Universidad Autónoma de Madrid
Escuela Politécnica Superior

Proyecto fin de carrera

INTEGRACIÓN DE UNA NARIZ ELECTRÓNICA ULTRA-PORTÁTIL EN UN ROBOT MODULAR PARA EL CONTROL DE SU MOVIMIENTO A TRAVÉS DE LOS ODORANTES RECIBIDOS

Ingeniería de Telecomunicación

Tomás Vázquez Rubio
Abril 2013

INTEGRACIÓN DE UNA NARIZ ELECTRÓNICA ULTRA-PORTÁTIL EN UN ROBOT MODULAR PARA EL CONTROL DE SU MOVIMIENTO A TRAVÉS DE LOS ODORANTES RECIBIDOS

AUTOR: Tomás Vázquez Rubio
TUTOR: Francisco de Borja Rodríguez Ortiz

GNB
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Abril 2013

Resumen

Resumen

Este proyecto fin de carrera muestra la obtención y el análisis de la locomoción en un robot modular de 8 módulos robotizados empleando Generadores Centrales de Patrones (CPGs). Los CPGs son redes neuronales que emplean los seres vivos para producir actividad motora que implique cierta ritmicidad, tal como puede ser el caminar en el ser humano. A la hora de buscar un diseño bio-inspirado para la locomoción de robots, puede ser muy útil emplear CPGs simulados, pues éstos aportan al sistema robustez, autonomía y recuperación ante posibles errores. El modelo de neurona empleado en este proyecto vendrá definido por una serie de mapas iterados, y la interconexión entre las neuronas tendrá una serie de reglas tales como la localidad (una neurona sólo puede conectarse con las de su entorno) y la competición (conexiones inhibidoras). El modelo sináptico que conecte a las neuronas se tratará de un modelo eléctrico, siendo más sencillo que el modelo utilizado en proyectos anteriores (modelo químico). Este nuevo modelo es mucho más instantáneo y, por tanto, hace que el CPG sea más ágil.

Los modelos de CPG diseñados se integrarán en el propio robot. Al utilizarse un modelo de sinapsis eléctrico se podrá generar la locomoción en un sencillo microcontrolador (con la sinapsis química era necesario simular el CPG previamente en un PC). Se deberá buscar un microcontrolador que permita la simulación del CPG de manera fluida y que facilite la tarea de controlar los servos del robot y de establecer la comunicación con la Nariz Electrónica. Gracias a esto se realizará un robot completamente autónomo, sin la necesidad de estar próximo a un ordenador.

La Nariz Electrónica se encargará de medir la presencia de odorante que exista en el ambiente y transmitirá esta información al microcontrolador encargado de controlar al robot. Esta comunicación se hará mediante el protocolo I2C, protocolo sencillo, fiable y muy útil para conectar microcontroladores entre sí. Habrá que definir la manera de establecer este protocolo entre los dos dispositivos, siendo necesario reprogramar la Nariz Electrónica para introducir los comandos necesarios mediante los cuales podrá recibir y transmitir información mediante I2C.

Una vez se consiga establecer comunicación entre los dos dispositivos, se realizará una prueba de concepto de la integración sensorial en el CPG. Se diseñará un sencillo algoritmo mediante el que el robot responderá ante estímulos olfativos. Éste será el primer paso dentro de una serie de proyectos del departamento que pretenden, a largo plazo, obtener la localización de la fuente de odorante.

Palabras Clave

Redes neuronales, CPG, Bioinspiración, Robot modular, Microcontroladores, Arduino, Nariz Electrónica, SkyPic, SkyMega

Abstract

This dissertation shows the obtaining and analysis of the locomotion in a modular robot of eight robotic modules using Central Pattern Generators(CPGs). Furthermore, it shows, from previous projects of the department, the introduction of a sensor in the robot. By the use of the Electronic Nose it is pretended to change robot's behaviour depending on the amount of odorant present in the environment. CPGs are neural networks that living beings use to generate motor activity, which is involved in motion that requires periodicity, for instance, human being locomotion. The use of simulated CPGs in the context of bio-inspired design for robot locomotion is highly appropriate because CPGs are found to contribute to system robustness, autonomy and stability, in the sense of error recovery capability. In the present project, the neuron model used is defined by a set of iterated equations, and interconnection among neurons is determined by a set of rules, for instance, the principle of locality, which states that a neuron only can be connected to neighbour neurons. The synaptic model that connects neurons will be an electrical model, being more simple than the model used in previous projects (chemical model). This new model is much more instant and therefore makes CPG more quick.

The CPG models designed will be integrated in the robot itself. The use of an electrical synaptic model will allow to generate locomotion in the same microcontroller that controls the robot (with the chemical synapsis the CPG had to be simulated in a PC). A search for a microcontroller which holds the simulation of the CPG decently must be done. Also this microcontroller should have the possibility to manage the robot servos and to establish connection with de Electronic Nose. This will make the robot totally autonomous, without the necessity of having a PC in the surroundings.

The Electrical Nose will have the mission to measure the presence of odorant in the surroundings and will transmit this information to the microcontroller which drives the robot. This communication will take place using I2C protocol, a simple, reliable and useful protocol to connect microcontrollers between them. It will have to be defined the way to establish this protocol between both devices, having the need to reprogram the electronic nose to indroduce the necessary commands to receive and transmit information through the I2C protocol.

Once communication is established between devices, a concept proof of the sensorial CPG integration will be done. A simple algorithm will be designed to make the robot change its behaviour because of olfactory stimulus. This will be the first step of a much bigger project of the GNB that will try, in a long term, to obtain the localization of the odorant origin source.

Key words

Neural Networks, CPG, Bioinspiration, Modular Robot, Microcontrollers, Arduino, Electronic Nose, SkyPic, SkyMega

Agradecimientos

Me gustaría agradecer a los miembros del GNB por la ayuda que me han prestado a lo largo de estos meses. Quiero dar las gracias especialmente a David y a Fernando por las innumerables conversaciones que hemos tenido y por todos los conocimientos que han sido capaces de traspasar a mi cabeza. Sin ellos y sin Sara las horas que he invertido en el laboratorio hubieran sido mucho más aburridas. Y por supuesto quiero agradecer a Paco por permitirme vivir durante unos años la vida que rodea a la investigación, involucrándome en el departamento y aportándome una gran cantidad de conocimientos científicos, aparte de permitirme ver el lado humano de la enseñanza en la universidad.

Se acaba una etapa en la que he conocido gente increíble. Gracias a toda la gente que he conocido a lo largo de la carrera y que han permitido que nuestra amistad creciese. Comencé esta aventura de la mano de Sergi, Carlos y Vicen; al poco tiempo se unieron Pablo, Arturo, Antonio, Low, Ángel; y por último llegaron Katas, Tomás, Corri y Dani, aparte de Chemita, Burgués, Campesino, Rober, Henar, Javi, Luis... Son tantos los buenos amigos que he podido hacer a lo largo de estos 8 años que simplemente no caben aquí. Gracias por hacer de la dura vida de un estudiante de teleco un poco más feliz.

El que se acabe esta etapa solo significa que otra empieza. La vida de trabajador sólo acaba de empezar pero ya he encontrado un grupo de trabajo increíble. Espero (como todos vosotros) que podamos compartir mucho más de lo que hemos podido hasta el momento, es decir, que todos nosotros sigamos trabajando allí a partir de Julio. Son solo 6 meses pero ya hay muchísimas personas a las que he cogido aprecio así que, de manera más simple, agradezco en bloque a los becarios de la linea de negocio Móvil, a los de la Fija y a nuestros especialistas secuaces de Tasio.

Quiero agradecer a mis hermanos todos los buenos momentos que hemos vivido a lo largo de todos estos años de dura convivencia. Las peleas se acabaron ya hace años y desde hace unos cuantos ya disfrutamos unos de los otros (aunque a veces sea duro aguantarse entre nosotros). Gracias a mi padre por todo y sobretodo por aportarme mi vena de ingeniero, además sin su ayuda los primeros años de carrera hubieran sido mucho más duros. Muchas gracias a toda mi familia por haberme mostrado siempre tanto cariño. Gracias a los amigos de toda la vida. Ángel, Mani y Alex, muchas gracias por los infinitos momentos que hemos llegado a vivir. Una vez entregue este proyecto espero que volvamos a disfrutar de aventuras inolvidables. También gracias a amigos, que aunque ya no tenga mucha amistad con ellos, me han hecho ser como soy: Felipe, David, Álvaro, Manu, Quinti, Maria, Nerea... Gracias a todos los que han compartido conmigo algún momento de vuestras vidas, como son todas las personas que conocí en el camino, con las que en un mes llegamos a vivir momentos indescriptibles.

Y por último tengo que agradecer a mi tres mujeres todo el cariño y ... TODO que me han dado a lo largo de esta vida. A mi madre por cuidarme siempre, por sentarse conmigo en la cocina a estudiar hasta que fui capaz de hacerlo solo, por despertarme la inquietud de tener siempre un libro cerca, en fin, por hacerme como soy. Gracias a Cris por todo el amor que me ha demostrado en estos casi 4 años. Hemos vivido buenos y no tan buenos momentos, pero seguro que los mejores están por llegar. No me imagino un futuro sin ti. A la Bueli le debo todo lo que soy: Gracias.

Índice general

Índice de figuras **x**

Índice de cuadros **xv**

1. Introducción **1**

2. Objetivos **7**

3. Estado del arte **9**

3.1. Bioinspiración . . . 9

3.2. Modelos dinámicos . . . 10

3.3. La neurona . . . 10

3.4. La sinapsis . . . 12

3.4.1. Clasificación de la sinapsis atendiendo a su naturaleza . . . 13

3.4.2. Clasificación de la sinapsis atendiendo al efecto generado . . . 14

3.5. Red neuronal artificial . . . 14

3.5.1. Clasificación de redes neuronales en función de su grado de inspiración biológica . . . 15

3.6. Robótica modular . . . 16

3.7. Locomoción en robots mediante CPGs . . . 17

3.7.1. Propiedades de los CPGs . . . 18

3.7.2. Metodología de diseño para CPGs . . . 18

3.7.3. Feedback en CPGs . . . 19

3.8. Reglas básicas en la construcción del CPG . . . 20

3.8.1. Principio de localidad . . . 20

3.8.2. Escalas temporales . . . 20

3.8.3. Competencia: Winnerless Competition . . . 20

3.9. Narices electrónicas . . . 21

3.9.1. Sensores electrónicos . . . 21

3.9.2. Nariz electrónica artificial . . . 21

3.9.3. Robots móviles con capacidad olfativa . . . 23

3.10. Microcontroladores . . . 23

3.11. Comunicación I2C . . . 25

4. Diseño **27**

4.1. Introducción del robot 27
4.2. Modelo neuronal de Rulkov 33
4.3. Conexión entre neuronas: Modelo sináptico 36
4.4. Del código neuronal a las señales motoras 38
4.5. Diseño del CPG 40
4.5.1. Patrones de conectividad 40
4.5.2. Evaluación de los modelos propuestos 44
4.5.3. Topología escogida 45
4.6. Adquisición de olores 46
4.6.1. Introducción a Olus2 46
4.6.2. Sistema funcional de Olus2 47
4.6.3. Funcionamiento de los componentes de Olus2 48
4.6.4. Principales características de Olus2 53
4.7. I2C 54
4.7.1. Protocolo de Comunicación del bus I2C 54
4.7.2. Formato de transacción 57

5. Desarrollo e Integración **59**

5.1. Introducción 59
5.2. Modelo neuronal de Rulkov: Simulación de una neurona 59
5.2.1. De las ecuaciones al modelo neuronal desarrollado 60
5.2.2. Comportamiento del modelo neuronal 61
5.3. Sinapsis de Rulkov: Conexión de las neuronas 63
5.3.1. De las ecuaciones al modelo sináptico desarrollado 63
5.3.2. Comportamiento del modelo sináptico 63
5.4. Modelo de la motoneurona: Generación del movimiento 66
5.4.1. De las ecuaciones al modelo de motoneurona desarrollado 66
5.4.2. Comportamiento del modelo de la motoneurona 67
5.5. Construcción del CPG completo 70
5.5.1. Inhibición Simétrica entre módulos 70
5.5.2. Cálculo del desfase entre módulos 73
5.5.3. Inhibición Asimétrica entre módulos 75
5.5.4. Simulación del comportamiento del robot 82
5.6. Integración del CPG en el robot 84
5.6.1. Introducción del CPG en el microcontrolador: SkyPIC 84
5.6.2. Búsqueda de soluciones ante la falta de espacio: Arduino 87
5.6.3. Placa controladora definitiva: SkyMEGA 92

5.7. Comunicación del Robot con la Nariz Electrónica 97
5.7.1. Programación de la Nariz Electrónica 98
5.7.2. Desarrollo del módulo I2C en la Nariz Electrónica 99
5.7.3. Comunicación entre el Arduino y la Nariz Electrónica 105
5.8. Integración de todos los elementos 108
5.8.1. Establecimiento del patrón de búsqueda 109
5.8.2. Mejora del algoritmo de búsqueda 112

6. Resumen de objetivos **115**

7. Conclusiones **117**

8. Trabajo futuro **119**

Glosario de acrónimos **121**

Bibliografía **122**

A. Documentación técnica de la comunicación **129**

B. Manual del programador **133**
B.1. Código utilizado para desarrollar el CPG 133
B.2. Código utilizado para integrar el CPG en el simulador RoboSim 142
B.3. Código utilizado para desarrollar la comunicación I2C con la que comunicar Robot y Nariz 145

C. Presupuesto **163**

D. Pliego de condiciones **165**

Índice de figuras

1.1. Juan González con su robot, Cube Revolutions 2
1.2. Robot controlado mediante un CPG 2
1.3. Idealización de una Nariz Electrónica 3
1.4. Sistema con feedback 4

3.1. Estructura de una neurona 11
3.2. Variación en el potencial de membrana 11
3.3. Elementos implicados en la sinapsis 13
3.4. Estructura sináptica de una red neuronal 13
3.5. Ejemplo de robots modulares 16
3.6. Robot que mediante el uso de CPGs imita el movimiento de una lamprea 17
3.7. Experimento con un CPG perteneciente al sistema digestivo de una langosta . . 19
3.8. Procesos llevados a cabo por una Nariz Electrónica comparados con los procesos que realiza el sistema olfativo biológico. 22
3.9. Placa Arduino de desarrollo de hardware libre 24
3.10. Estructura del bus I2C (No incluye ni alimentación ni masa de los dispositivos) . 25

4.1. Descripción Cube Revolutions 27
4.2. Configuración Cabeceo-Cabeceo 28
4.3. Configuración Cabeceo-Viraje 28
4.4. Métodos de interconexión de los módulos 29
4.6. Distribución de señales del control locomotor 30
4.5. Matrices de interconexión de los motores 30
4.7. Distribución de pines en el bus 31
4.8. Movimiento del robot en línea recta 32
4.9. Movimiento del robot en una trayectoria circular 33
4.10. Modos de funcionamiento del modelo neuronal 34
4.11. Regiones de funcionamiento del modelo neuronal 35
4.12. Detalles en la señal neuronal 36
4.13. Arquitectura neuronal para la generación locomotora 38
4.14. Salida Motoneurona en función de Promotora y Remotora 39
4.15. Correspondencia entre osciladores y módulos 40

4.16. Modelo CPG Inhibición simétrica 41
4.17. Modelo CPG Inhibición asimétrica 42
4.18. Modelo CPG Bucle inhibitorio 42
4.19. Modelo CPG Push-Pull 43
4.20. Modelo CPG de Neuronas biestables intermedias 43
4.21. Funcionamiento de las Neuronas biestables intermedias 44
4.22. Fotografía de Olus2 46
4.23. Estructura básica de una Nariz Electrónica 47
4.24. Diagrama de sistema de Olus2 48
4.25. Fotografía del PIC18LF4550 48
4.26. Fotografía del sensor TGS2600 49
4.27. Circuito de acondicionamiento del sensor TGS2600 49
4.28. Comportamiento TGS2600 con la temperatura de calefacción 50
4.29. Estructura de montaje sensor de temperatura TC1047A 51
4.30. Caja de Olus2 51
4.31. Puerto de comunicaciones de Olus2 52
4.32. Esquema resistencias PULL-UP en la comunicación I2C 54
4.33. Condición de bus libre de I2C 55
4.34. Condición de arranque de I2C 55
4.35. Condición de cambio de I2C 56
4.36. Condición de parada de I2C 56
4.37. Trama para seleccionar dispositivo con el que comunicarse y establecimiento R/W 56

5.1. Comparativa neuronal en función de alfa y sigma. Muestra el potencial de membrana (rojo) y el susbsistema de dinámica rápida (verde) 61
5.2. Comparativa neuronal en función de la corriente sináptica, beta y sigma. Muestra el potencial de membrana (rojo) y el susbsistema de dinámica rápida (verde) . . 62
5.3. Dos neuronas conectadas con *g=0* 64
5.4. Dos neuronas conectadas con *g=0.043* 65
5.5. Dos neuronas conectadas con *g=-0.029* 65
5.6. Señal de salida de la motoneurona en función del valor de las neuronas Promotora y Remotora 67
5.7. Salida de la motoneurona para $\chi = 20$ 68
5.8. Salida de la motoneurona para $\alpha_M = 15$ 69
5.9. Salida de la motoneurona para $\nu = -0{,}5$ 69
5.10. Diagrama de un CPG con la nomenclatura de neurona y módulo que se utiliza en secciones posteriores 70
5.11. Comparativa de la Neurona #0 frente a las Neuronas #1, #2 y #3 71
5.12. Señal de las motoneuronas en el Modelo con Inhibición Simétrica 72
5.13. Señal de las motoneuronas para 4, 6 y 8 módulos respectivamente 72

