54          ind2=int(sys.argv[4])
55          stayAlive=int(sys.argv[5])
56          if avTemp1<0 or avTemp1>100 or avTemp2<0 or avTemp2>100 or ind1
               <0 or ind1>3 or ind2<0 or ind2>3 :
57                  print "Value parametric Error *temperatureS1[0-100]
                       temperatureS2[0-100] sensor1[0,1,2,3] sensor2
                       [0,1,2,3] stayAlive[0|1]*"
58                  return 0
59          #File
60          nameF="Olus.txt" ;      nameAv= "Average.txt"
61          fileString= str(ind1)+"-"+str(ind2)+"_"+nameF ; avString= str(
               ind1)+"-"+str(ind2)+"_"+nameAv
62          if os.path.exists(fileString):
63                  localtime   = time.localtime()
64                  now = time.strftime("%HH%MM%SS_", localtime)
65                  dst= now+fileString
66                  os.rename(fileString, dst)
67          if os.path.exists(avString):
68                  localtime   = time.localtime()
69                  now = time.strftime("%HH%MM%SS_", localtime)
70                  dst= now+avString
71                  os.rename(avString, dst)
72
73
74          # Open a file
75          fo = open(fileString, "w+")
76          count=0 ;
77          x=[]
78          concent1=[];    concent2=[]
79          temp1=[];       temp2=[]
80          avList1=[];     avList2=[];     avTempList1=[]; avTempList2=[]
81          conList1=[];    conList2=[];
82          setup(avTemp1, avTemp2, ind1, ind2)
83          while count<SAMPLES:
84                  #Measure
85                  temperature1 = avTemp1 + round(VAR*math.sin(2*math.pi*
                       count/NP),2)
86                  temperature2 = avTemp2 + round(VAR*math.sin(2*math.pi*
                       count/NP),2)
87                  PWM.set_duty_cycle(heatPin[ind1], temperature1)
88                  PWM.set_duty_cycle(heatPin[ind2], temperature2)
89                  time.sleep(SLEEP)
90                  value1=ADC.read_raw(senPin[ind1])
91                  value2=ADC.read_raw(senPin[ind2])
92                  x.append(count)
93                  concent1.append(value1)
94                  concent2.append(value2)
95                  temp1.append(temperature1)
96                  temp2.append(temperature2)
97
98                  #Average Values
99                  conList1.append(value1)
100                 conList2.append(value2)
101                 average1=0; average2=0
102
103                 #Adding concentrate to calculate average
104                 for value1,value2 in zip(conList1,conList2) :
105                         average1+=value1
106                         average2+=value2
```

```
107
108                 #Loop iteracion
109                 count+=1
110
111                 #Calculate average
112                 if (count)<NP :
113                         avList1.append(round(average1/(count),2))
114                         avList2.append(round(average2/(count),2))
115                         avTempList1.append(avTemp1)
116                         avTempList2.append(avTemp2)
117                 else :
118                         avList1.append(round(average1/NP,2))
119                         avList2.append(round(average2/NP,2))
120                         avTempList1.append(avTemp1)
121                         avTempList2.append(avTemp2)
122                         del conList1[0];
123                         del conList2[0];
124                 #Export Data
125                 wline=str(count-1)+'    '+str(value1)+' '+str(value2)+'
                        '+str(temperature1)+'   '+str(temperature2)+'\n'
126                 fo.writelines(wline)
127                 if DEBUG:
128                         print 'Muestra[',count-1,']     Value: ',value1
                            ,'----',value2,'  Temperature: ',
                            temperature1,'----',temperature2
129         PWM.set_duty_cycle(heatPin[ind1], avTemp1)
130         PWM.set_duty_cycle(heatPin[ind2], avTemp2)
131         if not stayAlive :
132                 PWM.cleanup()
133                 print "Cleaning Up PWM"
134         fo.close()
135
136         avo= open(avString, "w+")
137         for index in range(len(avList1)) :
138                 wline=str(index)+'      '+str(avList1.pop(0))+' '+str(
                    avList2.pop(0))+' '+str(avTempList1.pop(0))+'     '+
                    str(avTempList2.pop(0))+'     '+'\n'
139                 avo.writelines(wline)
140         avo.close()
141         return 0
142
143 if __name__ == '__main__':
144     main()
```

C.7. Protocolo de ajuste de offset

C.7.1. ajusteOffset

```
 #!/usr/bin/python
"""
    Adjust the variable difference between two sensors.
    Decissions dependant the minimize:
        *Differential Average between sensors > EDGEDIF
    #Reprogram offset later and before an execution

    @ Alejandro Pequeno
    @ Grupo de Neurocomputacion Biologica
    @ Universidad Autonoma de Madrid

    Keyword arguments:
    temperature of Sensor1 --- int variable between [0-100]
    temperature of Sensor2 --- int variable between [0-100]
    index of sensor1 --------- int variable [0,1,2,3]
    index of sensor2 --------- int variable [0,1,2,3]
    stay Alive Temperature --- int flag indicates continue Temperature
       in pin when finish 0 or 1
"""

import Adafruit_BBIO.ADC as ADC
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.PWM as PWM
import Adafruit_DHT

import time
import os
import sys
import threading
import math
import subprocess

#############SYSTEM PARAMETERS####################
TIMEOUT=10
SLEEP=1
SAMPLES=204
DEBUG=0

IND1=0
IND2=1
IND3=2
IND4=3

LIMITABOVE=0.3
LIMITBELOW=-0.3
LIMITAVABOVE=1
LIMITAVBELOW=-1
LIMITT=1        #Minimun GAP to consider an incoming plume
################################################
############DHT PARAM#############
DHTSAMPLES=30
DHTPIN='P9_41'
#################################
# Motor Pins -- (LEFT, RIGHT)
```

```
54  senPin = ('P9_38','P9_33','P9_37','P9_39') #(Black, Green, Yellow, Red)
55  heatPin = ('P8_13','P8_19','P9_21','P9_22')
56
57  #Threading Locks
58  Sampling= threading.Semaphore(2)
59  Execute= threading.Semaphore(2)
60  Queue= threading.Semaphore(2)
61  ExitLock= threading.Lock()
62
63  #Globals
64  exitFlag=0
65  directionE='A'
66  direction0='B'
67
68  #Modulating Variables
69  NP = 12
70  VAR = 0.98
71
72  #Algorithm Parameters
73  CHVAL= 2 #Number of values for changing the optimization adjust
74  KP = 1.8/10
75  KI = 0.54/10
76
77  #Temperature Working Range parameter
78  TMIN=28
79  TMAX=40
80
81  class myThread (threading.Thread):
82          def __init__(self, threadID, name, param1, param2, param3,
                param4, param5, param6, param7):
83                  threading.Thread.__init__(self)
84                  self.threadID = threadID
85                  self.name = name
86                  self.param1 = param1
87                  self.param2 = param2
88                  self.param3 = param3
89                  self.param4 = param4
90                  self.param5 = param5
91                  self.param6 = param6
92                  self.param7 = param7
93          def run(self):
94                  print "Starting " + self.name
95                  if self.name == 'Measure':
96                          Queue.acquire()
97                          sampling(self.param1,self.param2,self.param3,
                                self.param4,self.param5,self.param6,self.
                                param7)
98                  elif self.name == 'Execute':
99                          execute(self.param1)
100                 elif self.name == 'DHT':
101                         sensor = Adafruit_DHT.AM2302
102                         sampleTH(sensor,DHTPIN)
103                 else:
104                         print "Error in self.name = " + self.name
105
106 def setup(start_temp1,start_temp2, ind1, ind2):
107         ADC.setup()
108         PWM.start(heatPin[ind1], start_temp1, 2000, 0)
109         PWM.start(heatPin[ind2], start_temp2, 2000, 0)
```