5.14. Diagrama explicativo para el cálculo del desfase 74
5.15. Diferencia de fase entre la salida de dos motoneuronas 74
5.16. Mapa de conductancias para 2 módulos 78
5.17. Mapa de conductancias para 4 módulos 78
5.18. Mapa de conductancias para 6 módulos 79
5.19. Mapa de conductancias para 8 módulos 79
5.20. Mapa de conductancias ampliado para 8 módulos 80
5.21. Mapa de conductancias ampliado para 8 módulos y con el valor medio de los desfases internos 81
5.22. Desfase mostrado con 8 módulos utilizando conductancia óptima 81
5.23. Comparativa señal motoneurona con función seno 83
5.24. Robot desplazándose en RoboSim 83
5.25. Swap de las conductancias 84
5.26. Cuadro de mando de servos8 85
5.27. Descarga de programas a SkyPic mediante Pydownloader 86
5.28. Correspondencia de numeración de los módulos con la de los servos en el programa servos8 86
5.29. Alimentación de los servos 88
5.30. Correspondencia de los pines de Arduino con el conector del robot 89
5.31. Arduino y robot conectados mediante cables independientes 89
5.32. Cube Revolutions controlado por SkyMega 92
5.33. Relación pines del conector del robot con el puerto de expansión de SkyMega . . 93
5.34. Adaptador jack-molex 93
5.35. Disposición CPG de SkyMega 94
5.36. Configuración de los módulos #3 y #6 en los 3 modos de movimientos 94
5.37. Mapa paramétrico de las conductancias, con zoom en la zona de interés 95
5.38. Mapa paramétrico de valores válidos filtrando mediante el desfase de los módulos intermedios 95
5.39. Desfase mostrado por los 6 módulos que conforman el CPG a introducir en SkyMega 96
5.40. Desconexión del conector jack-molex 96
5.41. Adaptador jack-molex integrado en la estructura del robot 97
5.42. Distribución de pines del Pickit2 y de Olus2 98
5.43. Diagrama de funcionamiento del MSSP 100
5.44. Registro SSPSTAT del MSSP 100
5.45. Registro SSPCON1 del MSSP 101
5.46. Registro SSPCON2 del MSSP 102
5.47. Tratamiento de direcciones I2C por parte de Olus2 y Arduino 106
5.48. Formato de las tramas de Transmisión y Recepción creadas para la comunicación I2C (a la izquierda el bit más significativo) 107

5.49. Captura de Arduinoscope 108
5.50. Olus2 integrada en la estructura del robot 108
5.51. Actuación del robot al aplicar el algoritmo de búsqueda 111
5.52. Señal de odorante comparada con el sentido de movimiento del robot 111
5.53. Valor obtenido por la Nariz Electrónica 112
5.54. Señal de odorante comparada con el sentido de movimiento del robot (segundo algoritmo) 113

A.1. Arquitectura comunicaciones entre Robot y Nariz Electrónica 129
A.2. Diagrama de interconexión de SkyMega y Olus2 130

B.1. Estructura de `script`, mostrando los pasos que ejecuta 133
B.2. Estructura de `source`, programa principal de la simulación 140
B.3. Relación entre HW y SW al comunicar Robot y Nariz Electrónica 145

Índice de cuadros

4.1. Ángulo de corrección en los módulos 29

5.1. Código que representa a las ecuaciones 4.1 y 4.2 60
5.2. Código que representa a la ecuación 4.3 60
5.3. Estructura de la neurona utilizada en el código del modelo neuronal 61
5.4. Código que define la sinapsis eléctrica entre neuronas 63
5.5. Código que representa a la ecuación 4.10 66
5.6. Código que representa a las ecuaciones 4.11 y 4.12 67
5.7. Pseudocódigo del programa que se encarga de calcular el desfase 73
5.8. Variables que almacenan las conductancias del modelo 75
5.9. Ocurrencia de fases para *g1=-0.019* y *g2=0.035* 76
5.10. Ocurrencia de fases para *g1=-0.02* y *g2=0.0197* 76
5.11. Tamaño de las variables necesarias para simular el CPG 87
5.12. Ejemplo de la librería Servo de Arduino 90
5.13. Locomoción del robot mediante osciladores 91
5.14. Configuración de los pines de SkyMega para controlar el robot 93
5.15. Configuración de los pines de Olus2 99
5.16. Configuración de Olus2 102
5.17. Dirección de esclavo I2C de la Nariz Electrónica 106
5.18. Configuración de la comunicación I2C 109
5.19. Algoritmo de búsqueda por umbral 110

1

Introducción

El Proyecto Fin de Carrera que se describe en este documento se trata de un proyecto de investigación que trata acerca del control locomotor de un robot modular mediante un CPG (Generador Central de Patrones) y las variaciones de este control locomotor ante estímulos tales como son los odorantes recibidos por una Nariz Electrónica.

El trabajo que se ha realizado no parte de cero, si no que se apoya en trabajos previos realizados en el departamento. Está compuesto por tres partes claramente diferenciadas que se han tratado de unir para realizar un proyecto multidisciplinar. Estas partes son: Robot (Parte mecánica), CPG (Control locomotor de la mecánica) y Nariz Electrónica (Elemento sensorial). En todo sistema natural de locomoción se puede distinguir la parte mecánica, la parte de control locomotor y la parte sensorial.

Comenzando por la parte mecánica, el robot que se ha utilizado es fruto del trabajo realizado durante mucho tiempo por parte de Juan González. Ha sido capaz de diseñar una serie de robots modulares e implementar las secuencias de movimiento para conseguir un correcto desplazamiento. El modelo que se utiliza en este proyecto tiene como nombre Cube Revolutions y se trata de un robot modular que se mueve utilizando señales sinusoidales desplazadas unas de otras. Juan González ha dispuesto un página web donde se encuentra toda la documentación acerca de los robots que ha diseñado [1]. En esta web se pueden encontrar los pasos para fabricar los robots, el código que hay que insertarles para conseguir que funcionen y los programas con los que descargar al robot el código desde el PC. En la Figura 1.1 se puede ver a Juan González sosteniendo el robot.

En segundo lugar está el Generador Central de Patrones. El trabajo aquí expuesto se ha basado en el trabajo previo realizado por Fernando Herrero [2] y Damián Zamorano [3]. Ellos tuvieron ocasión de trabajar con el robot que se acaba de presentar, y fueron capaces de realizar el control locomotor mediante el uso de CPGs. Los Generadores Centrales de Patrones (CPG) son circuitos neuronales que pueden reproducir patrones rítmicos de actividad neuronal sin recibir entradas rítmicas. Fernando Herrero fue el primero en trabajar en este ámbito, y se ocupó de integrar el CPG en el control del robot para que este se desplazase imitando el movimiento de un gusano. Posteriormente Damián Zamorano se encargó de diseñar varios CPGs con los que podía generar distintos movimientos en el robot. Comentar, aunque respecto a esto se entrará en detalle luego, que el primero consiguió movimientos del robot con un único grado de libertad (una dirección, dos sentidos) mientras que el segundo consiguió con su trabajo realizar movimientos con dos grados de libertad (movimiento por todo el plano horizontal). En la Figura 1.2 se puede apreciar la idea de un robot controlado mediante un CPG.

Figura 1.1: Juan González con su robot, Cube Revolutions

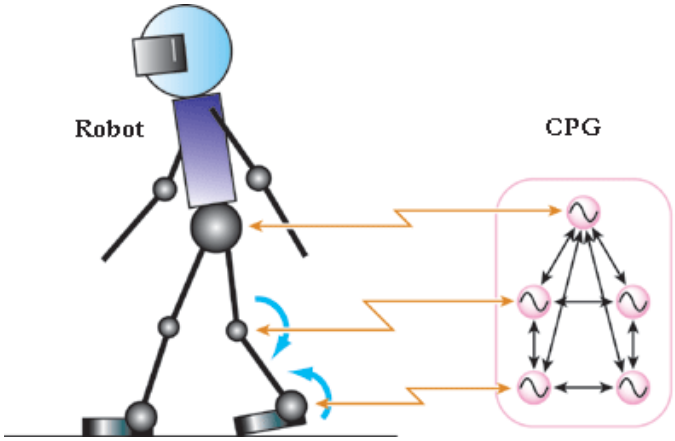


Figura 1.2: Robot controlado mediante un CPG

Por último se encuentra el elemento sensorial que le dará feedback a nuestro sistema. Se trata de un elemento innovador, siendo este trabajo el pilar sobre el que se van a sustentar futuros proyectos del GNB con los que se quiere llegar a poder realizar la búsqueda de la fuente de odorante. La Nariz Electrónica que se va a utilizar en este proyecto fue desarrollada por David Yáñez en su Trabajo Fin de Máster en el GNB [4], y posteriormente fue mejorada en su etapa profesional [5]. Se trata de un modelo de Nariz Electrónica de bajo coste y de electrónica sencilla que se está continuamente perfeccionando. Tras diseñarla, David Yáñez se encuentra ahora intentando caracterizar distintos tipos de odorantes para así poder clasificar elementos en función del olor que desprenden. Por ejemplo se ha utilizado para realizar test con los que caracterizar el olor que desprenden sustancias como el café o la mantequilla de cacahuete [6].

Este proyecto ha sido llevado a cabo dentro del Grupo de Neurocomputación Biológica, GNB. El GNB se encarga de investigar, basándose en la naturaleza que nos rodea a todos nosotros, el uso de sistemas bio-inspirados. Para llevar el proyecto a cabo se asignó un puesto en su laboratorio, y se puso a disposición a las personas que podían aportar la información necesaria para su realización. También se suministró el material necesario: el robot utilizado en anteriores proyectos, la nariz electrónica de David Yáñez, las diversas placas controladoras del robot, y también todo el material que se fue necesitando a lo largo de la realización del proyecto.

La ejecución del proyecto comenzará con la lectura, comprensión e introducción de modificaciones de los CPGs propuestos en los anteriores trabajos del grupo. Se empezará probando con sistemas básicos, que ayudarán a la comprensión de las ecuaciones que condicionan el CPG

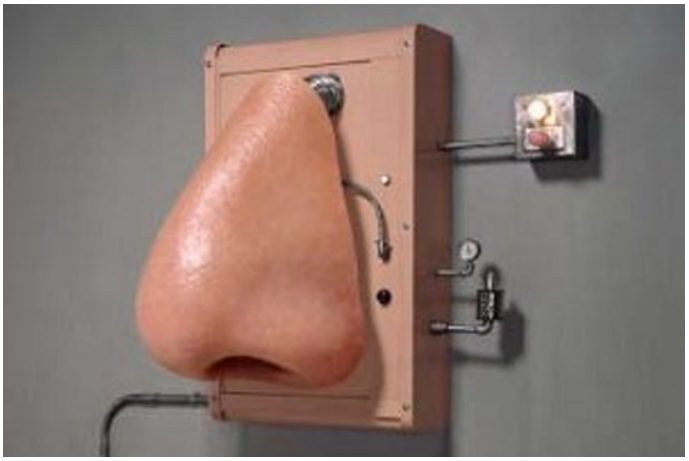


Figura 1.3: Idealización de una Nariz Electrónica

y poco a poco se irá completando hasta llegar al modelo final, creando un modelo totalmente nuevo y diferente a los realizados en trabajos anteriores del GNB. Terminado el programa que simule el CPG se ajustará el modelo hasta que se consiga un resultado válido para conseguir el desplazamiento del robot. Antes de probarlo en el propio robot se trabajará con el simulador RoboSim, el cual permitirá ver el funcionamiento del modelo sin tener que preocuparse de la programación del robot. Realizando pruebas en el simulador se terminarán de ajustar los parámetros.

Conseguido unos parámetros válidos se pasará a introducir el modelo en la placa que controla el robot, la SkyPic. Para ello primero habrá que aprender a programar la placa. En este momento se verá si la SkyPic es capaz de manejar el modelo por si misma ya que en anteriores trabajos se le pasaban los resultados de la simulación del CPG desde el ordenador y únicamente se encargaba de posicionar a los servos en función de esos valores recibidos. Se explorarán varias posibilidades (SkyPic, Arduino y SkyMega) teniendo como objetivo la inserción del modelo en las mismas. Una vez encontrado el microcontrolador adecuado se reescribiría el código perteneciente al modelo para hacerlo lo más eficiente posible y se pasaría a observar la respuesta de los motores del robot, ajustando otra vez los parámetros hasta obtener un movimiento convincente.

Llegado a este punto se pasará a estudiar la Nariz Electrónica para saber que datos se pueden obtener de ella y como. Se procederá a diseñar los módulos I2C, tanto en la nariz como en el robot, que permitan comunicar a ambos dispositivos. Se decidirá si utilizar las librerías existentes o bien se utilizar módulos propios.

En cuanto la comunicación esté establecida se integrará la medición de odorante en el programa encargado de mover al robot. En este punto, disponiendo el CPG de feedback, se diseñará alguna estrategia para actuar frente a la detección de odorante. haciendo que la locomoción cambie si la medida obtenida presenta variaciones en sus capturas.

MOTIVACIÓN

Centrándose en la función de locomoción, el estudio del fundamento neurológico ha facilitado el descubrimiento de estrategias que pueden ser aplicadas a diseños en robótica. La investigación de sistemas de locomoción bio-inspirados tiene grandes expectativas de futuro ya que se pueden lograr optimizaciones y mejoras en la eficiencia en máquinas ya diseñadas o que están por desarrollar. En particular, y centrándose en el control motor, a la robótica le interesa el estudio de los Generadores Centrales de Patrones [7, 8, 9, 10, 11]. Los CPG son unas redes neuronales existentes en muchos seres vivos (en el caso de los mamíferos se encuentra en la columna vertebral), que generan actividades rítmicas capaces de controlar neuronas motoras que se encargan de producir aquellos movimientos que requieran cierta periodicidad, robustez y

precisión (tales como caminar, nadar, reptar, volar, etc.). Son capaces de generar el movimiento sin tener que programar exactamente la cinemática del movimiento y se recuperan del ruido rápidamente, lo que los hace muy robustos. Se ha estudiado ampliamente la robustez, precisión y periodicidad del ritmo generado con los CPGs por parte de del GNB [10, 11, 12, 13]. Hay que hacer una mención especial a la autonomía que los CPGs puede aportar a la robótica, pues son capaces de resolver situaciones inesperadas al introducir un feedback sensorial. Además, se tratan de redes autónomas, por lo que no es necesario recibir un estímulo externo para producir un determinado patrón.

Hace casi 90 años, Brown sugirió que la alternancia entre la flexión y extensión de los músculos de la pierna al andar podía ser producida por circuitos centrales encargados del ritmo en el que músculos antagonistas eran manejados por neuronas que se inhibían una a la otra [14]. Recientes estudios [8, 15, 16] confirman esta teoría, demostrando que estos circuitos neuronales, los CPGs, tienen como componente básico la inhibición mutua y que están formados por las neuronas y las sinapsis que se producen entre estas. Además, muestran una dinámica con diferentes escalas temporales [17, 18], que rápidamente pueden producir una fuerte secuencia de activaciones [19]. Otra característica que se muestra en los circuitos neuronales es una dinámica invariante para presentar ritmos que sean robustos y flexibles al mismo tiempo [20, 21, 22].

La adaptación ocurre cuando un sistema tiene ciertas partes cuya función es la percepción, y otras cuya función es la acción. A través de la percepción un sistema adaptativo percibe el ambiente que lo rodea, y posiblemente, condiciones internas del propio sistema. A través de la acción el sistema es capaz de adaptar su estado a esas condiciones y asegurar su existencia.

Un caso especial de adaptación es la auto-regulación. Es el proceso por el cual varias o todas las partes del sistema trabajan para asegurar que el estado interno del sistema se mantiene en límites seguros. En seres vivos esto incluye la adquisición de energía suficiente, deshecho de residuos, respiración, etc. Es crucial que las partes responsables de la auto-regulación reciban información de su actuación. En el caso de la auto-regulación generalmente se habla de feedback, en vez de percepción o detección.

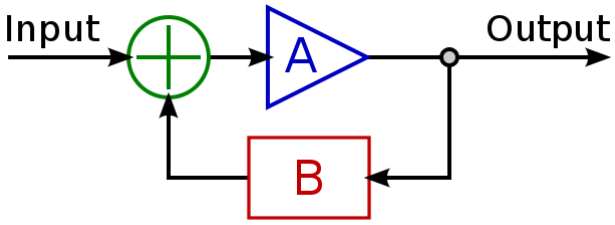


Figura 1.4: Sistema con feedback

Los CPGs pueden aportar una gran autonomía a la robótica, pues son capaces de resolver situaciones inesperadas al introducir un feedback sensorial. La información proveniente de los sentidos es capaz de modular la actividad de los CPGs, con el fin de ajustarlos a las condiciones externas. Por tanto, el CPG tiene que ser capaz de asumir múltiples estados en respuesta al feedback que se obtenga.

La idea de incluir la Nariz Electrónica al sistema responde a la necesidad de introducir un estímulo sensorial que permita analizar el entorno en el que se encuentra y variar los parámetros del CPG en función de la presencia de odorante que se reciba en cada momento. Es el primer paso de un proyecto mucho más grande que se irá realizando a lo largo de los póximos años en el GNB.

Mediante el valor medido por la Nariz Electrónica, y haciendo un correcto análisis de los datos obtenidos, se pueden realizar aplicaciones realmente interesantes. Comparando las medidas realizadas por el sensor con valores almacenados en una completa base de datos se puede

discriminar entre diferentes odorantes [23, 24, 6]. Por otro lado, si se integra la nariz en un robot (como se va a realizar en este proyecto) se puede realizar la búsqueda de la fuente de odorante [25, 26]. Por último, para obtener la pluma de odorante se pueden utilizar una red de sensores estáticos, estableciendo comunicación entre ellos para realizar un análisis con todos los valores obtenidos en distintos puntos [27, 28].

Dentro del mundo de la robótica, cabe destacar la flexibilidad que puede proporcionar un diseño modular, y en el que se pueden estudiar diferentes técnicas de locomoción [29, 30]. En sí mismo, el diseño modular es bio-inspirado, pues en el proceso evolutivo se generan seres vivos compuestos por diferentes módulos (cochinilla, ciempiés, etc.). A partir de módulos homogéneos se puede construir fácilmente robots de distinto tamaño o reconfigurar su topología. La modularidad puede emplearse además en sistemas distribuidos, donde se puedan diseñar patrones reproducidos localmente en cada módulo pero manteniendo una locomoción del conjunto de ellos, es decir el robot, adecuada.