```
110
111 def sampling(temperature1, temperature2, sensor1, sensor2, offsetExt,
    stayAlive, axis):
112         global exitFlag
113         global directionE; global directionO
114         av1TotOff=0;    av2TotOff=0
115         tempINI1=temperature1
116         tempINI2=temperature2
117         #File
118         nameF="Olus.txt" ;      nameAv= "Average.txt";  nameOf= "Offset
                .txt";   nameDe= "Decission.txt"
119         fileString= str(sensor1)+"-"+str(sensor2)+"_"+nameF ; avString=
                str(sensor1)+"-"+str(sensor2)+"_"+nameAv
120         ofString= str(sensor1)+"-"+str(sensor2)+"_"+nameOf ;  deString=
                str(sensor1)+"-"+str(sensor2)+"_"+nameDe
121
122         avTemp1=temperature1
123         avTemp2=temperature2
124         setup(avTemp1, avTemp2, sensor1, sensor2)
125         ii=0
126         controlbucle=0
127         while ii<NP: #First Cycle. Warm-Up
128                 ii+=1
129                 temperature1 = avTemp1 + round(VAR*math.sin(2*math.pi*
                        ii/NP),2)
130                 temperature2 = avTemp2 + round(VAR*math.sin(2*math.pi*
                        ii/NP),2)
131                 PWM.set_duty_cycle(heatPin[sensor1], temperature1)
132                 PWM.set_duty_cycle(heatPin[sensor2], temperature2)
133                 time.sleep(SLEEP)
134                 av1TotOff+=ADC.read_raw(senPin[sensor1])
135                 av2TotOff+=ADC.read_raw(senPin[sensor2])
136         offset=(av1TotOff/NP)-(av2TotOff/NP)
137         print 'Offset Ini: ',offset
138         ExitLock.acquire()
139         while not exitFlag:
140                 ExitLock.release()
141         #Control Loop Begining
142                 Sampling.acquire()
143                 Queue.release()
144                 print "Sampling ",axis
145                 if os.path.exists(fileString):
146                         localtime   = time.localtime()
147                         now = time.strftime("%HH%MM%SS_", localtime)
148                         dst= now+fileString
149                         os.rename(fileString, dst)
150                 if os.path.exists(avString):
151                         localtime   = time.localtime()
152                         now = time.strftime("%HH%MM%SS_", localtime)
153                         dst= now+avString
154                         os.rename(avString, dst)
155                 if os.path.exists(ofString):
156                         localtime   = time.localtime()
157                         now = time.strftime("%HH%MM%SS_", localtime)
158                         dst= now+ofString
159                         os.rename(ofString, dst)
160                 if os.path.exists(deString):
161                         localtime   = time.localtime()
162                         now = time.strftime("%HH%MM%SS_", localtime)
```

```
163                         dst= now+deString
164                         os.rename(deString, dst)
165                 # Open a file
166                 fo = open(fileString, "w+")
167                 count=0 ;       flagState= 0 ;  Hpositive=0 #
                        Independent Decission
168                 incrementI=0;   differentialTemp=1;     decissionTemp=0
169                 av1Total=0;     av2Total=0
170                 x=[]
171                 concent1=[];    concent2=[]
172                 temp1=[];       temp2=[]
173                 avList1=[];     avList2=[];     avTempList1=[];
                        avTempList2=[]; hypoList=[]
174                 conList1=[];    conList2=[];
175                 avTemp1Sta=avTemp1
176                 avTemp2Sta=avTemp2
177
178                 #Measure Loop Begining
179                 while count<SAMPLES:
180                         #Measure
181                         temperature1 = avTemp1 + round(VAR*math.sin(2*
                                math.pi*count/NP),2)
182                         temperature2 = avTemp2 + round(VAR*math.sin(2*
                                math.pi*count/NP),2)
183                         PWM.set_duty_cycle(heatPin[sensor1],
                                temperature1)
184                         PWM.set_duty_cycle(heatPin[sensor2],
                                temperature2)
185                         time.sleep(SLEEP)
186                         value1=ADC.read_raw(senPin[sensor1])
187                         value2=(ADC.read_raw(senPin[sensor2])+offset)
188                         x.append(count)
189                         concent1.append(value1)
190                         concent2.append(value2)
191                         temp1.append(temperature1)
192                         temp2.append(temperature2)
193
194                         #Average Values
195                         conList1.append(value1)
196                         conList2.append(value2)
197                         average1=0; average2=0
198                         #Adding concentrate to calculate average
199                         for value1,value2 in zip(conList1,conList2) :
200                                 average1+=value1
201                                 average2+=value2
202
203                         #Loop iteracion
204                         count+=1
205
206                         #Calculate average
207                         if (count)<NP :
208                                 avList1.append(round(average1/(count)
                                        ,2))
209                                 avList2.append(round(average2/(count)
                                        ,2))
210                                 avTempList1.append(avTemp1)
211                                 avTempList2.append(avTemp2)
212                                 hypoList.append(Hpositive)
213                         else :
```

```
214                                     avList1.append(round(average1/NP,2))
215                                     avList2.append(round(average2/NP,2))
216                                     avTempList1.append(avTemp1)
217                                     avTempList2.append(avTemp2)
218                                     hypoList.append(Hpositive)
219                                     del conList1[0];
220                                     del conList2[0];
221                             #Export Data
222                             wline=str(count-1)+'    '+str(value1)+' '+str(
                                    value2)+' '+str(temperature1)+'   '+str(
                                    temperature2)+'\n'
223                             fo.writelines(wline)
224                             if DEBUG:
225                                     print 'Muestra[',count-1,']     Value:
                                        ',value1,'----',value2,'
                                        Temperature: ',temperature1,'----',
                                        temperature2
226                                     sys.stdout.flush()
227
228
229                             #Adjust Temperature
230                             if count%NP==0 and count!=NP :
231                                     difference=avList1[-1] - avList2[-1]
232                                     #Initial Param State
233                                     if flagState==0:
234                                             if (difference) > 0:
235                                                     Hpositive=1
236                                             else:
237                                                     Hpositive=-1
238                                             flagState=1
239                                     else :
240                                             controlParamP=0
241                                             for index in range(0,CHVAL):
242                                                     if (index*NP+1)>len(
                                                        avList1):
243                                                             break
244                                                     elif (avList1[-1-int(
                                                        index*NP)] - avList2
                                                        [-1-int(index*NP)])
                                                        >0:
245                                                             controlParamP
                                                                +=1
246                                             if controlParamP==CHVAL:
247                                                     Hpositive=1
248                                             elif controlParamP==0:
249                                                     Hpositive=-1
250                                     #Sign of the increment differential
251                                     if (avList1[-1]-avList1[-NP-1])-(
                                        avList2[-1]-avList2[-NP-1])>0:
252                                             Dpositive=1
253                                     else:
254                                             Dpositive=-1
255                                     #Sign of the past increment
256                                     actualSign=1
257                                     actualSign=math.copysign(actualSign,
                                        differentialTemp)
258
259                                     #Differential Increments
```