Los estudios realizados hasta la fecha de los CPGs en el entorno de la neurociencia, han aportado ideas en el ámbito del control de robots, aplicando dichos conocimientos en diferentes aspectos. Por ejemplo un modelo muy realista del movimiento que realiza la extremidad de un crustáceo [31]; el desarrollo de una red neuronal artificial para controlar un robot hexápodo [32]; diferentes formas de controlar aletas [33] y alas [34] realizando un análisis riguroso acerca de la convergencia y estabilidad de los controladores; la locomoción bípeda [35, 36]; o la locomoción modular [37]. Desde el GNB también se han aportado ideas muy interesantes por parte de Fernando Herrero con un robot gusano [38, 39, 40, 41] y por parte de Urziceanu con un robot con movilidad reducida [42].

Por último indicar que la memoria de este proyecto se ha realizado en gran parte como un manual con el fin de que el GNB pueda disponer de la información suficiente con la cual pueda ir introduciendo a nuevas personas en proyectos que giren alrededor de éste. Esto se realiza para que la información esté reunida y no se tenga que invertir tiempo en buscarla.

ORGANIZACIÓN DE LA MEMORIA

La memoria está dividida en los siguientes capítulos:

Objetivos: En este apartado se exponen los objetivos que se pretenden alcanzar con la realización de este proyecto.

Estado del arte: En este apartado se establece el marco teórico en el que se encuentra el proyecto, se expone de forma más precisa el problema a tratar y las soluciones que se pueden encontrar en la literatura.

Diseño: En este apartado se mostrará el diseño del sistema. Se mostrarán las ecuaciones que forman las neuronas del modelo elegido y las conexiones que existen entre ellas. Además, se realizará una introducción del CPG que se usará en el proyecto, no con el fin de entender su funcionamiento en este punto, sino para representar las conexiones y las neuronas del CPG. Se presentará el modelo de Nariz Electrónica que se utilizará en el proyecto. Por último también se presentarán los diferentes microcontroladores que se pueden llegar a utilizar para controlar el robot.

Desarrollo e Integración: Este apartado primero se centrará en el funcionamiento del CPG. Se empezará explicando el programa desarrollado para tal fin, inicialmente introduciendo un único módulo para después mostrar el modelo completo y su funcionamiento global. Posteriormente se realizarán pruebas en el simulador del robot RoboSim para visualizar el correcto movimiento con el CPG diseñado A continuación se introducirá el modulo de comunicación entre la Nariz Electrónica y el robot. Por ultimo se comprobará que el robot se desplaza adecuadamente,

probándose también la comunicación entre la nariz y el robot.

Resumen de objetivos: En este capítulo se presentará el resultado al que se ha llegado en los objetivos específicos que se marcaron en el inicio del proyecto. Gracias a esta enumeración se verá si se han cumplido o no los objetivos marcados, y por tanto, el éxito o no del trabajo realizado.

Conclusiones: En este apartado se exponen las conclusiones de este Proyecto Fin de Carrera.

Trabajo futuro: En este capítulo se presentan posibles continuaciones del trabajo realizado.

Referencias: Aquí se listarán las referencias de los artículos, libros y web consultadas para la realización de este proyecto.

Glosario: Se expondrán las siglas utilizadas en el proyecto.

Anexos: Serie de anexos en los que entre otras cosas se incluirán los programas implementados.

2
Objetivos

En este breve capítulo se va a introducir el conjunto de objetivos que se plantea haber cumplido tras la realización del proyecto. En primer lugar se presentarán una serie de objetivos generales, en los que se puede apreciar el resultado que se quiere lograr, pero de una manera muy superficial. En segundo lugar se detallarán los objetivos específicos que se quieren llegar a completar, detallándose en este caso las tareas que se necesitará realizar.

OBJETIVOS GENERALES

El objetivo principal de este proyecto es el desarrollo del CPG como generador de los comandos motores y la introducción de una nariz ultra-portátil en el robot modular y así, controlar ciertos movimientos básicos que realizará éste mediante la recepción de odorantes por parte de la nariz electrónica.

La locomoción del robot modular estará controlada por un CPG que será simulado de manera autónoma por el controlador del robot. Esta locomoción tendrá un único grado de libertad por lo que dispondremos únicamente de una dirección y dos sentidos en el movimiento (hacia delante y hacia atrás).

A continuación los sub-objetivos que se abordan en este proyecto:

1. Estudio de la literatura sobre el modelo neuronal.
2. Estudio de la literatura sobre narices electrónicas.
3. Desarrollo de un modelo neuronal propio basándose en los ya existentes.
4. Implementación del modelo neuronal en el robot.
5. Realización de un programa que adquiera los datos de salida de la nariz electrónica.
6. Diseño de la comunicación entre la nariz y el robot.
7. Integración de la nariz en el funcionamiento del robot.
8. Prueba de concepto de algoritmo con el que se responda a estímulos olfativos.
9. Análisis de los resultados y extracción de conclusiones.

OBJETIVOS ESPECÍFICOS

Los objetivos, ya de una manera más específica, que se establecen como objetivos primarios y se deben haber cumplido total o parcialmente al finalizar el proyecto son los siguientes:

- Utilizar un **modelo de sinapsis eléctricas** para establecer la comunicación entre las neuronas del CPG. En trabajos anteriores se utilizan sinapsis químicas pero para este proyecto se busca un modelo más sencillo que signifique una menor carga computacional en el microcontrolador como resulta ser el modelo de sinapsis eléctricas.

- **Diseñar el CPG** de manera que el robot pueda desplazarse **utilizando el modelo de sinapsis eléctricas implementado** en el objetivo anterior para conectar a las neuronas. Sería interesante realizar un CPG que **permita realizar los movimientos básicos**, pudiendo así desplazarse el robot en varias direcciones del plano horizontal.

- Diseño de un **algoritmo con el que calcular el desfase entre módulos**, ya que será un elemento muy importante para ajustar el peso de las sinapsis y ajustar el CPG para obtener desplazamiento.

- **Generación del movimiento de manera autónoma** por el microcontrolador. Se realizará un estudio de cómo se puede introducir el CPG en una tarjeta controladora. Búsqueda del microcontrolador adecuado para poder ejecutar la simulación.

- Establecimiento de la **comunicación I2C** entre los dos dispositivos, permitiendo el traspaso de información entre el sensor y el robot. Se estudiará si se debe utilizar los módulos y librerías existentes o si se debe de crear algo específico para este problema.

- **Mejora de la movilidad del robot** mediante la realización de una serie de añadidos en la estructura del robot, tratando de integrar toda la electrónica que se necesita en la propia estructura. Tras ello se podrán realizar pruebas sin tener que preocuparse de posibles desconexiones.

- **Variación del movimiento ante estímulos olfativos** mediante algún sencillo algoritmo. No se pretende realizar una búsqueda concluyente de la fuente de odorante, ya que sería un objetivo demasiado ambicioso, pero si que el estado en el que se encuentra el robot se vea afectado por la presencia de odorante medida. Se tratará de una prueba de concepto para posteriores proyectos.

3

Estado del arte

Antes de introducir el diseño que se va a utilizar en el proyecto se van a introducir en detalle ciertos aspectos que ayudarán a entender el por qué se hace este proyecto, situando al lector en una posición en la que sea capaz de entender los temas que se van a desarrollar en los siguientes capítulos.

3.1. Bioinspiración

En primer lugar se va a hablar de la bioinspiración, explicando lo que es y porque es un aspecto importante en el proyecto.

La bioinspiración, también llamada biomimética, se trata del proceso de entender y aplicar a problemas planteados por los humanos, soluciones procedentes de la naturaleza en forma de principios biológicos, biomateriales, o de cualquier otra índole.

Conforme la sociedad ha ido evolucionando en torno al alcance de las tecnologías, la biología se ha convertido cada vez más en una ciencia fundamental de la que obtener ideas con las que realizar avances en ingeniería. Desde la perspectiva de la ingeniería, los organismos vivos son las máquinas definitivas; se adaptan a condiciones que varían continuamente, son capaces de buscar energía del medio en el que se encuentran y además son capaces de hacerlo de la manera más sencilla [43].

Otro aspecto muy importante es que estos organismos han encontrado la forma de hacer lo que hacen mientras cuidan del hábitat que ocupará su descendencia. Si se consigue comprender y reproducir todo lo que ha sido capaz de desarrollar la vida, se conseguirá vivir con elegancia en la tierra llevando a cabo un desarrollo sostenible. No se trata de que se puede extraer de la naturaleza, sino de que se puede aprender de ella.

En este proyecto se va a introducir un modelo neuronal bioinspirado, basado en el circuito neuronal encargado de generar la coordinación de las articulaciones en seres vivos. Este modelo no está únicamente basado en la naturaleza, ya que introduce aspectos creados artificialmente, pero si que aplica ciertos aspectos presentes en la naturaleza y que le aportan características enriquecedoras (establecimiento y mantenimiento del ritmo necesario para el desplazamiento,

ausencia de señales de control, feedback con el que poder adaptarse al medio, etc).

3.2. Modelos dinámicos

Modelar es una tarea fundamental en numerosas disciplinas científicas. Se trata del resultado del proceso de generar una representación de los sistemas a fin de analizar los fenómenos o procesos que se desarrollan en ellos.

Los modelos que se introducen en el siguiente capítulo [44, 45] se tratan de modelos dinámicos. Un sistema dinámico consiste en una serie de variables que describen su estado y unas leyes que describen la evolución de estas variables de estado a lo largo del tiempo [46]. El estado del sistema en el siguiente momento depende del valor de sus entradas y de sus estado (el valor de sus variables) en el instante anterior. La modelización de estos sistemas da lugar a modelos dinámicos en los que el estado actual depende de instantes previos.

El modelo que se va a desarrollar en este proyecto es una Red Neuronal con inspiración biológica, más en concreto un CPG, y tiene como objetivo la generación de las señales de control locomotor del robot (los aspectos del robot a utilizar se encuentran en el Apartado 4.1). La Red Neuronal está formada por neuronas y sus conexiones, las sinapsis, elementos que tienen sus propios modelos. A continuación se presentan estos elementos para mas tarde poder introducirlos en el modelo global de la red neuronal.

3.3. La neurona

Dentro del modelo neuronal que se va a realizar, la unidad mínima con la que se va a trabajar es la neurona. Se utilizará un modelo de neurona, pero para comprender su funcionamiento se debe entender previamente las características de las neuronas reales.

A principios del siglo XX, Santiago Ramón y Cajal situó por vez primera las neuronas como elementos funcionales del sistema nervioso. Cajal propuso que actuaban como entidades discretas que, intercomunicándose, establecían una especie de red mediante conexiones especializadas o espacios. Esta idea es reconocida como la doctrina de la neurona, uno de los elementos centrales de la neurociencia moderna.

Las neuronas son un tipo de células del sistema nervioso cuya principal característica es la excitabilidad eléctrica de su membrana plasmática [47, 48]. Están especializadas en la recepción de estímulos y conducción del impulso nervioso (en forma de potencial de acción) entre ellas y con otros tipos celulares, como por ejemplo las fibras musculares de la placa motora. Presentan unas características morfológicas típicas que sustentan sus funciones: un cuerpo celular llamado soma o pericarion; una o varias prolongaciones cortas que generalmente transmiten impulsos hacia el soma celular, denominadas dentritas; y una prolongación larga, denominada axón, que conduce los impulsos desde el soma hacia otra neurona.

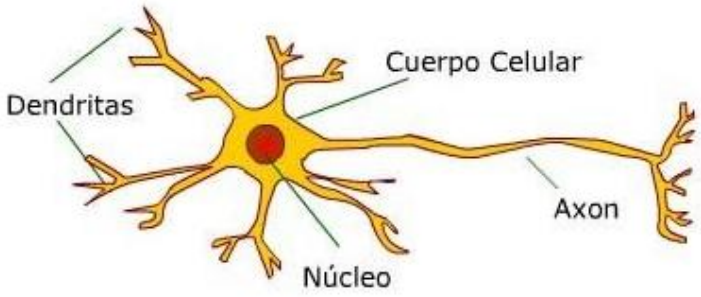


Figura 3.1: Estructura de una neurona

Tienen la capacidad de comunicarse con precisión, rapidez y a larga distancia con otras células, ya sean nerviosas, musculares o glandulares. Si se trata de una conexión que une una neurona con otra se denomina sinapsis.

Las neuronas transmiten ondas de naturaleza eléctrica originadas como consecuencia de un cambio transitorio de la permeabilidad en la membrana plasmática [49]. Su propagación se debe a la existencia de una diferencia de potencial o potencial de membrana (que surge gracias a las concentraciones distintas de iones a ambos lados de la membrana) entre la parte interna y la externa de la célula (por lo general de unos -70 mV). La carga de una célula inactiva se mantiene en valores negativos (el interior respecto del exterior) y varía dentro de unos estrechos márgenes. Cuando el potencial de membrana de una célula excitable se despolariza más allá de cierto umbral (de -65 a -55 mV aproximadamente), la célula genera un potencial de acción. Un potencial de acción es un cambio muy rápido en la polaridad de la membrana, cambiando de negativo a positivo y vuelta a negativo, en un ciclo que dura varios milisegundos. Este proceso se puede apreciar en la siguiente figura:

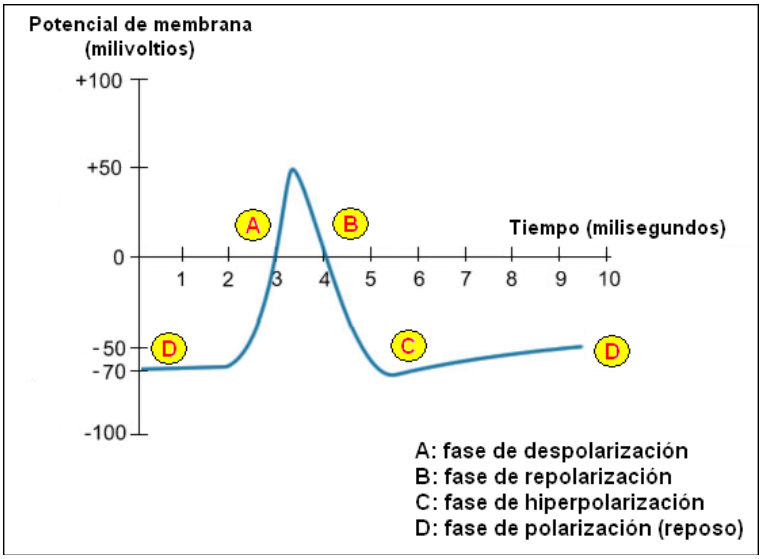


Figura 3.2: Variación en el potencial de membrana

El impulso nervioso se transmite a través de las dentritas y el axón (véase Figura 3.1). La velocidad de transmisión del impulso nervioso depende fundamentalmente de la velocidad de conducción del axón, la cual depende a su vez del diámetro del mismo. El axón lleva el impulso a una sola dirección y es transmitido de un espacio a otro. Las dentritas son las fibras nerviosas de una neurona, que reciben los impulsos provenientes desde otras neuronas. El espacio entre axón y dentrita se denomina hendidura sináptica o espacio sináptico (véase Figura 3.3). En las grandes neuronas alfa de las astas anteriores de la médula espinal, las velocidades de conducción axonal pueden alcanzar los 120 m/s. Si consideramos que una persona normal puede medir 1,75

metros de altura, el impulso eléctrico tardaría únicamente 14.5 milisegundos en llegar desde la punta del pie hasta el cerebro.

El número de neuronas en el cerebro varía drásticamente según la especie estudiada. Se estima que cada cerebro humano posee en torno a cien mil millones de neuronas [47]. No obstante, un gusano nematodo muy empleado como animal modelo para la biología, posee sólo 302; y la mosca de la fruta unas 300.000, que bastan para permitirle exhibir conductas complejas. La fácil manipulación en el laboratorio de estas especies, que tienen un ciclo de vida muy corto y condiciones de cultivo poco exigentes, permite a los investigadores científicos emplearlas para dilucidar el funcionamiento neuronal, puesto que el mecanismo básico de su actividad neuronal es común al de nuestra especie.

Atendiendo a su especialización funcional las neuronas pueden clasificarse en los siguientes tipos:

- Neuronas sensoriales: Especializadas en la recepción de sensaciones provenientes del exterior; su característica esencial es disponer de especializaciones que les permiten transformar la energía física de los estímulos en fenómenos de despolarización/polarización para la generación de potenciales de acción.
- Interneuronas: Se encuentran en el camino entre dos o mas neuronas; presentan la capacidad de cambiar el signo (excitación-inhibición) del estímulo ocasionando despolarización o hiperpolarización de las células postsinápticas.
- Motoneuronas: Reciben su información sensorial bien directamente de las células sensoriales, bien a través de las interneuronas, estando sus terminaciones axónicas acopladas a órganos efectores, como es el caso de los músculos.

En el modelo que se va a presentar en el siguiente capítulo estarán incluidas tanto Interneuronas (las neuronas encargadas de generar y mantener el ritmo) como Motoneuronas (las neuronas encargadas de generar las señales para controlar a los servos del robot).

Para simular a la neurona, la unidad básica del sistema que se va a realizar, se utilizará el sencillo modelo propuesto por Rulkov [44], que imita en gran medida el comportamiento eléctrico ofrecido por neuronas reales (este tema se trata en el Apartado 4.2) mediante el uso de dos señales principales (una de dinámica lenta y otra rápida).

A la hora de simular a la motoneurona se utilizará el modelo "Oscilador Half-Center", propuesto por Matsouka [50] y que muchos trabajos han utilizado como base para establecer la generación de actividad rítmica en robots [29, 51, 52]. Tanto este modelo como el anterior se han utilizado debido a que ya aparecían en los trabajos de Fernando Herrero [2] y Damián Zamorano [3], que son los dos trabajos principales en los que se basa este Proyecto. Este aspecto se desarrolla en el Apartado 4.4.

3.4. La sinapsis

Para establecer una red neuronal se necesita conectar unas neuronas con otras, y este cometido se realiza mediante la sinapsis.

La sinapsis es el mecanismo mediante el cual se produce el paso del impulso nervioso, desde una terminación axónica (véase Figura 3.1) a una prolongación dentrítica, mediante la generación de potenciales postsinápticos en la terminación dentrítica. Este proceso tiene su localización física en la presencia de dos superficies de membrana neuronal (una axónica y otra dentrítica, provenientes de diferentes neuronas), situadas en una gran proximidad una de otra

pero separadas por la denominada brecha o hendidura sináptica [48]. En la siguiente imagen se puede observar el proceso:

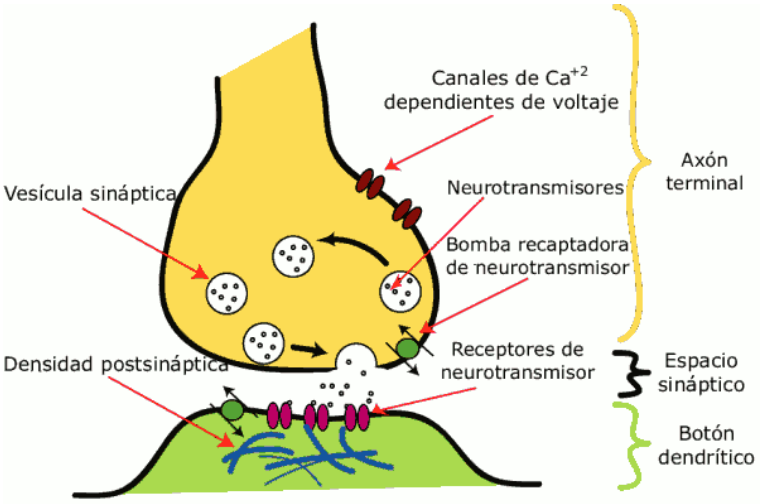


Figura 3.3: Elementos implicados en la sinapsis

El flujo de información que se produce entre las neuronas se puede apreciar en la siguiente imagen, mostrando los puntos de entrada y salida que existen en las neuronas:

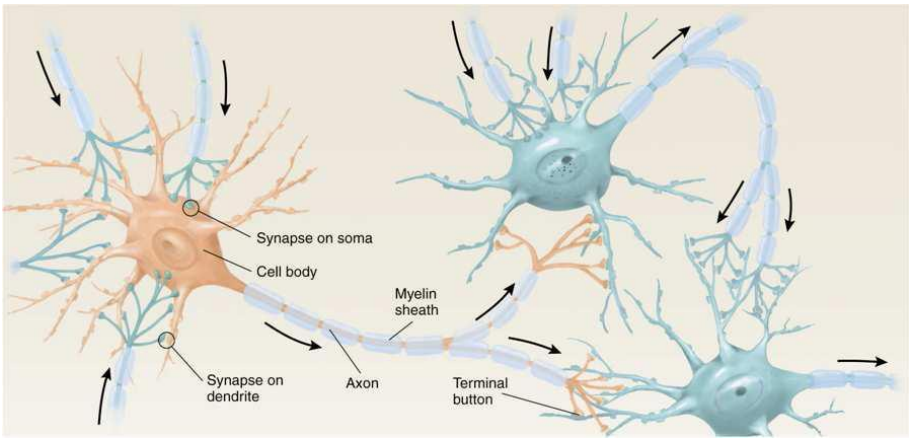


Figura 3.4: Estructura sináptica de una red neuronal

3.4.1. Clasificación de la sinapsis atendiendo a su naturaleza

La sinapsis suele clasificarse en dos tipos según la naturaleza de la transmisión del impulso [47, 48]:

- Sinapsis Química

  Son las predominantes en el cerebro de los mamíferos. En ellas una sustancia, el neurotransmisor, hace de puente entre las dos neuronas, se difunde a través del estrecho espacio y se adhiere a los receptores, que son moléculas especiales de proteínas que se encuentran en la membrana postsináptica.

  El mensajero químico enviado por una neurona a otra puede propagarse por difusión a otros lugares alejados de la sinapsis. Así, la neurotransmisión puede ocurrir en cualquier receptor compatible dentro del radio de acción del neurotransmisor, de forma equivalente a la comunicación moderna con teléfonos celulares que funcionan dentro del radio de

transmisión de una célula dada. Por tanto, el cerebro es no solamente una colección de cables, sino también una sofisticada sopa química.

- Sinapsis Eléctrica

  El acoplamiento eléctrico entre células se produce a través de conexiones de baja resistencia por las que el potencial presináptico pasa con pocos cambios de atenuación al terminal postsináptico. Físicamente es necesario el acoplamiento entre canales de las membranas pre y postsináptica (es necesario que las dos neuronas implicadas en la sinapsis estén conectadas físicamente). Este tipo de sinapsis se encuentra en el corazón humano.

  Las propiedades más importantes de las sinapsis eléctricas son las siguientes:

  a) Son rápidas y sin retraso.
  b) Pueden provocar flujo de corriente en ambas direcciones o pueden ofrecer más resistencia en una dirección que en la otra.
  c) Facilitan los procesos de sincronización entre neuronas.
  d) Son menos dependientes de los cambios metabólicos del medio extracelular que las químicas.

En este proyecto se van a utilizar sinapsis eléctricas entre las neuronas. En trabajos anteriores del departamento [2, 3, 38, 42, 39, 40, 41] se utilizaron sinapsis químicas para un mayor control de la comunicación entre las neuronas. En este proyecto, al querer desarrollar la simulación del modelo neuronal en un microcontrolador, se necesita utilizar modelos sencillos, efectivos y con el mínimo tiempo de ejecución. Por tanto, la sinapsis eléctrica es la mejor candidata para realizar las conexiones entre las distintas neuronas a utilizar en este proyecto (se presenta el modelo elegido en el Apartado 4.3).

3.4.2. Clasificación de la sinapsis atendiendo al efecto generado

Se dice que los estímulos que despolarizan la membrana de la neurona son excitatorios, debido a que una cantidad suficiente de ellos iniciarán el potencial de acción. Algunos estímulos originan el efecto opuesto de hiperpolarización, y en este caso el potencial de membrana excede la cifra del potencial de reposo (-70 mV). Los impulsos que causan hiperpolarización inhiben la generación de potenciales de acción, debido a que se oponen a los efectos de los estímulos despolarizantes [53].

La suma de los impulsos (tanto excitatorios como inhibitorios) que llegan por todas las sinapsis que se relacionan con cada neurona (de 1.000 a 200.000) determina si se produce o no la descarga del potencial de acción por el axón de esa neurona.

En el modelo desarrollado en este Proyecto se utilizan únicamente sinapsis inhibitorias, tanto en la conexión realizada entre las neuronas (interneuronas) como las conexiones realizadas con las motoneuronas.

3.5. Red neuronal artificial

El cerebro humano es el sistema de cálculo más complejo que conoce el hombre. El ordenador y el hombre realizan tareas de diferentes maneras; así la operación de reconocer el rostro de una persona resulta una tarea relativamente sencilla para el hombre y difícil para el ordenador, mientras que la contabilidad de una empresa es tarea costosa para un experto contable y una sencilla rutina para un ordenador básico.

La capacidad del cerebro humano de pensar, recordar y resolver problemas ha inspirado a muchos científicos intentar o procurar modelar en el ordenador el funcionamiento del cerebro

humano. El resultado ha sido una nueva tecnología llamada Computación Neuronal o también Redes Neuronales Artificiales, ANN [54]. Están constituidas por elementos que se comportan de forma similar a la neurona biológica en sus funciones más comunes, y están organizados de una forma parecida a la que presenta el cerebro humano.

Una red neuronal se define como una población de neuronas físicamente interconectadas o un grupo de neuronas aisladas que reciben señales que procesan a la manera de un circuito reconocible [48]. La comunicación entre neuronas (sinapsis), que conlleva un proceso electroquímico, implica que una vez una neurona es excitada por encima de cierto umbral, ésta se despolariza transmitiendo a través de su axón una señal que excita a las neuronas aledañas, y así sucesivamente. El sustento de la capacidad del sistema nervioso, por tanto, radica en dichas conexiones.

El primer modelo de ANN fue propuesto en 1943 por McCulloch y Pitts [55], en términos de un modelo computacional de actividad nerviosa. El modelo de McCulloch-Pitts es un modelo binario, en el que cada neurona tiene un escalón o umbral prefijado. Este primer modelo sirvió de ejemplo para los modelos posteriores de John Von Neumman (Ordenadores digitales) [56], Marvin Minsky (Inteligencia artificial) [57] o Frank Rosenblatt (Red Perceptron) [58].

Hoy en día, existen muchos grupos con sede en diferentes universidades de todo el mundo que están realizando trabajos de investigación en el área de las redes neuronales artificiales. Cada grupo tiene diferente énfasis y motivación como son los neurólogos, psicólogos del conocimiento, físicos, programadores y matemáticos. Todos ellos ofrecen nuevos puntos de vista e intuiciones en esta área de la técnica.

En este proyecto se utiliza un tipo concreto de Red Neuronal de inspiración biológica, el Generador Central de Patrones (explicado en el siguiente apartado), unidad encargada de generar la coordinación entre las distintas articulaciones del robot (se introducirá en la Sección 3.6) y así obtener el desplazamiento del mismo.

A la hora de clasificar a las Redes Neuronales se puede atender a varios aspectos. Dos clasificaciones posibles son:

3.5.1. Clasificación de redes neuronales en función de su grado de inspiración biológica

Una primera clasificación de los modelos de ANN podría ser atendiendo a su similitud con la realidad biológica.

- **Redes neuronales de tipo biológico**

  El objetivo principal de las redes neuronales de tipo biológico es desarrollar un elemento sintético para verificar las hipótesis que conciernen a los sistemas biológicos, así como las funciones sensoriales, de coordinación o motoras. En otros contextos, estas redes se utilizan para incorporar elementos de inspiración en un diseño de ingeniería.

- **Redes neuronales para aplicaciones concretas**

  Las diferentes configuraciones y algoritmos que se diseñan para las redes neuronales artificiales están inspiradas en la organización del complejo sistema neuronal del cerebro humano. No obstante, conviene aclarar que esta inspiración no supone que las ANN lleguen a emular el cerebro como algunos optimistas lo desean ya que entre otras limitaciones, el conocimiento sobre el modo de funcionamiento y comportamiento del cerebro es bastante simple y reducido. De hecho los diseñadores de redes artificiales van más lejos del conocimiento biológico actual y prueban nuevas estructuras que presentan un comportamiento adecuado y útil.

  Las ANNs dirigidas a aplicación, están en general poco ligadas a las redes neuronales biológicas. Dado que el conocimiento que se posee sobre el sistema nervioso en general no

es completo, se han de definir otras funcionalidades y estructuras de conexión distintas a las vistas desde la perspectiva biológica.

La Red Neuronal diseñada en este proyecto está enfocada a un diseño de tipo Biológico,ya que trata de imitar la generación de movimiento que se produce en seres vivos. Para ello cada neurona vendrá representada por un mapa iterado que simula el potencial de membrana de la neurona y que genera dinámicas similares a las que se pueden observar biológicamente, mediante Spiking-Bursting (uno de los tipos de codificación más complejos). La conectividad se basará fundamentalmente en la inhibición mutua de entre neuronas.

3.6. Robótica modular

Antes de explicarse la generación de movimiento mediante el uso de CPGs, se va a introducir brevemente la idea de robótica modular y por qué se ha elegido esta plataforma para la realización del proyecto.

Uno de los mayores problemas en la robótica es la locomoción. La solución tradicional se basa en analizar las características del terreno y a partir de ellas diseñar un robot específico [59]. En 1995, Mark Yim propuso utilizar robots formados a partir de módulos con la capacidad para ensamblarse unos con respecto a otros [60]. De esta forma, estos nuevos robots modulares podrían cambiar la forma adoptando diferentes configuraciones (véase Figura 3.5) y modos de caminar en función del terreno por el que se desplazasen en cada momento.

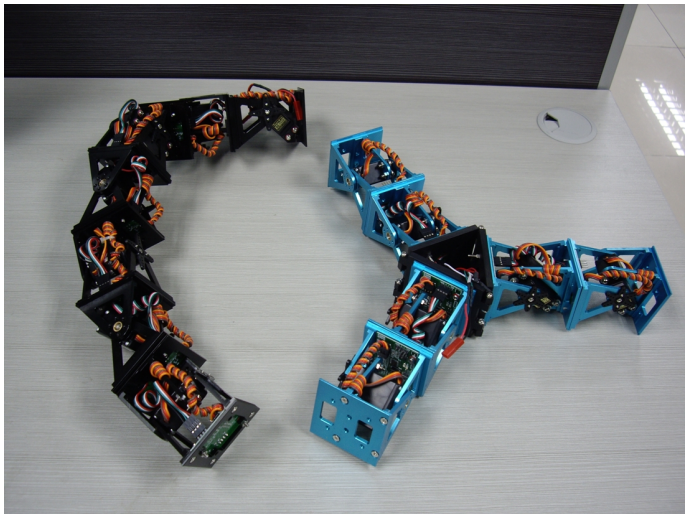


Figura 3.5: Ejemplo de robots modulares

Las tres principales ventajas de la robótica modular son la versatilidad, la fiabilidad y el bajo coste [61]. La versatilidad es debida a que estos robots pueden cambiar su forma y desplazarse por terrenos muy diversos. La fiabilidad radica en la capacidad de auto-reparación. Si uno de los módulos falla, se elimina o sustituye por otro. Finalmente, el bajo coste se consigue aplicando la economía de gran escala a la fabricación de los módulos. Si se fabrican masivamente, el precio se reducirá.

Aunque existen prototipos con un diseño muy depurado (véase Figura 3.6), todavía se están explorando sus posibilidades en usos prácticos. Las tres aplicaciones principales que se están desarrollando utilizando esta tecnología son:

- Búsqueda y rescate [62].

- Inspección de tuberías y puentes [63].
- Aplicaciones espaciales [64].

Debido a las ventajas presentadas en este apartado se ha utilizado el robot modular Cube Revolutions [65] en este trabajo y también en trabajos anteriores del departamento [2, 3].

3.7. Locomoción en robots mediante CPGs

La habilidad de moverse eficientemente en entornos complejos es un aspecto muy importante en los animales. Gracias a ello pueden evitar a sus predadores, buscar comida y encontrar compañeros de especie para reproducirse, lo cual es vital para su supervivencia. De modo similar, proveer a los robots de una buena locomoción es muy importante para diseñar robots que puedan realizar tareas útiles en una variedad amplia de entornos [66]. Esta importancia acerca de la locomoción tanto en la biología como en la robótica ha llevado a muchas interesantes interacciones entre los dos campos de la ciencia. Las interacciones han sido principalmente en un sentido, donde la robótica se inspira en la biología en términos de morfología, modos de locomoción yøcontrol de mecanismos. Cada vez con más frecuencia, la robótica esta devolviendo algo de esta inspiración a la biología, usando robots como herramientas científicas para probar hipótesis biológicas.

Los Generadores Centrales de Patrones son circuitos neuronales que se encuentran tanto en animales vertebrados como invertebrados que pueden producir patrones rítmicos de actividad neuronal sin recibir entradas rítmicas [47]. En su nombre, el término central indica que no es necesario un feedback sensorial para generar los ritmos de manera autónoma. Los modelos de CPG implementados como redes neuronales o sistemas de osciladores acoplados pueden usarse en la robótica para controlar la locomoción de los robots articulados. Un ejemplo es el CPG que se puede apreciar en la siguiente imagen (Figura 3.6) y que se diseñó para controlar un robot acuático imitando a una lamprea [67]:

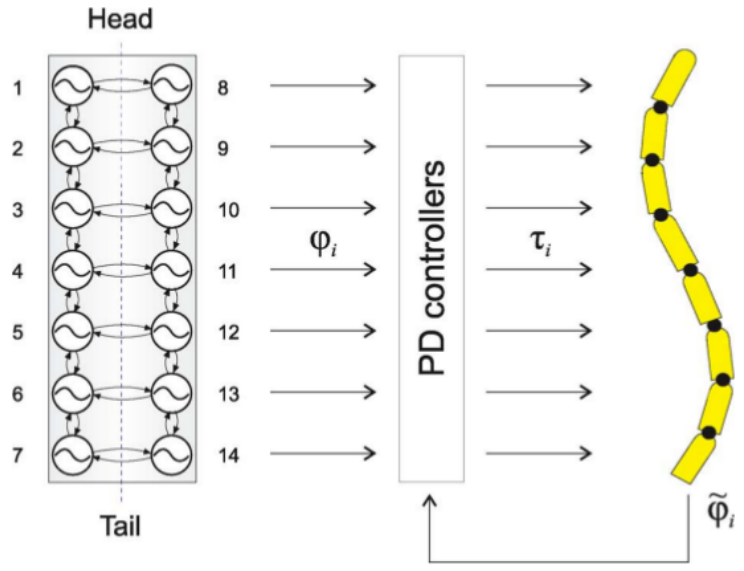


Figura 3.6: Robot que mediante el uso de CPGs imita el movimiento de una lamprea

Los CPGs se usan para controlar una gran variedad de tipos de robots o diferentes modos de locomoción. Por ejemplo, han sido utilizados para controlar el movimiento de:

- Robots hexápodos y octópodos inspirados en la locomoción de los insectos [68].
- Robots lamprea que imitan la forma de nadar de estos animales, utilizando una onda que viaja por el cuerpo desde la cabeza a la cola [67].
- Robots que imitan a la serpientes terrestres [69].
- Robots cuadrúpedos donde han encontrado que el feedback sensorial que modula la actividad del CPG trata de hacer que este llegue a una locomoción estable en terrenos complejos [70].
- Robots salamandra, los cuales han utilizado para estudiar la transición que realizan al cambiar entre andar y nadar [71].
- También son cada vez más usados para controlar la locomoción bípeda en robots humanoides [72].
- Robot Cube Revolutions controlado por CPGs con sinapsis químicas implementado en el GNB [2, 3].
- Robot con movilidad reducida controlado por CPGs e implementado en el GNB [42].