```
260                                 incrementP=(avList1[-1]-avList1[-NP-1])
                                        -(avList2[-1]-avList2[-NP-1])
261                                 incrementI+=incrementP
262                                 differentialTemp=(Hpositive*Dpositive*
                                        actualSign)*abs(round(incrementP*KP-
                                        incrementI*KI,1))
263                                 avTemp1+= differentialTemp
264                                 avTemp2+= differentialTemp
265
266                                 #Deccision
267                                 if differentialTemp > LIMITT :
268                                         decissionTemp= difference
269                                 #Keep between the working range
270                                 if(avTemp1>TMAX-VAR):
271                                                 avTemp1=TMAX-VAR
272                                 elif(avTemp1<TMIN+VAR):
273                                                 avTemp1=TMIN+VAR
274                                 if(avTemp2>TMAX-VAR):
275                                                 avTemp2=TMAX-VAR
276                                 elif(avTemp2<TMIN+VAR):
277                                                 avTemp2=TMIN+VAR
278                                 if DEBUG:
279                                         print 'Sign: SensorLead',
                                                Hpositive, '   DifIncrements
                                                ',Dpositive,'
                                                ActualIncrease',actualSign
280                                         sys.stdout.flush()
281                                 print 'Sensores ',sensor1,'-',sensor2,'
                                        Average:                ',avList1
                                        [-1],'----',avList2[-1],'
                                        DifIncrements ',incrementP,'
                                        SumDifIncrements ',incrementI
282                                 sys.stdout.flush()
283                                 print 'Sensores ',sensor1,'-',sensor2,'
                                        Temperature:            ','
                                        ChanAvTemp ',differentialTemp,'
                                              Average ',avTemp1,'     ',
                                        avTemp2,'\n'
284                                 sys.stdout.flush()
285                         PWM.set_duty_cycle(heatPin[sensor1], avTemp1)
286                         PWM.set_duty_cycle(heatPin[sensor2], avTemp2)
287
288                         #Measure Loop End
289                         fo.close()
290                         #Averages File
291                         elements=len(avList1)
292                         min1=min(avList1[NP:len(avList1)])
293                         min2=min(avList2[NP:len(avList2)])
294                         max1=max(avList1[NP:len(avList1)])
295                         max2=max(avList2[NP:len(avList2)])
296                         decissionAv=abs(max1-min1)-abs(max2-min2) # + => F   -
                                => B
297                         avo= open(avString, "w+")
298                         for index in range(len(avList1)) :
299                                 av1Total+=avList1[0]
300                                 av2Total+=avList2[0]
301                                 av1Last=avList1[0]
302                                 av2Last=avList2[0]
```

```
303                             wline=str(index)+'      '+str(avList1.pop(0))+'
                                    '+str(avList2.pop(0))+' '+str(avTempList1.
                                   pop(0))+'     '+str(avTempList2.pop(0))+'
                                        '+str(hypoList.pop(0))+'\n'
304                             avo.writelines(wline)
305                     avo.close()
306                     #Offset File
307                     avTemp1End=avTemp1
308                     avTemp2End=avTemp2
309                     decissionDiff=(av1Total-av2Total)/elements
310                     ofo = open(ofString, "w+")
311                     wline=str(sensor1)+'    '+str(sensor2)+'        '+'
                           StartAv'+'   '+str(avTemp1Sta)+'     '+str(
                           avTemp2Sta)+'     '+'FinAv'+'     '+str(avTemp1End)+
                           '     '+str(avTemp2End)+'\n'
312                     wline2='Offset '+'Ini '+str(offset)+'   Calc '+str(
                           decissionDiff)+'\n'
313                     ofo.writelines(wline)
314                     ofo.writelines(wline2)
315                     ofo.close()
316                     #Reprogram offset later an execution
317                     offset+=decissionDiff
318                     #Decission File
319                     #* Base on Total Average and difference between max -
                           min in average and Prioriry in Plume incoming(
                           Measure by the gap in temperature)
320                             #DecissionDiff
321                     if round(decissionDiff,2)>LIMITABOVE:
322                             decDiff=1#F
323                     elif round(decissionDiff,2)<LIMITBELOW:
324                             decDiff=-1#B
325                     else:
326                             decDiff=0#X
327
328                             #DecissionAv
329                     if round(decissionAv,2)>LIMITAVABOVE:
330                             decAv=1#F
331                     elif round(decissionAv,2)<LIMITAVBELOW:
332                             decAv=-1#B
333                     else:
334                             decAv=0#X
335                     if axis=='Parallel':
336                             if decDiff>0:
337                                     direction='F'   #Forward
338                                     directionE='F'
339                             elif decDiff<0:
340                                     direction='B'   #Backward
341                                     directionE='B'
342                             else:
343                                     direction='X'   #Don't Know
344                                     directionE='X'
345                     elif axis=='Perpendicular':
346                             if decDiff>0:
347                                     direction='E'   #Forward
348                                     directionO='E'
349                             elif decDiff<0:
350                                     direction='W'   #Backward
351                                     directionO='W'
352                             else:
```

```
353                                     direction='X'   #Don't Know
354                                     direction0='X'
355                     else:
356                             direction='U'           #Undefine
357                     defo = open(deString, "w+")
358                     wline= "Offset  Difference      Average
                           Temperature     Total\n"
359                     wline2= str(offset)+'   '+str(decissionDiff)+'  '+str(
                           decissionAv)+'    '+str(decissionTemp)+'  '+str(
                           direction)+'\n'
360                     defo.writelines(wline)
361                     defo.writelines(wline2)
362                     defo.close()
363                     print 'Axis ',axis,'    ',wline2
364
365                     #Semaphores
366                     Sampling.release()
367                     Execute.acquire()
368                     Execute.release()
369                     controlbucle+=1
370                     ExitLock.acquire()
371             ExitLock.release()
372             #Control Loop End
373             Sampling.release()
374             print "Finish ",axis
375             if not stayAlive :
376                     PWM.cleanup()
377                     print "Cleaning Up PWM"
378
379     def execute(param1):
380             global exitFlag
381             global directionE; global direction0
382             Queue.acquire()
383             Queue.acquire()
384             print "Execute ",param1
385             controlTime=1; controlLoop=1
386             while controlLoop :
387                     #Execute Lock Both Measures
388                     Execute.acquire()
389                     Execute.acquire()
390                     #Execute Wait Final Measure
391                     Sampling.acquire()
392                     Sampling.acquire()
393                     #Moves robot due to decissions global param direction
                           protected by semaphore
394                     print "Adjust ",directionE,direction0,"SHAKE"
395                     sys.stdout.flush()
396                     if (directionE == 'X' and direction0 == 'X') or
                           controlTime>=TIMEOUT:
397                             ExitLock.acquire()
398                             exitFlag=1
399                             ExitLock.release()
400                             controlLoop=0
401                     subprocess.call(["gnuplot", "plot_hyp.gp"])
402                     subprocess.call(["bash", "scriptNombreParam", "
                           AdjustOFF"])
403                     #Continue Measure
404                     Execute.release()
405                     Execute.release()
```