3.7.1. Propiedades de los CPGs

Hay diversas propiedades que hacen que los modelos basados en CPGs sean útiles para controlar la locomoción en los robots como una alternativa a los métodos basados en maquinas de estados finitos, generadores de senos o trayectorias previamente generadas [47]. Estas propiedades son:

1. Son capaces de producir patrones rítmicos estables. El sistema es capaz de recuperar rápidamente su comportamiento rítmico normal tras alteraciones transitorias en sus variables. Esto provee de robustez ante perturbaciones.

2. Los CPGs son muy apropiados para una implementación distribuida, lo que puede ser interesante para robots modulares, como robots serpiente, y para robots reconfigurables.

3. Los modelos típicamente tienen unos pocos parámetros de control que permiten la modulación de la locomoción, como por ejemplo modificar la velocidad o dirección, o incluso la manera de moverse. Un modelo de CPG convenientemente implementado reduce por tanto la dimensión del problema de control al hacer que los controladores de alto nivel no tengan que producir directamente multitud de comandos motrices, sino únicamente señales de control de alto nivel. Relacionado con esto, los CPGs son capaces de producir modulaciones fluidas de las trayectorias generadas aún cuando los parámetros de control son cambiados abruptamente. Esta propiedad es útil para la generación de trayectorias en tiempo real ya que evita el posible daño en motores debido a cambios abruptos en las señales de control motoras.

4. Los CPGs son adecuados para integrar señales sensoriales de feedback (que se pueden añadir de manera simple en las ecuaciones diferenciales del modelo neuronal). Esto genera la oportunidad de conectar a todos los niveles el CPG con el cuerpo mecánico.

5. Los CPGs ofrecen de manera usual un buen punto de comienzo para los algoritmos de aprendizaje y lo algoritmos de optimización.

3.7.2. Metodología de diseño para CPGs

Cuando construimos un modelo de CPG, debemos definir los siguientes elementos:

1. La arquitectura general del CPG. Esto incluye el tipo y número de osciladores o neuronas.

2. El tipo y topología de los acoplamientos. Esto determinará las condiciones para conseguir un acoplamiento de fase entre osciladores y los resultantes movimientos que produzcan desplazamiento. Para ello se necesitará una fase estable entre los osciladores.

3. La forma de las ondas. Esto determinará que trayectorias realizará cada articulación durante un ciclo. La forma de las ondas son claramente dependientes de la forma producida en el ciclo límite por el oscilador neuronal elegido. La forma puede ser modificada mediante la introducción de filtros.

4. El efecto de las señales de entrada. Como los parámetros de control pueden modular cosas tan importantes como la frecuencia, amplitud, fases entre osciladores y la forma de las ondas.

5. El efecto de las señales de feedback. Las señales de feedback provenientes del mismo cuerpo pueden afectar la actividad del CPG, como puede ser acelerar o decelerar en función de las condiciones del entorno.

3.7.3. Feedback en CPGs

Los CPGs reciben información del desempeño que está realizando el organismo al utilizar los comandos generados por el Sistema Neuronal. Es decir, reciben información de como está cumpliendo las órdenes el organismo. Esta información que reciben se denomina feedback sensorial.

Mientras que el feedback no es necesario para la generación de ritmos, juega un papel muy importante en la forma de los ritmos generados. Esto es fundamental para mantener el CPG y los movimientos del cuerpo coordinados entre sí.

Para demostrar que no es necesaria una señal que controle el CPG para la generación de ritmos, se realizó un experimento [21]. Se quitó el sistema nervioso de una langosta y se introdujo en un recipiente con solución salina fisiológica. Bajo estas condiciones no hay caminos sensoriales y por tanto no se introduce información externa al CPG. Se pudo observar que el ritmo se seguía produciendo, y que era muy similar al obtenido cuando el CPG estaba aún dentro del animal (Véase Figura 3.7). Los ritmos que generan estas muestras se denominan patrones motores ficticios, patrones motores que se encargarían de manejar los movimientos de los músculos si estos estuvieran conectados.

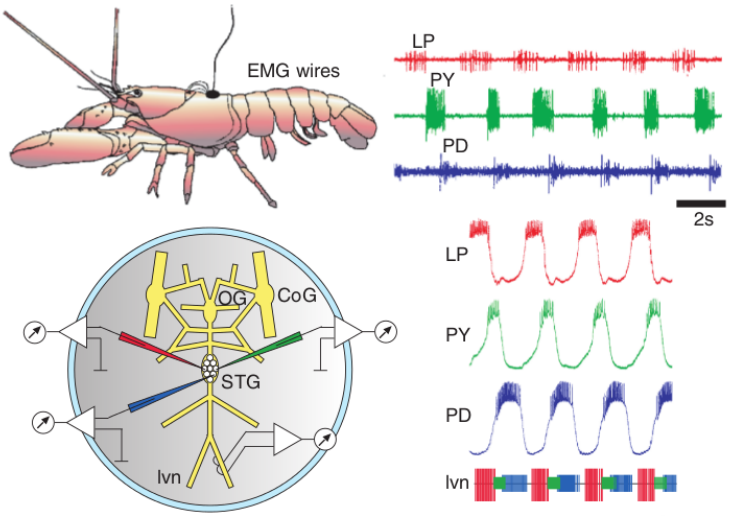


Figura 3.7: Experimento con un CPG perteneciente al sistema digestivo de una langosta

En la imagen anterior se pueden apreciar los resultados del experimento, observándose cómo

se obtienen los mismos resultados con el animal vivo (señales de la parte superior) que cuando se obtiene la muestra in vitro (señales de la parte inferior).

En este proyecto se va a utilizar feedback sensorial introduciendo la Nariz Electrónica en el modelo y modulando el comportamiento del CPG con el valor medido por el sensor. Aún así, sería interesante introducir nuevos feedbacks sensoriales en futuros trabajos que partan de lo que plantea esta primera fase para, por ejemplo, poder adaptarse dinámicamente al medio o a funcionamientos erróneos en los servo-motores del robot.

3.8. Reglas básicas en la construcción del CPG

A partir del estudio de la sinapsis y de los principios presentes en los CPGs biológicos se infieren una serie de reglas básicas a la hora de desarrollar un CPG con un diseño bioinspirado. Por tanto, debemos cumplirlas de manera rigurosa en nuestro diseño. Estas reglas son:

3.8.1. Principio de localidad

No todas las conexiones son posibles, pues en organismos reales, las neuronas están conectadas en gran medida con sus vecinas y, por tanto, conexiones entre neuronas muy separadas o correspondientes a subsistemas distintos no tienen sentido. Una solución muy fácil en este proyecto para conseguir determinados movimientos del robot sería conectar la neurona que controla el primer módulo, correspondiente a la cabeza del gusano, con el último módulo, correspondiente a la cola, pero es descartada por no cumplir con el principio de localidad para el diseño modular (explicación detallada de la correspondencia entre las neuronas y los módulos en el Apartado 5.5).

3.8.2. Escalas temporales

El CPG trabaja con varias escalas temporales. Se emplean pulsos muy rápidos a modo de disparo que emulan los potenciales de acción de las neuronas junto con ondas depolarizantes lentas que se encargan de establecer la activación y desactivación de los pulsos rápidos (Modelo Spiking-Bursting). Se ha demostrado que esta combinación resulta en una rica negociación entre las neuronas, lo que revierte en la generación de ritmos flexibles y a la vez robustos en el CPG [2]. Adicionalmente, cuando se desee cambiar de movimiento habrá que cambiar los parámetros de la sinapsis, teniéndose así un periodo de transición entre un movimiento y otro. Este cambio de los parámetros de la sinapsis se asemeja al cambio del modo de funcionamiento en organismos biológicos, provocado por la acción de los neuromoduladores [7, 21].

3.8.3. Competencia: Winnerless Competition

El mecanismo por el cual muchos tipos de neuronas se relacionan entre sí no es más que la competencia mediante coactividad inhibitoria asimétrica para generar ritmos. Esto ocurre en la mayor parte de los CPGs conocidos [8, 19].

3.9. Narices electrónicas

Una nariz electrónica no se trata más que de un sensor especializado en la captación de odorantes. Por tanto se va a hacer una pequeña introducción a los sensores electrónicos para poder entender mejor su funcionamiento.

3.9.1. Sensores electrónicos

Un sensor es cualquier dispositivo que detecta una determinada acción externa. Cualquier animal utiliza sensores biológicos para percibir su entorno. El hombre experimenta sensaciones como calor o frío, duro o blando, fuerte o flojo, agradable o desagradable, pesado o ligero. Poco a poco le ha ido añadiendo adjetivos a estas sensaciones para cuantificarlas como frígido, fresco, tibio, templado, caliente, tórrido. Es decir, ha ido necesitando el empleo de magnitudes medibles más exactas ante una percepción más clara del entorno [73].

En general se habla de sensores, pero se pueden distinguir los siguientes elementos:

- **Sensor**: Dispositivo que recibe una señal o estímulo y responde con una señal eléctrica. Pueden ser de dos tipos:
  - **Activo**: Requiere una fuente externa de excitación o células de carga, como por ejemplo las RTD (Detector de Temperatura Resistivo).
  - **Pasivo**: Sensor que no requiere una fuente externa de excitación, como los termopares o fotodiodos.
- **Transductor**: Convertidor de un tipo de energía a otra.

Existe una gran cantidad de sensores en el mercado para poder medir magnitudes físicas. De los más enfocados a medir características biológicas se pueden destacar los siguientes:

- Temperatura.
- Humedad.
- Presión.
- Luminosidad.
- Conductividad.
- Multipropósito.

La Nariz Electrónica que se va a utilizar se ocupa de medir la conductividad de las partículas que conforman el odorante presente en el medio. El funcionamiento específico del sensor se encuentra en el Apartado 4.6.

3.9.2. Nariz electrónica artificial

Ahora que ya se conoce por un lado como funciona el Sistema Olfativo y por otro como funcionan los Sensores Electrónicos, se va a proceder a juntar ambos conocimientos para poder describir el funcionamiento genérico de una Nariz Electrónica.

Los sistemas olfativos artificiales tratan de reproducir la capacidad de lectura e interpretación de los sistemas olfativos que se pueden encontrar en los animales y humanos. Las narices artificiales (NA) son máquinas que están diseñadas para detectar y discriminar con precisión distintos olores (véase Figura 3.8).

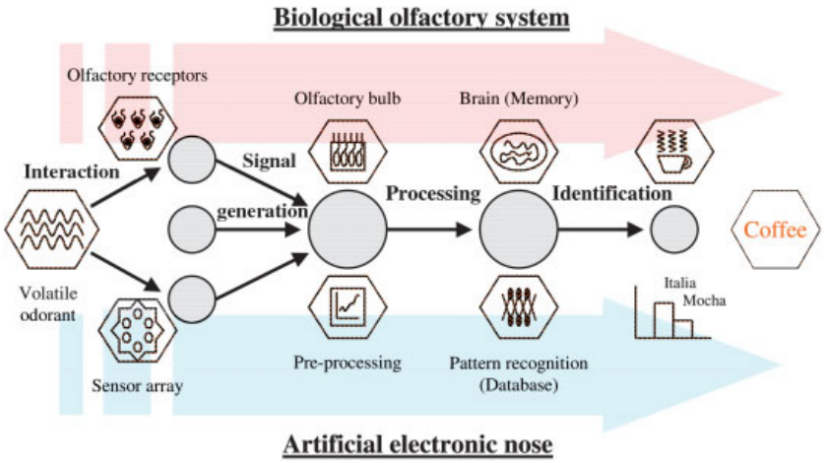


Figura 3.8: Procesos llevados a cabo por una Nariz Electrónica comparados con los procesos que realiza el sistema olfativo biológico.

En la anterior imagen se puede apreciar cual es la secuencia de procesos que identifica a una EN, comparando en todo momento esta actuación con la que se realiza en un sistema biológico. Se comienza realizando una medida con el sensor o array de sensores, generando una señal de entrada al sistema. Se procesa esta señal y se compara con los valores almacenados en la base de datos, buscando coincidencias con olores previamente almacenados. Si existe un olor con el que coincida en gran medida se identifica como tal odorante.

Las Narices Electrónicas han tenido un fuerte desarrollo durante los años 90 y tienen múltiples utilidades en la industria moderna, por tanto se vislumbra una enorme cantidad de potenciales aplicaciones en diferentes sectores:

- Medicina y farmacia.
- Industria química.
- Industria militar.
- Alimentación.
- Medio ambiente.

Los elementos directamente responsable de la adquisición de los olores son los sensores que convierten las concentraciones químicas en señales eléctricas. Hay varios tipos de sensores que nos permiten realizar este cometido:

- Quimioresistores.
- Quimiocapacitores.
- Quimiosensores potenciométricos.
- Quimiosensores gravimétricos.
- Quimiosensores ópticos.
- Quimiosensores térmicos.
- Quimiosensores amperimétricos.

Hay un amplio número de aspectos, muchos de ellos de inspiración biológica, que permiten enriquecer la información adquirida por este tipo de equipos [4]. Con el objetivo de optimizar su funcionamiento existen ya estudios que evalúan la mejora producida gracias al uso de mucosas artificiales, el enriquecimiento que se obtiene al introducir los sensores en cavidades olfativas con la correspondiente interferencia producida por su geometría, el uso de una cierta dinámica térmica o de presión y la caracterización del proceso de inhalación e incluso de exhalación.

Como ya se ha comentado en el apartado anterior, el sensor que vamos a utilizar se trata de un quimiosensor capacitivo. En el siguiente capítulo se explicarán todos los detalles de Olus2 [4, 6], nombre de la nariz creada por David Yañez y que se va a utilizar en este proyecto.

3.9.3. Robots móviles con capacidad olfativa

Olfatear juega un papel muy importante en las labores de orientación en los animales, y la supervivencia de muchas especies de animales depende de esta habilidad. Buscar una mayor o menor concentración de un componente químico es uno de los modos de comportamiento más antiguos y se puede encontrar incluso en microorganismos. Para algunas especies de animales, el olfato es más efectivo que otros sentidos como pueden ser la vista o el oído en la búsqueda de comida o del nido [74].

El olfato también es usado en varios tipos de comunicación que implican el uso de feromonas. Famosos ejemplos son las hormigas que gracias a sus antenas siguen las marcas de feromonas que se encuentran en el suelo o las polillas que siguen las plumas de feromonas por el aire.

Inspirado en estos comportamientos, se han desarrollado sistemas robóticos que tratan de seguir y buscar plumas de odorante [75, 25]. Hay dos tipos de sistemas que se han ido desarrollando. Uno es seguir pistas de odorante marcadas en el suelo, imitando el comportamiento de las hormigas. El otro tipo se trata de seguir plumas de odorante de manera aérea o bien submarina para encontrar la fuente de origen del odorante, como hacen mamíferos e insectos.

En este proyecto se busca desarrollar un robot que sea capaz de captar odorantes en el medio en el que se encuentra y que en función de esas medidas tome decisiones en consecuencia. No se trata de buscar la fuente de odorante, ya que es algo realmente complicado y aún no resuelto, pero si de responder ante estímulos recibidos por la Nariz Electrónica.

3.10. Microcontroladores

En este proyecto se van a utilizar microcontroladores para dos cometidos. La Nariz Electrónica incorpora uno para poder procesar la señal recibida y poder transmitirla a otros elementos. Por otro lado, el robot está controlado por otro microcontrolador que además tendrá que recibir la señal captada por la nariz.

Un microcontrolador (abreviado μC) es un circuito integrado programable capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.

Los microcontroladores son diseñados para reducir el costo económico y el consumo de energía de un sistema en particular. Por eso el tamaño de la unidad central de procesamiento, la cantidad de memoria y los periféricos incluidos dependerán de la aplicación a la que se vaya a destinar.

Al ser fabricados, la memoria ROM del microcontrolador no posee datos. Para que pueda controlar algún proceso es necesario generar algún programa y grabarlo en su memoria. Para generar el programa se necesitará un PC en el que escribirlo en lenguaje ensamblador u otro lenguaje que acepte el compilador para así generar el fichero en hexadecimal necesario. Posteriormente, este código se introducirá al microcontrolador mediante un programador.

Existen muchos tipos de microcontroladores pero hay dos que destacan sobre todos los demás:

- **PIC:** Fabricados por la compañía de semiconductores Microchip [76], que es la compañía dominante del mercado. El primer modelo se desarrolló en 1975 y se trata del microcontrolador con mayor cantidad de información disponible. Poseen una relativamente pequeña cantidad de espacio de datos direccionable (típicamente 256 Bytes).

- **AVR:** Familia de microcontroladores RISC del fabricante estadounidense Atmel [77]. Cuenta con una gran aceptación debido a su diseño simple y facilidad de programación, ya que fue diseñado desde un comienzo para la ejecución eficiente de código C compilado. Estos microcontroladores están soportados por tarjetas de desarrollo de costo razonable, capaces de descargar el código al microcontrolador y por una versión de las herramientas GNU [78]. Esto es posible por su uniformidad en el acceso al espacio de memoria (propiedad que carecen los PIC). Debido a esta uniformidad, la memoria de datos no está limitada a un tamaño reducido y fijo.

En este proyecto se comienza utilizando dos microcontroladores PIC, el incluido en la Nariz Electrónica (PIC18L4550) y el que controla el robot (PIC16F876). Sin embargo, mostraremos que el microcontrolador encargado del robot no va a ser capaz de realizar las tareas necesarias. En proyectos anteriores del GNB se utilizaba únicamente como cliente ligero [2, 3, 42], ejecutándose la simulación en el PC y siendo el microcontrolador el encargado de recibir los datos y en función de los valores posicionar correctamente a los servos del robot. En cambio, ahora se va a realizar la simulación en tiempo real en el microcontrolador. Para ello se necesitará una mayor cantidad de memoria, tanto para almacenar las variables como para introducir las líneas de código que conforman el modelo neuronal. Debido a esto (y a la mayor facilidad de poder utilizar software libre), se considera a los microcontroladores AVR como la mejor opción para realizar la simulación neuronal y controlar el robot.

Dentro de la familia AVR existe un plataforma muy interesante, Arduino [79]. Se trata de una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Por otro lado el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Wiring (muy similar al lenguaje C) [80] y el boot loader (cargador de arranque) que corre en la placa.