```
406                 Sampling.release()
407                 Sampling.release()
408                 time.sleep(10)
409                 controlTime+=1
410         print "Exiting ",param1
411
412 def sampleTH(sensor,pin):
413         ite=0
414         nameDHT="DHT.txt" ;
415         fileString= nameDHT ;
416         if os.path.exists(fileString):
417                 localtime   = time.localtime()
418                 now = time.strftime("%HH%MM%SS_", localtime)
419                 dst= now+fileString
420                 os.rename(fileString, dst)
421
422         dht= open(fileString, "w+")
423         wline='Iteracion'+'     '+'Temperatura(C)'+'    '+'Humedad(%)'+
              '\n'
424         dht.writelines(wline)
425         localtime   = time.localtime()
426
427         while ite<SAMPLES:
428                 humidity, temperature = Adafruit_DHT.read_retry(sensor,
                      pin)
429                 if humidity is not None and temperature is not None:
430                         if DEBUG:
431                                 print 'Temp={0:0.1f}*C Humidity={1:0.1f
                                    }%'.format(temperature, humidity)
432                                 sys.stdout.flush()
433                         temp=   round(temperature,1)
434                         hum=    round(humidity,1)
435                         wline=str(ite)+'        '+str(temp)+'   '+str(
                                    hum)+'\n'
436                 else:
437                         print 'Failed to get reading. Try again!'
438                         wline="MACK"
439                 dht.writelines(wline)
440                 ite+=1
441         localtime   = time.localtime()
442         dht.close
443         print "Exiting DHT"
444
445 def main():
446         #Input Parameters
447         if(len(sys.argv)<6):
448                 print 'Error define parameter TempParallel
                       TempPerpendicular OFFParallel OFFPerpendicular
                       stayAlive[0|1]'
449                 return 0
450         tempParal=float(sys.argv[1])
451         tempPerp=float(sys.argv[2])
452         offParal=float(sys.argv[3])
453         offPerp=float(sys.argv[4])
454         stayAlive=int(sys.argv[5])
455         if stayAlive!=1 and stayAlive!=0 :
456                 print "Value parametric Error stayAlive[0|1]*"
457                 return 0
458
```

```
459             #Threads
460             threads = []
461             thread1 = myThread(1, "Measure",tempParal,tempParal,IND1,IND2,
                    offParal,stayAlive,"Parallel") #Parallel
462             thread2 = myThread(1, "Measure",tempPerp,tempPerp,IND3,IND4,
                    offPerp,stayAlive,"Perpendicular")
463             #Other measures
464             thread3 = myThread(2, "Execute","XTimeout",1,2,3,4,5,6) #
465             thread4 = myThread(3, "DHT","XTimeout",1,2,3,4,5,6)
466             # Start new Threads
467             thread1.start()
468             thread2.start()
469             thread3.start()
470             thread4.start()
471             # Add threads to thread list
472             threads.append(thread1)
473             threads.append(thread2)
474             threads.append(thread3)
475             threads.append(thread4)
476             #Wait until exit
477             for t in threads:
478                     t.join()
479             print "Exiting Main Thread"
480             return 0
481
482     if __name__ == '__main__':
483         main()
```

C.7.2. algoritmo

```
#!/usr/bin/python
"""
    Increment the average temperature in order to maximize the
       difference between two sensors.
    Decissions are taken depends on two factors:
        *Differential in average between axis in the two rotate
           positions > EDGEDIF
          or *Differential in tendency  between axis in the two rotate
             positions
        *Locate Source condition combination between maximun average in
            sensors
                and increment in average which is measurable between
                   the increase in temperature.
    @ Alejandro Pequeno
    @ Grupo de Neurocomputacion Biologica
    @ Universidad Autonoma de Madrid

   Keyword arguments:
    temperature of Sensor0 --- int variable between [0-100]
    temperature of Sensor0 --- int variable between [0-100]
    temperature of Sensor2 --- int variable between [0-100]
    temperature of Sensor3 --- int variable between [0-100]
    offset of sensor0-1 --------- int variable [0,1,2,3]
    offset of sensor2-3 --------- int variable [0,1,2,3]
    stay Alive Temperature --- int flag indicates continue Temperature
       in pin when finish 0 or 1
"""

import Adafruit_BBIO.ADC as ADC
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.PWM as PWM
import Adafruit_DHT
import time
import os
import sys
import threading
import math
import subprocess

#*****************************SYSTEM PARAMETERS
   ************************************#
SALTOS=20 #Maximun Cycles per Search
SLEEP=1
SAMPLES=108
SAMPLESDHT=10
DEBUG=0
IND1=0
IND2=1
IND3=2
IND4=3
LIMITABOVE=0.5
LIMITBELOW=-0.5
LIMITAVABOVE=1
LIMITAVBELOW=-1
LIMITT=1        #Minimun GAP to consider an incoming plume
HRESTART=2
```

```
50  #Axis Comparative Conditions
51  MAXGIRO=30.0    #Grad FLOAT
52  MINGIRO=5.0     #Grad FLOAT
53  DIFMAX=5.0      #Diference FLOAT
54
55  #Locate Source Conditions
56  TEMPlIMIT=15
57  SOURCEaV=80
58  #
        **********************************************************************************
59  ############DHT PARAM#############
60  DHTSAMPLES=70
61  DHTPIN='P9_41'
62  ##################################
63  # Motor Pins -- (LEFT, RIGHT)
64  senPin = ('P9_38','P9_33','P9_37','P9_39')
65  heatPin = ('P8_13','P8_19','P9_21','P9_22')
66
67  #Threading Locks
68  Sampling= threading.Semaphore(2)
69  Execute= threading.Semaphore(2)
70  Queue= threading.Semaphore(2)
71  ExitLock= threading.Lock()
72
73  #Globals
74  exitFlag=0
75  directionE='A'
76  direction0='B'
77  locateSource=0
78
79  #Modulating Variables
80  NP = 12
81  VAR = 0.98
82
83  #Algorithm Parameters
84  CHVAL= 3
85  KP = 1.8/10
86  KI = 0.54/10
87
88  #Temperature Working Range
89  TMIN=23
90  TMAX=50
91
92  class myThread (threading.Thread):
93          def __init__(self, threadID, name, param1, param2, param3,
                param4, param5, param6, param7):
94                  threading.Thread.__init__(self)
95                  self.threadID = threadID
96                  self.name = name
97                  self.param1 = param1
98                  self.param2 = param2
99                  self.param3 = param3
100                 self.param4 = param4
101                 self.param5 = param5
102                 self.param6 = param6
103                 self.param7 = param7
104         def run(self):
105                 # Get lock to synchronize threads
```