El formato de presentación de Arduino es:

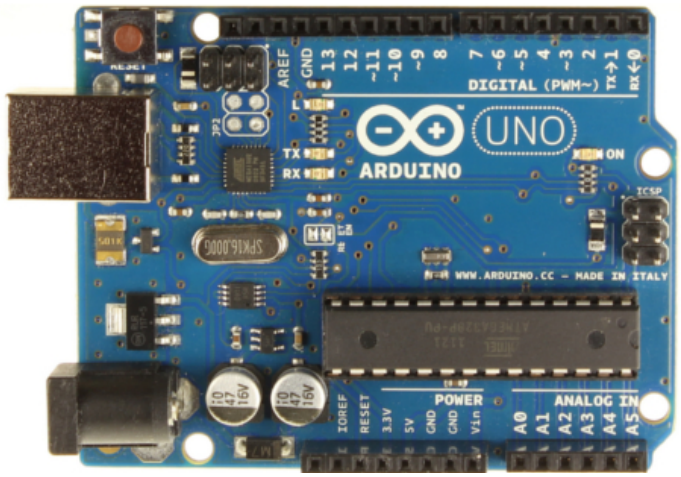


Figura 3.9: Placa Arduino de desarrollo de hardware libre

En caso de hacer uso de los microcontroladores Atmel se utilizará la plataforma Arduino ya que facilita muchas labores a la hora de integrar el código del modelo neuronal gracias a la generosa biblioteca de funciones de control existente en internet [79]. Aparte, Juan González, creador del robot que se va a utilizar en este proyecto ha diseñado una placa basada en Arduino, SkyMega [81], y que se integra en la estructura del robot. Sería interesante utilizar SkyMega para así poder dar mayor libertad al robot a la hora de realizar movimientos sin entorpecerlo con las conexiones de los elementos. En el Apartado 5.6 se muestran detalles más específicos de

estos microcontroladores y sobre la decisión que se debe tomar para poder controlar el robot.

3.11. Comunicación I2C

A la hora de conectar el microcontrolador encargado de obtener la señal de odorante con el microcontrolador encargado de mover al robot se va a utilizar la comunicación I2C [82]. Se trata de un bus de comunicaciones serie diseñado por Philips en 1992. Su principal uso es comunicar microcontroladores y sus periféricos en sistemas integrados, y también para comunicar circuitos integrados entre sí que normalmente residen en un mismo circuito.

La estructura básica del bus se puede apreciar en la siguiente imagen:

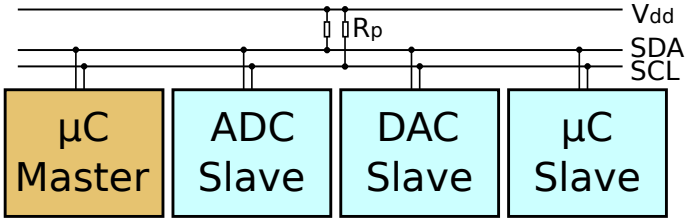


Figura 3.10: Estructura del bus I2C (No incluye ni alimentación ni masa de los dispositivos)

La comunicación se realiza únicamente a través de dos cables. Se trata de una comunicación serie, utilizando un cable para transmitir los datos (SDA) y otro para enviar los pulsos de reloj que sincronizan a los elementos (SCL). Incluye unas resistencias de PULL-UP necesarias para mantener las líneas SDA y SCL a "1.[en] su estado normal.

En la comunicación I2C aparecen dos roles:

- **Maestro (Master):** Dispositivo que determina la temporización y la dirección del tráfico de datos en el bus. Es el único que aplica los pulsos de reloj en la línea SCL. Cuando se conectan varios dispositivos maestros (para, por ejemplo, reconfigurar la comunicación ante el fallo del maestro) a un mismo bus la configuración obtenida se denomina multi-maestro.
- **Esclavo (Slave):** Cualquier dispositivo conectado al bus incapaz de generar pulsos de reloj. Reciben señales de comando y de reloj provenientes del dispositivo maestro.

Por otro lado, las características que definen a la comunicación I2C son las siguientes:

- **Se trata de un bus serie síncrono.** El término síncrono comprende que, además de la línea de datos (SDA), hay una señal de sincronía (SCL) explícita para validar los datos en el canal serie.
- **El sentido del enlace es semibidireccional.** Es decir, existe una única línea de datos que puede utilizarse para el flujo en ambos sentidos, pero no simultáneamente. De esta manera se ahorra el número de señales del bus. Esta limitación es un hecho menor debido al ámbito de aplicación de este tipo de bus, en el que no se necesita la bidireccionalidad.
- **Admite topologías multipunto.** Es decir, pueden conectarse al bus varios dispositivos esclavos, pudiendo actuar varios de ellos como emisores, aunque no simultáneamente. El maestro les va cediendo el turno para transmitir.

- **Un mismo dispositivo puede actuar como emisor o como receptor** en distintos momentos.
- **Admite topologías multimaestro.** Es decir, más de un dispositivo puede intentar gobernar el bus. Un maestro puede actuar tanto como emisor como receptor; y lo mismo puede decirse de un esclavo. Lo que diferencia a un maestro de un esclavo es la capacidad de gobierno del bus (se encarga de suministrar la señal de sincronía).
- **Establece un mecanismo de arbitraje** para el caso en que simultáneamente más de un maestro intente gobernar el bus.
- **Un maestro puede funcionar también como un esclavo.** En las topologías multimaestro un maestro que haya ganado el gobierno del bus puede dirigirse a cualquier otro maestro, que entonces deberá comportarse como un esclavo.
- **Establece un mecanismo de adaptación de velocidad**, para el caso en que un esclavo no pueda seguir la velocidad impuesta por el maestro.
- **Establece un criterio de direccionamiento abierto** por medio del cual se sabe a qué esclavo se dirige un maestro. Además, la dirección asignada a un circuito puede ser modificada dinámicamente. Por otra parte, el mecanismo de direccionamiento admite futuras ampliaciones del número máximo de elementos direccionables.
- **El formato de las tramas transmitidas es suficientemente flexible** como para poder adaptarse a la funcionalidad de cualquier tipo de circuito. Hasta que no se finaliza la transmisión se pueden mandar todas las tramas que sean necesarias para transmitir los datos.
- **Desde el punto de vista de la capa física de la interfaz, admite la conexión de circuitos de diferentes tecnologías** (Bipolares, MOS, etc.).

En el Apartado 4.7 se describe en detalle el funcionamiento de la comunicación I2C, explicando cómo se establece y finaliza la comunicación, cómo se debe hacer para mantener el flujo de datos y la configuración de los dispositivos Maestro y Esclavo. Más adelante, en los Apartados 5.7.2 y 5.7.3, se desarrolla el código necesario para disponer de esta comunicación entre el Robot y la Nariz Electrónica.

4
Diseño

4.1. Introducción del robot

El robot diseñado por Juan González [1] se trata de un robot modular con forma de gusano compuesto por 8 módulos idénticos (véase Figura 4.1). Es capaz de desplazarse mediante ondulaciones de su cuerpo, que se propagan desde la cola hasta la cabeza. En este proyecto se reutiliza el robot construido por Damián Zamorano para su Proyecto Final de Carrera [3], por lo que no se construirá pero si que se realizarán cambios en el robot.

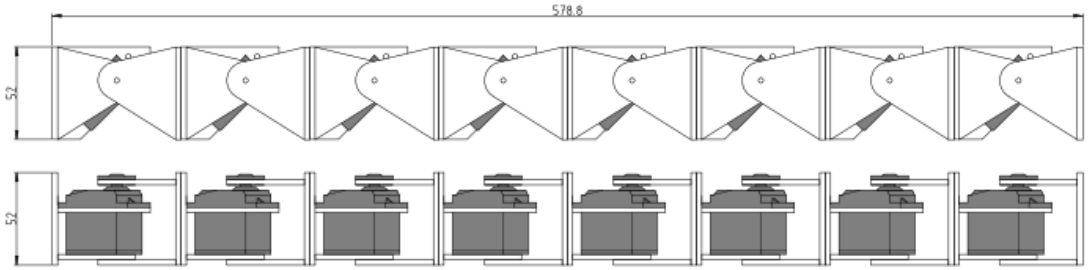


Figura 4.1: Descripción Cube Revolutions

Cada módulo está formado por piezas de metacrilato unidas con pegamento de contacto, un servo-motor Futaba S3003 y los tornillos necesarios para unir el motor a la estructura [83]. Las piezas de metacrilato hay que pedirlas por encargo a alguna empresa que fabrique piezas en este material a partir de los planos que le entregue el cliente. En este caso, las piezas del robot fueron encargadas al SEGAINVEX, Servicio de Apoyo a la Investigación Experimental de la UAM. Los servo-motores se pueden comprar en tiendas de aeromodelismo, y los tornillos y elementos de conexionado en ferreterías y tiendas de electrónica. Las piezas de cada módulo están diseñadas para que los motores Futaba encajen perfectamente, por lo tanto, una vez unidos cuesta separarlos. Antes de encajar los motores en las piezas, habrá que conectar cada servo al microcontrolador para situarlos en la posición central, para que así tengan el mismo recorrido para un lado que para el otro.

La configuración de los módulos diseñados por Juan González puede ser distinta en función de su orientación:

- **Cabeceo-Cabeceo:** Los módulos se desplazan únicamente en el plano x.

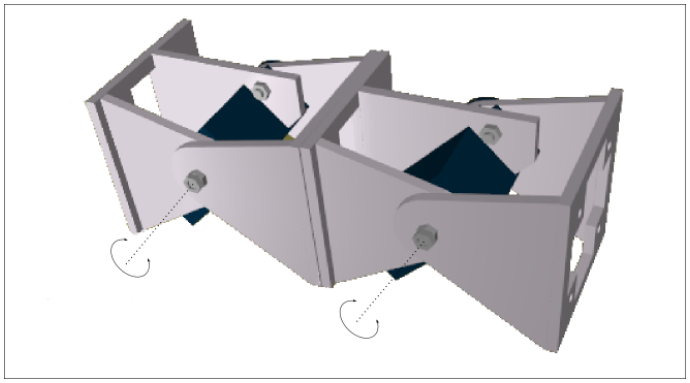


Figura 4.2: Configuración Cabeceo-Cabeceo

- **Cabeceo-Viraje:** Los módulos se desplazan tanto en el plano x cómo en el plano y.

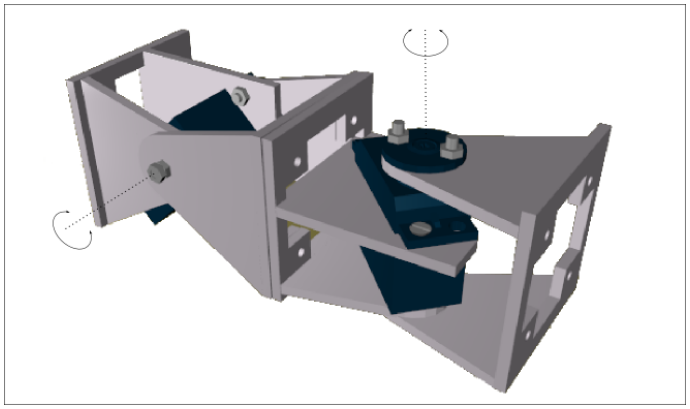


Figura 4.3: Configuración Cabeceo-Viraje

- **Viraje-Viraje:** Similar a la configuración Cabeceo-Cabeceo pero con los módulos girando en el plano y en vez del x. Se realiza tumbando de lado la configuración Cabeceo-Cabeceo.

Se utiliza el robot fabricado por Damián Zamorano pero se cambia la configuración de los módulos que el dispuso. Fabricó 4 módulos blancos y 4 negros, que fue conectando intercalados. Cada color correspondía a una de las dos orientaciones, el blanco al cabeceo y el negro al viraje, y gracias a esta distribución era capaz de visualizar más fácilmente el desempeño de cada uno de los módulos. Se seguirá utilizando la disposición de colores que el propuso pero se cambiará la orientación de los módulos para posicionarlos a todos en una configuración de Cabeceo-Cabeceo.

En un principio se diseñará el movimiento del robot para la configuración descrita, pero cuando ese movimiento esté controlado se probará a cambiar la orientación de varios de los módulos. Así se podrá conseguir que el robot además de desplazarse en línea recta, sea capaz de realizar giros a izquierda y derecha.

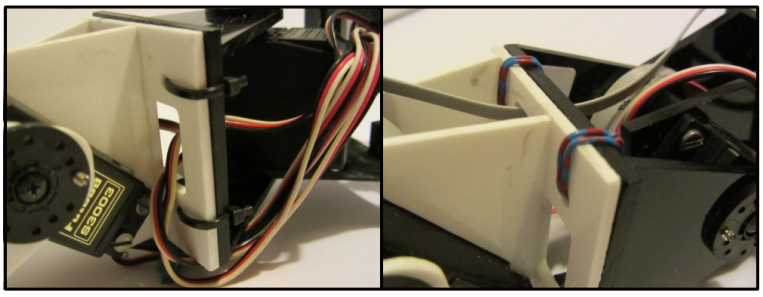
El siguiente paso consiste en medir el error de cada módulo. Aunque se intentó montar las piezas ajustando los servos a su posición central, al manipularlos es normal que se desplacen y aparezca cierto error. Se mide el error de los distintos módulos, probando valores en el robot mediante el programa star-servos8 (en el Apartado 5.6.1) hasta observar al robot totalmente horizontal respecto al suelo. Se obtiene como resultado los siguientes valores:

Módulo	Error
1	22º
2	30º
3	23º
4	25º
5	29º
6	23º
7	25º
8	29º

Cuadro 4.1: Ángulo de corrección en los módulos

Al presentar este error, los módulos no son capaces de girar 90 grados en ambos sentidos (el ángulo de giro de los servos es de 180 grados, 90 a cada lado del centro de giro) ya que su punto central está desplazado. Al no estar centrado el módulo podrá girar más hacia uno de los lados que el otro pero en el robot no se desea que esto ocurra, sino que se debe mover con el mismo ángulo máximo en ambos sentidos. Como el ángulo máximo que se necesita en el robot es de 40 grados y el error máximo que se presenta en la tabla anterior es de 30 grados (limitando el giro en uno de los dos sentidos a un valor máximo de 60 grados), no significa un problema la presencia de estos errores. Para que este error no afecte al correcto funcionamiento del robot, en el programa que se encargue de generar el movimiento se procederá a corregirlo mediante una simple adición o sustracción del término obtenido en cada módulo, y así hacer que los módulos se encuentren en su posición central u oscilen de manera controlada en torno a él cuándo así se desee (véase como se ha realizado en el Apartado 5.6.2).

Una vez montados todos los módulos, etiquetados con su numeración correspondiente y medido su error, se procede a la conexión entre ellos. En montajes anteriores se utilizaron tornillos o bridas para enganchar un módulo a otro. En esta ocasión, al no tener disponible bridas del tamaño adecuado, se probó la interconexión de los módulos mediante un cable anudado (nudo Prusik) y se observó que los módulos se mantenían totalmente unidos sin dar ninguna muestra de debilidad. Al obtener buenos resultados, ser un método barato (se utilizaron cables viejos disponibles en el laboratorio) y ser fácil la labor de desmontaje y montaje, y así poder probar otras configuraciones, se eligió como el método de conexionado definitivo en este proyecto. La unión de los módulos mediante estos métodos se puede apreciar en la siguiente imagen (Figura 4.4):



Bridas **Nudo Prusik**

Figura 4.4: Métodos de interconexión de los módulos

Para no tener los cables sueltos de cada uno de los servos, Damián Zamorano construyó dos placas PCB (Printed Circuit Board) que agrupaban los conectores, que dependiendo de si los módulos están entre los cuatro primeros o los cuatro últimos van conectados a un PCB o al otro. Estos PCB están conectados entre sí y también al microcontrolador mediante un bus compuesto por 10 cables, ya que se necesita la alimentación positiva, la tierra de referencia y 8 señales de control, una por cada servo-motor. La distribución de las matrices de los PCB es la siguiente (véase Figura 4.5):

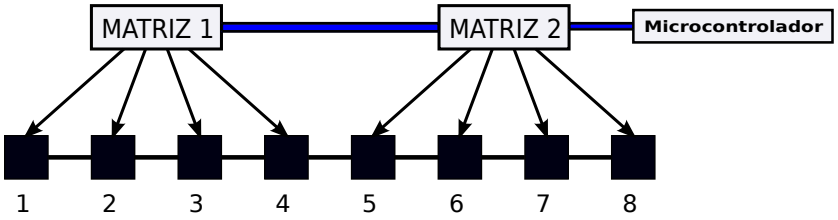


Figura 4.6: Distribución de señales del control locomotor

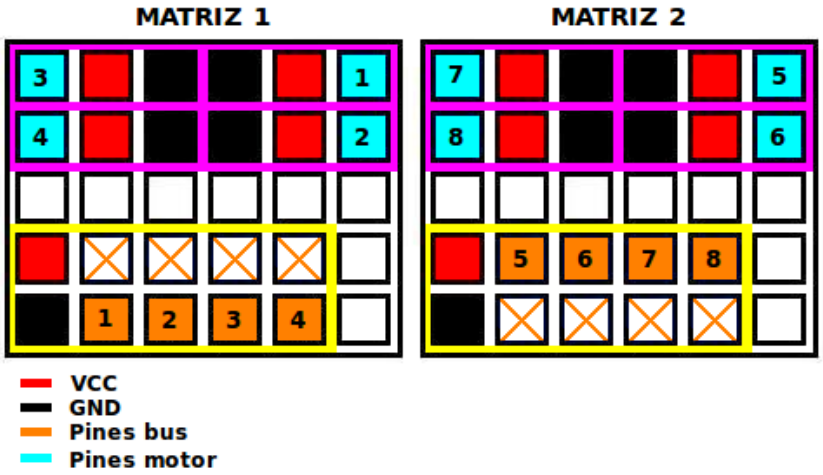


Figura 4.5: Matrices de interconexión de los motores

En la imagen se puede apreciar el conexionado que se realizó. Los pines naranjas corresponden a la conexión del bus que proviene del microcontrolador con el PCB, y los pines azules están conectados a la señal de control del servo-motor. Cada pin azul está soldado con su correspondiente pin naranja en la cara posterior de la matriz. Los pines VCC y GND también están conectados todos entre sí para así distribuir la alimentación que necesitan los servos para funcionar. Por tanto, los pines englobados en el rectángulo amarillo están directamente conectados al bus que va hasta el microcontrolador. Por otro lado, los pines englobados en los rectángulos de color violeta corresponden a los 8 servo-motores, que tienen conectores rectangulares con tres pines (GND, VCC y señal de control). La primera matriz se encarga de llevar las señales a los primeros cuatro módulos y la segunda matriz a los cuatro restantes (véase Figura 4.6).