```
106                 #threadLock.acquire()
107                 print "Starting " + self.name
108                 #print_time(self.name, self.counter, 3)
109                 # Free lock to release next thread
110                 if self.name == 'Measure':
111                         Queue.acquire()
112                         sampling(self.param1,self.param2,self.param3,
                               self.param4,self.param5,self.param6,self.
                               param7)
113                 elif self.name == 'Execute':
114                         execute(self.param1)
115                 elif self.name == 'DHT':
116                         sensor = Adafruit_DHT.AM2302
117                         sampleTH(sensor,DHTPIN)
118                 else:
119                         print "Error in self.name = " + self.name
120
121 def setup(start_temp1,start_temp2, ind1, ind2):
122         ADC.setup()
123         PWM.start(heatPin[ind1], start_temp1, 2000, 0)
124         PWM.start(heatPin[ind2], start_temp2, 2000, 0)
125
126 def sampling(temperature1, temperature2, sensor1, sensor2, offset,
       stayAlive, axis):
127         global exitFlag
128         global directionE; global direction0
129         global DifParalel; global DifPerpend
130         global locateSource
131         Hpositive=0; cycles=0
132         lastCrossAxis=0;        lastAvTemp1=0
133         #File
134         nameF="Olus.txt" ;      nameAv= "Average.txt";  nameOf= "Offset
               .txt";   nameDe= "Decission.txt"
135         fileString= str(sensor1)+"-"+str(sensor2)+"_"+nameF ; avString=
                str(sensor1)+"-"+str(sensor2)+"_"+nameAv
136         ofString= str(sensor1)+"-"+str(sensor2)+"_"+nameOf ;  deString=
                str(sensor1)+"-"+str(sensor2)+"_"+nameDe
137
138         countValue1=0;  countValue2=0
139
140         avTemp1=temperature1
141         avTemp2=temperature2
142         setup(avTemp1, avTemp2, sensor1, sensor2)
143         ii=0
144         controlbucle=0;                 rotateDecission=0;
               rotateTendency=0;       rotateCrossAxis=0
145         while ii<NP: #First Cycle for Warm-Up
146                 ii+=1
147                 temperature1 = avTemp1 + round(VAR*math.sin(2*math.pi*
                       ii/NP),2)
148                 temperature2 = avTemp2 + round(VAR*math.sin(2*math.pi*
                       ii/NP),2)
149                 PWM.set_duty_cycle(heatPin[sensor1], temperature1)
150                 PWM.set_duty_cycle(heatPin[sensor2], temperature2)
151                 time.sleep(SLEEP)
152         ExitLock.acquire()
153         while(controlbucle<SALTOS and exitFlag!=1 ):
154         #Control Loop Begining
155                 ExitLock.release()
```

```
156                     Sampling.acquire()
157                     Queue.release()
158                     print "Sampling ",axis
159                     if os.path.exists(fileString):
160                             localtime   = time.localtime()
161                             now = time.strftime("%HH%MM%SS_", localtime)
162                             dst= now+fileString
163                             os.rename(fileString, dst)
164                     if os.path.exists(avString):
165                             localtime   = time.localtime()
166                             now = time.strftime("%HH%MM%SS_", localtime)
167                             dst= now+avString
168                             os.rename(avString, dst)
169                     if os.path.exists(ofString):
170                             localtime   = time.localtime()
171                             now = time.strftime("%HH%MM%SS_", localtime)
172                             dst= now+ofString
173                             os.rename(ofString, dst)
174                     if os.path.exists(deString):
175                             localtime   = time.localtime()
176                             now = time.strftime("%HH%MM%SS_", localtime)
177                             dst= now+deString
178                             os.rename(deString, dst)
179                     # Open a file
180                     fo = open(fileString, "w+")
181                     count=0 ;       flagState= 1 ;
182                     incrementI=0;   differentialTemp=1;     decissionTemp=0
183                     av1Total=0;     av2Total=0
184                     if cycles%HRESTART==0:
185                             Hpositive=0 #Independent Decission
186                             flagState=0 #Unforced Hipothesis
187                             rotateDecission=0
188                             rotateTendency=0
189                             rotateCrossAxis=0
190                     x=[]
191                     concent1=[];    concent2=[]
192                     temp1=[];       temp2=[]
193                     avList1=[];     avList2=[];     avTempList1=[];
                           avTempList2=[]; hypoList=[]
194                     conList1=[];    conList2=[];
195                     avTemp1Sta=avTemp1
196                     avTemp2Sta=avTemp2
197                     #Measure Loop Begining
198                     while count<SAMPLES :
199                             #Measure
200                             temperature1 = avTemp1 + round(VAR*math.sin(2*
                                   math.pi*count/NP),2)
201                             temperature2 = avTemp2 + round(VAR*math.sin(2*
                                   math.pi*count/NP),2)
202                             PWM.set_duty_cycle(heatPin[sensor1],
                                   temperature1)
203                             PWM.set_duty_cycle(heatPin[sensor2],
                                   temperature2)
204                             time.sleep(SLEEP)
205                             value1=ADC.read_raw(senPin[sensor1])
206                             value2=(ADC.read_raw(senPin[sensor2])+offset)
207                             x.append(count)
208                             concent1.append(value1)
209                             concent2.append(value2)
```

```
210                             temp1.append(temperature1)
211                             temp2.append(temperature2)
212                             #
                                    ############################################################
213                             #Average Values
214                             conList1.append(value1)
215                             conList2.append(value2)
216                             average1=0; average2=0
217                             #Adding concentrate to calculate average
218                             for value1,value2 in zip(conList1,conList2) :
219                                     average1+=value1
220                                     average2+=value2
221
222                             #Loop iteracion
223                             count+=1
224
225                             #Calculate average
226                             if (count)<NP :
227                                     avList1.append(round(average1/(count)
                                            ,2))
228                                     avList2.append(round(average2/(count)
                                            ,2))
229                                     avTempList1.append(avTemp1)
230                                     avTempList2.append(avTemp2)
231                                     hypoList.append(Hpositive)
232                             else :
233                                     avList1.append(round(average1/NP,2))
234                                     avList2.append(round(average2/NP,2))
235                                     avTempList1.append(avTemp1)
236                                     avTempList2.append(avTemp2)
237                                     hypoList.append(Hpositive)
238                                     del conList1[0];
239                                     del conList2[0];
240                             #Export Data
241                             wline=str(count-1)+'    '+str(value1)+' '+str(
                                    value2)+' '+str(temperature1)+'   '+str(
                                    temperature2)+'\n'
242                             fo.writelines(wline)
243                             if DEBUG:
244                                     print 'Muestra[',count-1,']     Value:
                                            ',value1,'----',value2,'
                                            Temperature: ',temperature1,'----',
                                            temperature2
245                                     sys.stdout.flush()
246                             #
                                    ############################################################
247                             #Adjust Temperature
248                             if count%NP==0:
249                                     difference=avList1[-1] - avList2[-1]
250                                     #Sign of the Difference
251                                     controlParamP=0
252                                     outrange=0
253                                     for index in range(0,CHVAL):
254                                             if (index*NP+1)>len(avList1):
255                                                     outrange=1
256                                                     break
```

```
257                                     elif (avList1[-1-int(index*NP)]
                                            - avList2[-1-int(index*NP)
                                        ])>0:
258                                             controlParamP+=1
259                                 #Consecutive Differences
260                                 if controlParamP==CHVAL:
261                                     Hpositive=1
262                                 elif controlParamP==0 :
263                                     Hpositive=-1
264                                 elif flagState==0:      #Initial Param
                                        State
265                                     if (difference) > 0:
266                                             Hpositive=1
267                                     else:
268                                             Hpositive=-1
269                                     flagState=1
270                                     offset+=difference
271                                     print sensor1,'-',sensor2,"
                                            Nuevo Offset: ",offset
272                                     sys.stdout.flush()
273                                 #Difference out of range
274                                 if outrange==1:
275                                     #Sign of the increment
                                            differential
276                                     if (avList1[-1]-avList1[0])-(
                                            avList2[-1]-avList2[0])>0:
277                                             Dpositive=1
278                                     else:
279                                             Dpositive=-1
280                                     #Differential Increments
281                                     incrementP=(avList1[-1]-avList1
                                            [0])-(avList2[-1]-avList2
                                            [0])
282                                     incrementI+=incrementP
283                                 else:
284                                     #Sign of the increment
                                            differential
285                                     if (avList1[-1]-avList1[-NP-1])
                                            -(avList2[-1]-avList2[-NP
                                            -1])>0:
286                                             Dpositive=1
287                                     else:
288                                             Dpositive=-1
289                                     #Differential Increments
290                                     incrementP=(avList1[-1]-avList1
                                            [-NP-1])-(avList2[-1]-
                                            avList2[-NP-1])
291                                     incrementI+=incrementP
292                                 #Sign of the past increment
293                                 actualSign=1
294                                 actualSign=math.copysign(actualSign,
                                        differentialTemp)
295                                 differentialTemp=(Hpositive*Dpositive*
                                        actualSign)*abs(round(incrementP*KP-
                                        incrementI*KI,1))
296                                 avTemp1+= differentialTemp
297                                 avTemp2+= differentialTemp
298
299                                 #Deccision
```

```
300                                 if differentialTemp > LIMITT :
301                                         decissionTemp= difference
302                                 #Keep between the working range
303                                 if(avTemp1>TMAX-VAR):
304                                                 avTemp1=TMAX-VAR
305                                 elif(avTemp1<TMIN+VAR):
306                                                 avTemp1=TMIN+VAR
307                                 if(avTemp2>TMAX-VAR):
308                                                 avTemp2=TMAX-VAR
309                                 elif(avTemp2<TMIN+VAR):
310                                                 avTemp2=TMIN+VAR
311                                 if DEBUG:
312                                         print 'Sign: SensorLead',
                                                    Hpositive, '   DifIncrements
                                                    ',Dpositive,'
                                                    ActualIncrease',actualSign
313                                         sys.stdout.flush()
314                                 print 'Sensores ',sensor1,'-',sensor2,'
                                        Average:                ',avList1
                                       [-1],'----',avList2[-1],'
                                       DifIncrements ',incrementP,'
                                       SumDifIncrements ',incrementI
315                                 sys.stdout.flush()
316                                 print 'Sensores ',sensor1,'-',sensor2,'
                                        Temperature:            ','
                                       ChanAvTemp ',differentialTemp,'
                                             Average ',avTemp1,'     ',
                                       avTemp2,'\n'
317                                 sys.stdout.flush()
318                         #
                                ##################################################################
319                         #Take Decission
320                         PWM.set_duty_cycle(heatPin[sensor1], avTemp1)
321                         PWM.set_duty_cycle(heatPin[sensor2], avTemp2)
322
323                         #Measure Loop End
324                         fo.close()
325                         #Averages File
326                         elements=len(avList1)
327                         min1=min(avList1[NP:len(avList1)])
328                         min2=min(avList2[NP:len(avList2)])
329                         max1=max(avList1[NP:len(avList1)])
330                         max2=max(avList2[NP:len(avList2)])
331                         decissionAv=abs(max1-min1)-abs(max2-min2) # + => F    -
                               => B
332                         avo= open(avString, "w+")
333                         for index in range(len(avList1)) :
334                                 av1Total+=avList1[0]
335                                 av2Total+=avList2[0]
336                                 av1Last=avList1[0]
337                                 av2Last=avList2[0]
338                                 wline=str(index)+'      '+str(avList1.pop(0))+'
                                        '+str(avList2.pop(0))+' '+str(avTempList1.
                                       pop(0))+'     '+str(avTempList2.pop(0))+'
                                            '+str(hypoList.pop(0))+'\n'
339                                 avo.writelines(wline)
340                         avo.close()
341                         #Offset File
```

```
342                     avTemp1End=avTemp1
343                     avTemp2End=avTemp2
344                     decissionDiff=(av1Total-av2Total)/elements
345                     crossAxis=av1Total/elements
346                     ofo = open(ofString, "w+")
347                     wline=str(sensor1)+'    '+str(sensor2)+'        '+'
                           StartAv'+'   '+str(avTemp1Sta)+'     '+str(
                           avTemp2Sta)+'     '+'FinAv'+'     '+str(avTemp1End)+
                           '     '+str(avTemp2End)+'\n'
348                     wline2='Offset '+'Ini '+str(offset)+'   Calc '+str(
                           decissionDiff)+'\n'
349                     ofo.writelines(wline)
350                     ofo.writelines(wline2)
351                     ofo.close()
352                     #Decission File
353                     #* Base on Total Average and difference between max -
                           min in average and Prioriry in Plume incoming(
                           Measure by the gap in temperature)
354                          #DecissionDiff
355                     if round(decissionDiff,2)>LIMITABOVE:
356                             decDiff=1#F
357                     elif round(decissionDiff,2)<LIMITBELOW:
358                             decDiff=-1#B
359                     else:
360                             decDiff=0#X
361
362                             #DecissionAv
363                     if round(decissionAv,2)>LIMITAVABOVE:
364                             decAv=1#F
365                     elif round(decissionAv,2)<LIMITAVBELOW:
366                             decAv=-1#B
367                     else:
368                             decAv=0#X
369
370                     #Construct Decission
371                     avLast=av1Last-av2Last
372                     tendency= avLast-decissionDiff
373                     wline= "Difference      Average         avLast
                                Tendency\n"
374                     wline2= str(decissionDiff)+'    '+str(decissionAv)+'
                            '+str(avLast)+' '+str(tendency)+'\n'
375                     print 'Axis ',axis,'    ',wline2
376
377                     #Decission by Rotate each HRESTART cycles
378                     if cycles%HRESTART==0:
379                             decissionDiff=-1*decissionDiff
380                             tendency=-1*tendency
381                     rotateDecission+=decissionDiff
382                     rotateTendency+=tendency
383                     rotateCrossAxis+=crossAxis
384                     rotateCrossAxis=rotateCrossAxis/HRESTART
385                     #Move Decission before Restart parameters
386                     if rotateDecission<LIMITABOVE and rotateDecission>
                           LIMITBELOW:
387                          uncertainty=1
388                     else:
389                             uncertainty=0
390                     if (cycles+1)%HRESTART==0:
391                             if axis=='Parallel':
```