Las matrices 1 y 2 están conectadas al microcontrolador mediante el bus. En la matriz 1, las salidas que se encargan de controlar a los motores 5, 6, 7 y 8 no están conectadas ya que no se utilizarán. En la matriz 2 ocurre lo mismo con las señales de control de los motores 1, 2, 3 y 4. En el esquema (véase Figura 4.7) se puede apreciar la distribución de las señales de control en el bus. Una vez soldadas estas matrices, fueron recubiertas con una silicona protectora para que así aguantasen mejor los movimientos mecánicos a la que son expuestas.

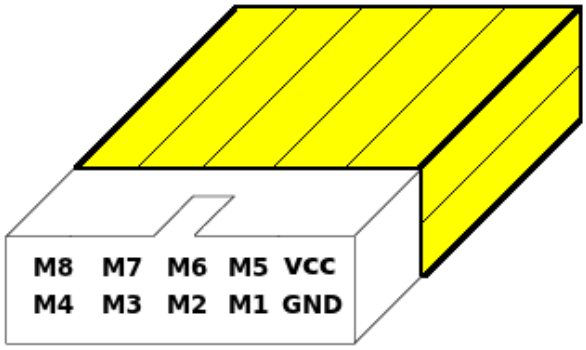


Figura 4.7: Distribución de pines en el bus

Un problema que se apreció al probar el movimiento de los servo-motores con el robot al completo es que los motores no respondían con suficiente fuerza. Esto se resolvió acortando el bus y así eliminando el ruido y disipación producidos en el mismo. El bus ahora posee dos conectores, siendo el más largo el que no ofrece el voltaje necesario para un correcto funcionamiento, y teniendo que utilizar el conector más corto para así conseguir que el robot se desplace.

La alimentación necesaria para que los motores funcionen correctamente es de 5 voltios. El consumo que ofrece varía de 80 mA (10 mA por motor) cuando el robot se encuentra en reposo a 1,6 A (0,2 A por motor) cuando el robot se encuentra en movimiento. La alimentación se realizará a través de una fuente de alimentación que transforma la corriente disponible en la red eléctrica para ofrecer una tensión de salida de 5 voltios y poniendo a disposición 3 Amperios, valores suficientes para conseguir que el robot funcione correctamente. Tanto la alimentación como la electrónica se encuentran situadas fuera del robot. Esto impedirá que el robot pueda desplazarse libremente, ya que sólo podrá alejarse lo que le permita la longitud del bus. En este proyecto se propone y realiza en parte la corrección de estos detalles para que haya una libertad de movimiento mayor (realizado en el Apartado 5.6.3).

Control locomotor

El control del robot lo realizará el microcontrolador íntegramente (el microcontrolador será uno de los presentados en el Apartado 3.10). En anteriores versiones del robot, las secuencias de locomoción eran generadas en el ordenador y almacenadas en un fichero. Posteriormente, este fichero era leído por un programa que se encargaba de ir pasando mediante el puerto serie al microcontrolador la información necesaria para posicionar los servos y generar movimiento en el robot. Es decir, los movimientos del robot estaban generados previamente. Se quiere que esta función la haga el microcontrolador para así generar el movimiento en tiempo real, lo cual tiene varias ventajas:

- No depender del funcionamiento de otro sistema para su funcionamiento, es decir, conseguir mayor autonomía.
- La duración del movimiento del robot no estará acotada al tamaño de la simulación realizada en el ordenador sino a la necesidad de movimiento que se requiera en el momento.
- Para grandes simulaciones se necesitaban grandes ficheros de almacenamiento de las variables del robot. Ahora podremos prescindir del almacenamiento de estos ficheros.
- Si el robot toma alguna decisión en función de los factores externos que le rodean, será capaz de cambiar el movimiento por sí mismo y no tendrá que pedirle al ordenador una nueva secuencia de locomoción. Tendrá la capacidad de variar los parámetros de la locomoción cuando así lo necesite.

Por tanto la introducción del modelo neuronal en el microcontrolador se trata de uno de los avances realizado sobre los anteriores trabajos que se basan en este robot. La elección del microcontrolador y la adaptación del código para que pueda ejecutarse en el mismo se puede leer en el Apartado 5.6.

Movimientos disponibles

Se pretende poder realizar dos tipos de movimientos para conseguir el desplazamiento del robot:

1. **Línea recta**

   Mediante el uso del CPG se pretende crear señales sinusoidales que permitan que el robot se desplace hacia delante o hacia atrás, es decir, en línea recta. Para la realización de este movimiento se precisa que los módulos se encuentren en modo Cabeceo-Cabeceo (en la Figura 4.2). No es necesario que todos los módulos se encuentren de esta manera sino que alguno se puede encontrar de la forma Cabeceo-Viraje (en la Figura 4.3) como ya hizo Damián Zamorano [3], aunque estos módulos deberán permanecerán centrados durante toda la secuencia de movimiento.

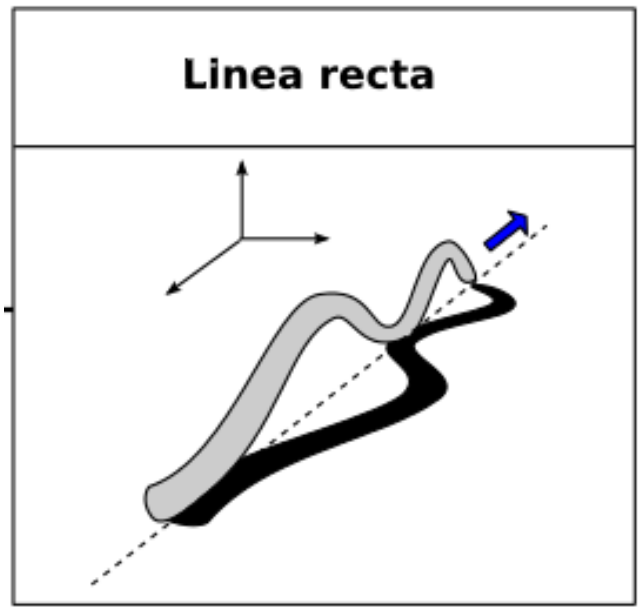


Figura 4.8: Movimiento del robot en línea recta

2. **Trayectoria circular**

   El desplazamiento en trayectoria circular emite al robot realizar giros de un determinado radio. El mecanismo de movimiento es similar al de la locomoción en línea recta pero el ángulo de doblaje de las articulaciones horizontales permanece fijo a un valor constante y distinto de cero por lo que el robot adopta una forma de arco circular. Es decir, este movimiento se produce cuando se superponen una curva circular en las articulaciones horizontales y una onda sinusoidal en las articulaciones verticales. La propagación de la onda sinusoidal por las articulaciones verticales del robot provoca el movimiento en un sentido u otro del arco. Este movimiento únicamente se puede utilizar si el robot combina módulos Cabeceo-Cabeceo (en la Figura 4.2) con módulos Cabeceo-Viraje (en la Figura 4.3) y los sitúa de manera alternada (en el Apartado 5.6.3 se puede observar la configuración necesaria para realizar este movimiento).

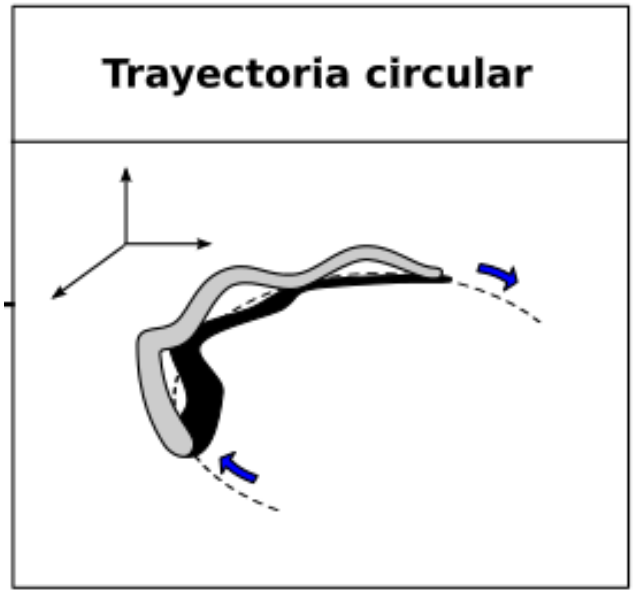


Figura 4.9: Movimiento del robot en una trayectoria circular

Estos movimientos que se han descrito se pueden realizar tanto con osciladores sinusoidales, como ya implementó Juan González [59], o bien mediante CPGs, que es lo que se realizó anteriormente por Fernando Herrero [2] y Damián Zamorano [3]. Para este proyecto también se pretende utilizar CPGs pero se quiere realizar de manera autónoma en un microcontrolador (desarrollado en el Apartado 5.6.2) y utilizando sinapsis eléctricas en vez de químicas para reducir la carga computacional introducida por el modelo (introducido en el Apartado 4.3). Se han probado los modelos creados por Juan González para así observar que el robot funciona de manera correcta y también para apreciar como se tienen que coordinar los distintos módulos para que se produzca movimiento y así poder replicarlo mediante los CPGs. Los resultados son los esperados, todo funciona de manera correcta, siendo hora de desarrollar los CPGs capaces de hacer actuar al robot de manera similar a lo observado.

4.2. Modelo neuronal de Rulkov

El modelo neuronal propuesto por Rulkov [44] presenta propiedades clave de las neuronas que se pueden encontrar en los organismos vivos: múltiples escalas temporales y diferentes regímenes de funcionamiento. Matemáticamente se trata de un modelo muy simple (por tanto ideal para introducirlo en un microcontrolador) y el posible conjunto de comportamientos puede ser controlado en función de la selección de sus parámetros. Tres regímenes estables pueden ser seleccionados mediante la combinación de sus parámetros (véase Figura 4.10): silencioso (a), en el que el potencial de la neurona se mantiene en un estado de descanso constante; disparo tónico (b), en el que la neurona emite disparos a una frecuencia constante; y ráfaga tónica (c), en el cual ráfagas de disparos son emitidos a una frecuencia constante, con un intervalo silencioso entre medias. Además, en los límites de las regiones paramétricas de estos regímenes, se puede encontrar un carácter caótico. Entre estos comportamientos, el que más nos interesa es la ráfaga tónica. Estos regímenes, tanto las oscilaciones intrínsecas como las silenciosas, se pueden observar en neuronas pertenecientes a CPGs de seres vivos. Mediante la combinación de ambas oscilaciones los seres vivos son capaces de generar ritmos fijos y estables para el movimiento de sus articulaciones, razón por la cual se quiere utilizar para la generación de las señales de locomoción en este proyecto (introducido en el Apartado 3.7 y desarrollado en el Apartado 4.5).

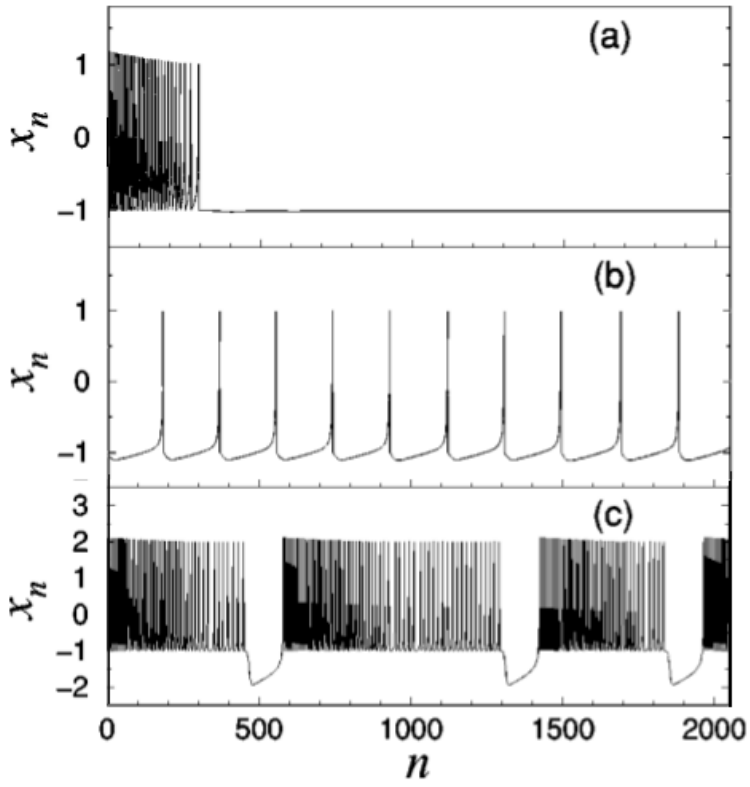


Figura 4.10: Modos de funcionamiento del modelo neuronal

El modelo de Rulkov, aún siendo bastante simple, es lo suficientemente complejo para mantener muchas de las propiedades que se pueden apreciar en las neuronas biológicas. El uso de un modelo con un mapa de bajas dimensiones, en este caso únicamente dos, puede ser útil para comprender los mecanismos dinámicos si son capaces de imitar las dinámicas en las oscilaciones observadas en neuronal reales, mostrar una correcta simulación del comportamiento colectivo, y es lo suficientemente sencillo como para poder estudiar las razones que se encuentran detrás de dicho comportamiento.

La descripción matemática del modelo de Rulkov se define mediante las siguientes ecuaciones:

$$x_{n+1} = f(x_n,\ y_n + \beta_e I_n) \tag{4.1}$$

$$y_{n+1} = y_n - \mu(x_n + 1) + \mu\sigma + \mu\sigma_e I_n \tag{4.2}$$

$$f(x,y) = \begin{cases} \frac{\alpha}{1-x} + y, & x \leq 0 \\ \alpha + y, & 0 \leq x < \alpha + 1 \\ -1, & x > \alpha + y \end{cases} \tag{4.3}$$

Este es un modelo bidimensional, donde la variable x_n representa el voltaje de la membrana de una neurona e y_n se trata de una variable de dinámica lenta sin un significado biológico directo, pero con un significado similar al de las variables de apertura en los modelos biológicos que representan la fracción de canales de iones abiertos en la célula. Mientras que el voltaje de la membrana oscila en una escala temporal rápida, representando disparos individuales de la neurona, la variable de dinámica lenta mantiene el ritmo del ciclo de las ráfagas, una especie de memoria contextual. No poseen unidades específicas, es decir, se pueden re-escalar arbitrariamente para hacer que se ajusten a los requerimientos del robot.

La combinación de σ y α selecciona el régimen de funcionamiento del modelo: silencioso, disparo tónico o ráfaga tónica (véase Figura 4.11). En el régimen de ráfagas, estos parámetros también controlan varias propiedades de la actividad neuronal.

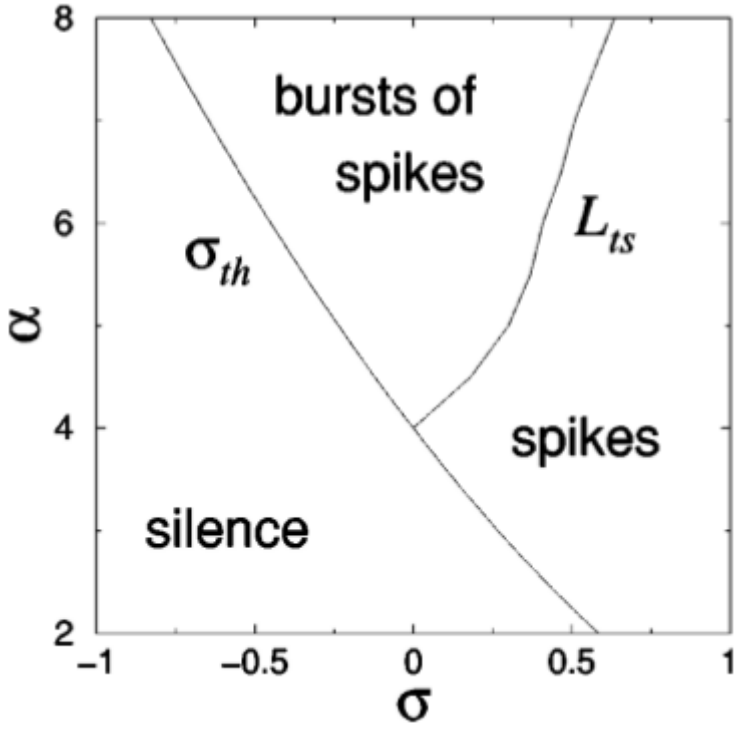


Figura 4.11: Regiones de funcionamiento del modelo neuronal

Cuando el valor de α es menor de 4 entonces, dependiendo del valor del parámetro σ, el mapa genera disparos o permanece en un estado silencioso. La frecuencia de los disparos aumenta según el valor de σ aumenta. Para α mayor que 4, las dinámicas del mapa son capaces de producir ráfagas de disparos. Los regímenes de ráfagas de disparos se pueden encontrar en la región intermedia del parámetro σ, entre el régimen de disparo tónico y el de estado silencioso. Los regímenes de ráfagas de disparos incluyen tanto ráfagas caóticas como periódicas. Las neuronas que conforman el CPG que se va a desarrollar en este proyecto deben situarse en el área paramétrica que permite producir ráfagas de disparos. Esto debe cumplirse ya que es mediante estas ráfagas como se consigue la señal de control de los servo-motores (en detalle en el Apartado 4.4).