```
392                                 DifParalel=abs(rotateDecission)
393                                 if rotateDecission>LIMITABOVE or (
                                       uncertainty and round(rotateTendency
                                       )>0):
394                                         direction='F'   #Forward
395                                         directionE='F'
396                                 elif rotateDecission<LIMITBELOW or (
                                       uncertainty and round(rotateTendency
                                       )<0):
397                                         direction='B'   #Backward
398                                         directionE='B'
399                                 else:
400                                         direction='X'   #Don't Know
401                                         directionE='X'
402                         elif axis=='Perpendicular':
403                                 DifPerpend=abs(rotateDecission)
404                                 if rotateDecission>LIMITABOVE or (
                                       uncertainty and round(rotateTendency
                                       )>0):
405                                         direction='E'   #Forward
406                                         directionO='E'
407                                 elif rotateDecission<LIMITBELOW or (
                                       uncertainty and round(rotateTendency
                                       )<0):
408                                         direction='W'   #Backward
409                                         directionO='W'
410                                 else:
411                                         direction='X'   #Don't Know
412                                         directionO='X'
413                         else:
414                                 direction='U'           #Undefine
415                         #locateSource
416                 wline3=axis+'Decission by Rotation:  '+str(
                       rotateDecission)+'           Tendency:  '+str(
                       rotateTendency)+'\n'+'Av Lead: '+str(rotateCrossAxis
                       )+'        Temperature: '+str(avTemp1)+'\n'
417                 if (cycles+1)%HRESTART==0:
418                         if lastCrossAxis!=0 and lastAvTemp1!=0 and (abs
                               (rotateCrossAxis-lastCrossAxis) > SOURCEaV
                               or abs(avTemp1-lastAvTemp1) > TEMPlIMIT):
419                                 print "Source Locate in ",direction
420                                 print "cross:   ",crossAxis,"Lascross:
                                         ",lastCrossAxis,"temp:  ",avTemp1,
                                       "lasTemp:     ",lastAvTemp1
421                                 ExitLock.acquire()
422                                 locateSource=1
423                                 ExitLock.release()
424                         lastCrossAxis=rotateCrossAxis
425                         lastAvTemp1=avTemp1
426                         print wline3
427                 defo = open(deString, "w+")
428                 defo.writelines(wline)
429                 defo.writelines(wline2)
430                 defo.writelines(wline3)
431                 defo.close()
432                 #Semaphores
433                 Sampling.release()
434                 Execute.acquire()
435                 Execute.release()
```

```
436                     controlbucle+=1
437                     cycles+=1
438                     ExitLock.acquire()
439             ExitLock.release()
440             #Control Loop End
441             Sampling.release()
442             print "Finish ",axis
443             ExitLock.acquire()
444             exitFlag=1
445             ExitLock.release()
446             if not stayAlive :
447                     PWM.cleanup()
448                     print "Cleaning Up PWM"
449
450     def execute(param1):
451             global exitFlag
452             global directionE; global direction0
453             global DifParalel; global DifPerpend
454             global locateSource
455             Queue.acquire()
456             Queue.acquire()
457             cycles=0
458             alternate=0
459             rotate=0
460             print "Execute ",param1
461             ExitLock.acquire()
462             while not exitFlag:
463                     ExitLock.release()
464
465                     cycles+=1
466
467                     #Execute Lock Both Measures
468                     Execute.acquire()
469                     Execute.acquire()
470                     #Execute Wait Final Measure
471                     Sampling.acquire()
472                     Sampling.acquire()
473                     #Moves robot due to decissions global param direction
                           protected by semaphore
474                     print "Moving Robot ",directionE,direction0," SHAKE IT
                           UP"
475                     sys.stdout.flush()
476                     #Preprocess
477                     if cycles%HRESTART==0 and cycles != SALTOS:
478                             #X axis
479                             if round(DifParalel)!=0:
480                                     normaliza=round(DifParalel)
481                             else:
482                                     normaliza=DIFMAX
483                             #Factor Axis
484                             factorAxis=(DifParalel/normaliza - DifPerpend/
                                   normaliza)
485                             if factorAxis>=0:
486                                     porcAxis=1-min(factorAxis,1)
487                             else:
488                                     porcAxis=1
489
490                             compareAxis=round(porcAxis*(MAXGIRO-MINGIRO)+
                                   MINGIRO)
```

```
491
492                         print 'Giro de Ejes: ',compareAxis,'    Porc: '
                                , porcAxis*100
493                 #
                        ###################################################################
                            ###

494                         if directionE=='F':
495                         #Y axis
496                                 if direction0=='E':
497                                         compareAxis=-1*compareAxis
498                                         subprocess.call(["python", "
                                            turnRobot.py", str(
                                            compareAxis)])
499                                 elif direction0=='W':
500                                         subprocess.call(["python", "
                                            turnRobot.py", str(
                                            compareAxis)])
501                                 subprocess.call(["python", "moveRobot.
                                    py", "200"])
502                         elif directionE=='B':
503                                 #Y axis
504                                 if direction0=='E':
505                                         subprocess.call(["python", "
                                            turnRobot.py", str(
                                            compareAxis)])
506                                 elif direction0=='W':
507                                         compareAxis=-1*compareAxis
508                                         subprocess.call(["python", "
                                            turnRobot.py", str(
                                            compareAxis)])
509                                 subprocess.call(["python", "moveRobot.
                                    py", "-200"])
510                         elif directionE=='X':
511                                 #Y axis
512                                 print "X Tendencia ",direction0
513                 #
                        ###################################################################
                            ###

514                 elif cycles%HRESTART!=0 and cycles != SALTOS:
515                         rotate+=1
516                         if rotate%HRESTART==0:
517                                 subprocess.call(["python", "turnRobot.
                                    py", "-180"])
518                         else:
519                                 subprocess.call(["python", "turnRobot.
                                    py", "180"])
520                 subprocess.call(["gnuplot", "plot_hyp_short.gp"])
521                 subprocess.call(["bash", "scriptNombreParam", "
                        AlgorithmF"])
522                 if locateSource:
523                         exitFlag=1
524                 #Continue Measure
525                 Execute.release()
526                 Execute.release()
527                 Sampling.release()
528                 Sampling.release()
529                 time.sleep(10)
530
531                 ExitLock.acquire()
```

```
532         ExitLock.release()
533         print "Exiting ",param1
534 
535 def sampleTH(sensor,pin):
536         global exitFlag
537         ite=0
538         nameDHT="DHT.txt" ;
539         fileString= nameDHT ;
540         if os.path.exists(fileString):
541                 localtime   = time.localtime()
542                 now = time.strftime("%HH%MM%SS_", localtime)
543                 dst= now+fileString
544                 os.rename(fileString, dst)
545 
546         dht= open(fileString, "w+")
547         wline='Iteracion'+'     '+'Temperatura(C)'+'    '+'Humedad(%)'+
                '\n'
548         dht.writelines(wline)
549         localtime   = time.localtime()
550         if not stayAlive :
551                 PWM.cleanup()
552                 print "Cleaning Up PWM"
553 
554         #Sampling Humidity-Temperature
555         ExitLock.acquire()
556         while ite<SAMPLESDHT and exitFlag==0:
557                 ExitLock.release()
558                 humidity, temperature = Adafruit_DHT.read_retry(sensor,
                        pin)
559                 if humidity is not None and temperature is not None:
560                         if DEBUG:
561                                 print 'Temp={0:0.1f}*C Humidity={1:0.1f
                                    }%'.format(temperature, humidity)
562                                 sys.stdout.flush()
563                         temp=   round(temperature,1)
564                         hum=    round(humidity,1)
565                         wline=str(ite)+'        '+str(temp)+'   '+str(
                                hum)+'\n'
566                 else:
567                         print 'Failed to get reading. Try again!'
568                         wline="MACK"
569                 dht.writelines(wline)
570                 ite+=1
571                 ExitLock.acquire()
572         ExitLock.release()
573         dht.close
574         print "Exiting DHT"
575 
576 
577 def main():
578         #Input Parameters
579         if(len(sys.argv)<8):
580                 print 'Error define parameter TempParallel0
                        TempParallel1 TempPerpendicular2 TempPerpendicular3
                        OFFParallel OFFPerpendicular stayAlive[0|1]'
581                 return 0
582         tempPara10=float(sys.argv[1])
583         tempPara11=float(sys.argv[2])
584         tempPerp2=float(sys.argv[3])
```

```
585           tempPerp3=float(sys.argv[4])
586           offParal=float(sys.argv[5])
587           offPerp=float(sys.argv[6])
588           stayAlive=int(sys.argv[7])
589           if stayAlive!=1 and stayAlive!=0 :
590                   print "Value parametric Error stayAlive[0|1]*"
591                   return 0
592
593           #Threads
594           threads = []
595           thread1 = myThread(1, "Measure",tempParal0,tempParal1,IND1,IND2
                 ,offParal,stayAlive,"Parallel") #Parallel
596           thread2 = myThread(1, "Measure",tempPerp2,tempPerp3,IND3,IND4,
                 offPerp,stayAlive,"Perpendicular") #Perpendicular
597           thread3 = myThread(2, "Execute","XTimeout",1,2,3,4,5,6) #
598           thread4 = myThread(3, "DHT","XTimeout",1,2,3,4,5,6)
599           # Start new Threads
600           thread1.start()
601           thread2.start()
602           thread3.start()
603           thread4.start()
604           # Add threads to thread list
605           threads.append(thread1)
606           threads.append(thread2)
607           threads.append(thread3)
608           threads.append(thread4)
609           #Wait until exit
610           for t in threads:
611                   t.join()
612           print "Exiting Main Thread"
613           return 0
614
615   if __name__ == '__main__':
616       main()
```

D
Códigos SCAD(Diseño 3D)

D.0.3. Carcasa sensores v2

```
#Parte de abajo
union(){
        difference(){

                cube([49,31,9]);
                translate([3,3,2]){
                        cube([43,25,10]);
                }
        }
        translate([3,3,2]){
                        translate([9,14,0]){
                                cylinder(h=5,r=1.5,$fn=100);
                        }
                        translate([33,14,0]){
                                cylinder(h=5,r=1.5,$fn=100);
                        }
        }
}
# Parte de arriba
difference(){
        cube([51,31,9]);
        translate([3,3,2]){
                cube([43,25,10]);
                translate([6.5,10,-5]){
                        cylinder(h=90,r=5.5,$fn=100);
                }
        }
        translate([-1,18,2]){
                cube([40,10,10]);
        }
        translate([40,3,2]){
                cube([40,25,10]);
        }
}
```

D.0.4. Rueda: Eje y tapa

```
#Eje
union(){
        difference(){
                cylinder(h=10,r=7,$fn=100);
                translate([-1.5,-3,-1]){
                        cube([3,6,10]);

                }
        }
        translate([0,0,10]){
                cylinder(h=20,r=3,$fn=100);
        }
}
#Tapa
difference(){
        cylinder(h=10,r=8,$fn=100);
        translate([0,0,-1]){
                cylinder(h=5,r=3,$fn=100);
        }
}
```

D.0.5. Soporte para batería

```
difference() {
        cube([69,64,8]);
        translate([3,3,3]){
                cube([63,58,6]);
        }
        translate([24.5,32,-1]){
                cylinder(h=50,r=2,$fn=100);
        }
        translate([44.5,32,-1]){
                cylinder(h=50,r=2,$fn=100);
        }
}
```

E

Presupuesto

1)	**Ejecución Material**	
	▪ Compra de ordenador personal	1.000 €
	▪ Material de oficina	100 €
	▪ Componentes robot	150 €
	▪ Componentes narices electrónicas	60 €
	▪ Alquiler de impresora 3D	260 €
	▪ Total de ejecución material	**1.570** €
2)	**Gastos generales**	
	▪ 16 % sobre Ejecución Material	251,2 €
3)	**Beneficio Industrial**	
	▪ 6 % sobre Ejecución Material	94,2 €
4)	**Honorarios Proyecto**	
	▪ 1800 horas a 15 €/ hora	27000 €
5)	**Material fungible**	
	▪ Gastos de impresión y encuadernación	110 €
6)	**Subtotal del presupuesto**	
	▪ Subtotal Presupuesto	29.025,4 €
7)	**I.V.A. aplicable**	
	▪ 21 % Subtotal Presupuesto	6.095,3 €
8)	**Total presupuesto**	
	▪ Total Presupuesto	35.120,7 €

Madrid, Mayo 2015
El Ingeniero Jefe de Proyecto

Fdo.: Alejandro Pequeño Zurro
Ingeniero de Telecomunicación

F

Pliego de condiciones

Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un *Uso de una nariz electrónica ultra-portátil en robots para la detección de fuentes de odorantes*. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.
Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:
Condiciones generales.

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contrataz anteriormente llamado "Presupuesto de Ejecución Material"que hoy designa otro concepto.

Condiciones particulares.

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.