Desde una perspectiva de control, es interesante saber el comportamiento de una neurona como función de sus parámetros. Es a través de estos parámetros que podemos cambiar las características de la locomoción final. Existe una relación cuasi-lineal entre el periodo de la ráfaga y el parámetro α, mientras que la influencia del parámetro σ es prácticamente irrelevante en este caso. Por otro lado, la relación entre el tiempo de ráfaga y y el periodo total es en su mayoría dependiente del parámetro σ. Con estos dos únicos parámetros el modelo neuronal puede ser configurado en un rango amplio de configuraciones. En la Figura 4.12 se muestra el significado de periodo (a), ciclo de trabajo (b) y número de disparos por ráfaga (c).

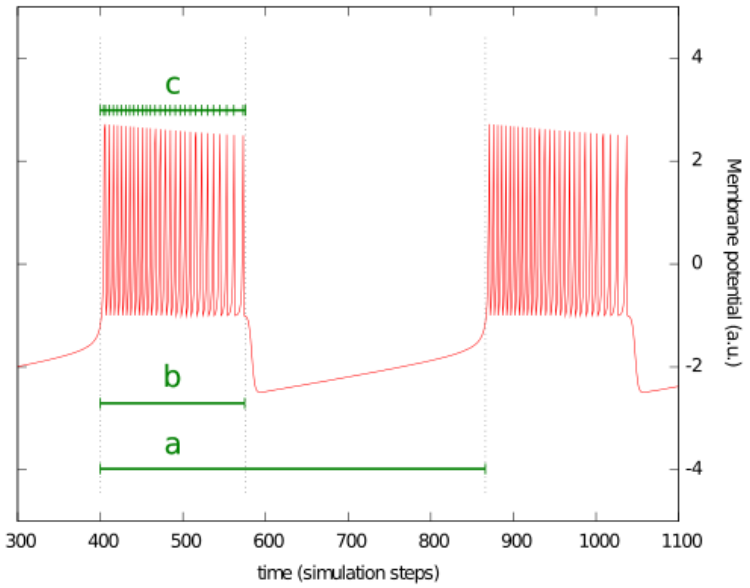


Figura 4.12: Detalles en la señal neuronal

Finalmente, la entrada externa está modelada a través de I_n. Esta variable es esencial para la organización autónoma: las unidades de procesado del CPG (las neuronas) deben ser capaces de negociar el ritmo entre ellas. Además, entre el CPG y el robot físico puede conseguirse entrainment mediante I_n añadiendo un término de error como entrada externa en la neurona [38]. El efecto de este parámetro dependerá del historial de eventos ocurridos en el pasado, el valor exacto de I_n y la fase del ciclo de ráfaga en el que se encuentre la neurona. Los parámetros σ_e y β_e controlarán cuanto afectan las corrientes externas tanto al subsistema lento como al rápido.

4.3. Conexión entre neuronas: Modelo sináptico

Una propiedad clave de los CPGs es que son autónomos, donde las diferentes unidades que lo forman hablan unas con otras para negociar la función a realizar entre todos. Ahora vamos a presentar el modelo que hemos elegido para implementar las sinapsis, el canal de comunicación entre las neuronas.

En anteriores trabajos del GNB [2, 3], se utilizó un modelo sináptico químico. Se trataba del modelo sináptico desarrollado por Destexhe [45]. En él, cuando un disparo de potencial llega desde la neurona pre-sináptica, la sinapsis libera una cierta cantidad de neurotransmisores que marcan a los receptores en las neuronas post-sinápticas. Con el paso del tiempo, las moléculas neurotransmisoras dejan de marcar a los receptores. Si una sucesión de disparos llega en un periodo corto de tiempo, la respuesta sináptica de cada uno de ellos se solapará. Por tanto, el estado de la sinapsis es dependiente de eventos pasados, como un tipo de memoria contextual. Las descripción matemática del modelo es la siguiente:

$$\dot{r} = \begin{cases} [T] \cdot \lambda(1-r) - \beta r, & t_f < t < t_f + t_r \\ -\beta r, & \text{resto} \end{cases} \tag{4.4}$$

$$I(t) = g \cdot r(t) \cdot (X_{post}(t) - E_{syn}) \tag{4.5}$$

La primera ecuación define el ratio de receptores químicos marcados en la neurona post-sináptica, y la segunda es donde se define la corriente post-sináptica creada durante cada instante t. Cuando se une estas ecuaciones al modelo de Rulkov se necesitará usar una versión discreta de este modelo sináptico.

En este Proyecto Final de Carrera se ha decidido utilizar un modelo sináptico más sencillo, tarea que se trata de uno de los objetivos principales de este proyecto (introducido junto al resto de objetivos en el Capítulo 1). Esto se debe a que la carga computacional que supone estar integrando cada uno de los pasos del modelo propuesto por Destexhe supone un problema para la ejecución de la red neuronal artificial en el microcontrolador. En anteriores proyectos, todos los cálculos se realizaban en un ordenador y además de manera offline, primero se obtenían los resultados no importando el tiempo requerido para conseguirlos y luego, estando almacenados en el PC, se iban transfiriendo al microcontrolador según se necesitaban. Ahora se necesita un modelo ligero, ya que el microcontrolador debe calcular en cada instante el siguiente paso a realizar para poder funcionar en tiempo real. Para ello se debe simplificar en aquellos aspectos en los que se pueda realizar una simplificación, y la sinapsis es uno de ellos. Podemos utilizar algún otro modelo más sencillo y con las características suficientes para que las neuronas puedan ser capaces de establecer una buena comunicación. El modelo que se va a utilizar es el propuesto por Rulkov [44] junto a su modelo neuronal. Se trata de un modelo compuesto por sinapsis eléctricas instantáneas.

Las ecuaciones definidas por Rulkov de los mapas neuronales para la simulación de varias neuronas acopladas unas con otras tienen la siguiente forma

$$x_{i,n+1} = f(x_{i,n}\ ,\ y_{i,n} + \beta_{i,n}) \tag{4.6}$$

$$y_{i,n+1} = y_{i,n} - \mu(x_{i,n} + 1) + \mu\sigma_i + \mu\sigma_{i,n} \tag{4.7}$$

dónde el índice i especifica la célula, y σ_i es el parámetro que define las dinámicas de la célula sin conectar. El acoplamiento entre las células se realiza mediante la corriente que fluye de una célula a otra, siendo este flujo instantáneo. Este acoplamiento está modelado como

$$\beta_{i,n} = g_{ji} \cdot \beta^e \cdot (x_{j,n} - x_{i,n}) \tag{4.8}$$

$$\sigma_{i,n} = g_{ji} \cdot \sigma^e \cdot (x_{j,n} - x_{i,n}) \tag{4.9}$$

dónde $i \neq j$, y g_{ji} es el parámetro que caracteriza la fuerza de los acoplamientos. Los coeficientes σ^e y β^e ajustan el balance entre los acoplamientos para los procesos de dinámica lenta y rápida, respectivamente.

Mientras el modelo de la sinapsis de Destexhe se trata de un modelo químico, el de Rulkov es eléctrico. El potencial eléctrico se propaga mucho más rápido que las sustancias químicas entre las neuronas, y por tanto, el modelo matemático que lo caracteriza se trata de un modelo mucho más sencillo. Por tanto, debido a las características que se buscan, para la realización de este Proyecto Final de Carrera se ha elegido un modelo de sinapsis eléctrico. La ventaja clara que ofrece es su velocidad de cálculo, aunque tiene una desventaja, que se trata de su menor estabilidad respecto al modelo químico. El modelo químico es más robusto, y ofrecerá una mejor estabilidad en el ritmo producido por las neuronas. Al no tener unos motores excesivamente precisos, se puede asumir esta menor estabilidad ya que no afectará en gran medida el desplazamiento del robot.

En la sinapsis, la unidad básica de sincronización será la ráfaga en sí, y no cada disparo de manera individual. Además de esto, las sinapsis deben introducir retardos para que se realice un control más preciso de la diferencia de fase entre las neuronas.

Se dice que una sinapsis es excitatoria cuando la probabilidad de que la neurona post-sináptica realice un disparo aumenta después de que la neurona pre-sináptica haya disparado. Si

la probabilidad decrece, la sinapsis es inhibitoria. Si la neurona post-sináptica emite disparos de manera periódica, una sinapsis excitatoria aumentaría su frecuencia mientras que una inhibitoria haría que esta frecuencia disminuyese. En el CPG desarrollado para este proyecto, como se verá en el Apartado 4.5.3, se utilizará únicamente la sinapsis inhibitoria que permitirá activar y desactivar a las neuronas cuando así se requiera.

4.4. Del código neuronal a las señales motoras

Las neuronas utilizadas en este trabajo son un sistema dinámico muy rico, en el sentido que contienen dos subsistemas emparejados: uno lento y otro rápido. Tomando ventaja de estas múltiples escalas temporales, se puede separar el mecanismo de sincronización (subsistema neuronal lento) del mecanismo de codificación de la locomoción (subsistema neuronal rápido). Realmente, aunque estén matemáticamente emparejados, son relativamente independientes. El subsistema lento es responsable de la activación y desactivación del subsistema rápido. El subsistema rápido, en cambio, es responsable de la generación de disparos de potencial. Estos disparos son traducidos a eventos discretos individuales de comunicación entre neuronas mediante la sinapsis. Al final, el mensaje enviado de una neurona a otra depende tanto de las dinámicas intrínsecas lentas y rápidas de la neurona como de las dinámicas de la sinapsis.

La distribución temporal de los disparos es un mecanismo biológico que permite codificar información [17, 18, 84], y nosotros lo usamos para codificar los comandos de locomoción. En el modelo de oscilador que vamos a utilizar, dos neuronas se activan alternativamente para controlar una articulación. Cuando una neurona está activa, tira de la articulación para un lado. Cuando esa neurona está inactiva, no debería ejercerse ninguna fuerza en la articulación en el sentido correspondiente. Por tanto, el patrón de activación dará forma a la trayectoria de la articulación: cada disparo individual realizará pequeños tirones en una dirección. Este modelo utilizado para las motoneuronas se ha extraido del trabajo realizado por Fernando Herrero-Carrón en su Tesis [2].

Una neurona llamada motoneurona es la responsable de decodificar esta información y traducirla a la señal que finalmente será enviada al actuador. En el caso del robot que se estudia, esta señal controla el ángulo de un servo (véase Figura 4.13).

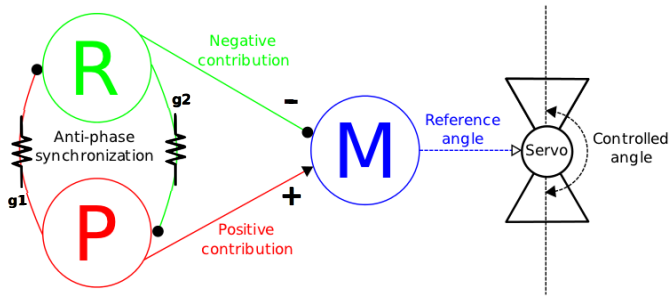


Figura 4.13: Arquitectura neuronal para la generación locomotora

Se ha demostrado [85] que los músculos de los seres vivos realmente suman los disparos de una ráfaga. Al llegar disparos individuales desde una motoneurona, los músculos sufren micro-contracciones. Si los disparos llegan de forma regular y con una frecuencia suficiente, los músculos llegan a conseguir situarse en un estado de contracción. Esto es lo que denominan la componente tónica de los comandos neuronales. Este comportamiento es muy similar al que muestra el modelo de motoneurona que vamos a introducir a continuación.

Las motoneuronas leen la actividad del oscilador a través de un par de sinapsis (véase Figura 4.13). Estas sinapsis conectan a las neuronas remotora y promotora con la motoneurona, y son gobernadas por la siguiente ecuación de umbral:

$$f_s(x,\nu) = \begin{cases} 1, & x > \nu \\ 0, & \text{resto} \end{cases} \tag{4.10}$$

El papel de esta función es el de detectar disparos de las neuronas. Poniendo este umbral a, por ejemplo, $\nu = -1{,}5$, esta función aplicada al potencial de una neurona tendrá un valor de salida igual a 1 cuando hayan disparos individuales y 0 en caso contrario.

El papel de la motoneurona (M) es integrar los eventos individuales producidos por cada una de las neuronas. Si la neurona promotora (P) emite un disparo, la motoneurona moverá el servo un poquito en un ángulo positivo. Si emite un segundo disparo lo suficientemente cercano al primero, el servo se posicionará un poco más lejos de manera positiva. Análogamente, la neurona remotora (R) hará que la motoneurona mueva el servo hacia un ángulo negativo. Si ambas neuronas se encuentran en silencio, M se encargará de mover lentamente el servo a una posición de descanso en el ángulo 0. Esto se realiza mediante la siguiente ecuación:

$$m_{n+1} = m_n + \frac{\alpha_M \cdot H}{\tau_M} \cdot (-m_n + \chi \cdot c) \tag{4.11}$$

$$c = f_s(P_x,\nu) - f_s(R_x,\nu) \tag{4.12}$$

donde, m_{n+1} es la salida de la neurona M (en grados), los dos términos f_s son la función umbral aplicada a las entradas provenientes de R y P, los términos α_M, H, y τ_M controlan la pendiente de crecimiento y decrecimiento y la forma de la señal de salida, y χ define la amplitud máxima de la señal de la motoneurona.

En esta ecuación, P contribuye positivamente y R de manera negativa. Dado el hecho que P y R oscilan en anti-fase uno respecto al otro, el resultado de m_{n+1} es una función oscilatoria delimitada entre $-\chi$ y $+\chi$ (véase Figura 4.14). Cuando la motoneurona no reciba ninguna entrada porque R y P se encuentren en estado silencioso, la salida volverá a 0 debido al termino de sustracción $(-m_n)$.

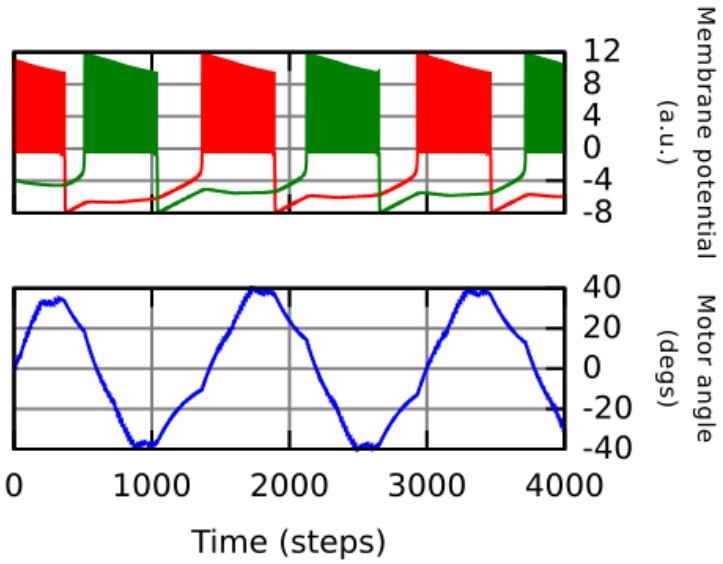


Figura 4.14: Salida Motoneurona en función de Promotora y Remotora

4.5. Diseño del CPG

Usando un modelo neuronal con unas ricas dinámicas intrínsecas y una topología simple basada en la inhibición mutua, se ha obtenido un oscilador modular autónomo. Ahora se va a mostrar el diseño que se ha pensado para implementar el CPG en la generación del movimiento para un robot modular con forma de gusano. La correspondencia de cada par de neuronas R y P, junto a su correspondiente motoneurona, con cada módulo del robot se puede apreciar en la siguiente imagen (véase Figura 4.15). En las representaciones de los distintos modelos a lo largo del capítulo se omite a las motoneuronas para tener una mayor claridad a la hora de visualizar las conexiones, por lo que aunque no aparezcan, se debe pensar que las motoneuronas están incluidas en el modelo.

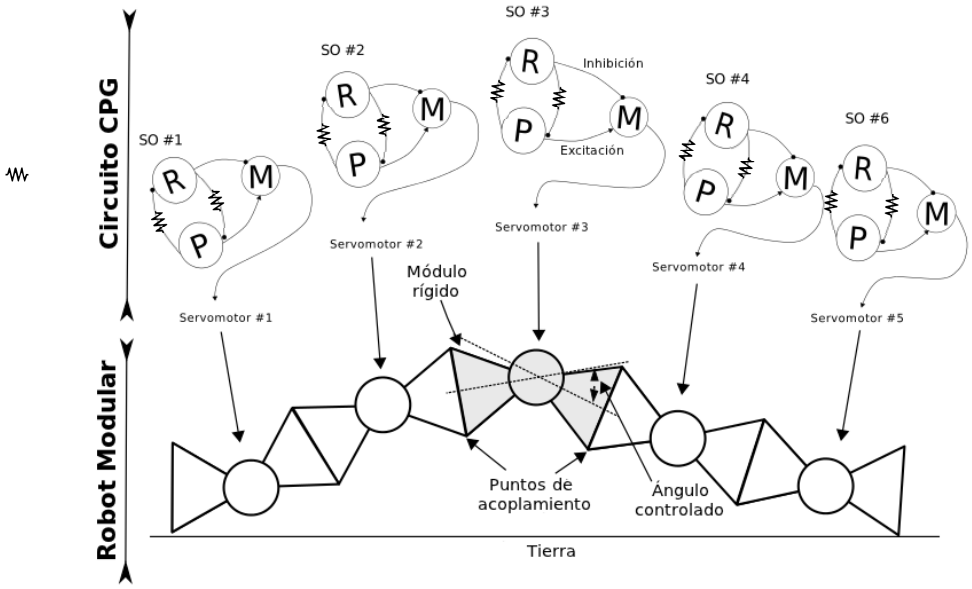


Figura 4.15: Correspondencia entre osciladores y módulos

La pregunta es que mecanismos son efectivos para construir canales de negociación por los cuales los módulos compartirían la información para llegar a un acuerdo global por el que conseguir un patrón de movimiento efectivo. Debido a que se quieren construir controladores para una cadena de módulos, se aplican dos restricciones: conectividad únicamente con los vecinos más próximos y repetitibilidad. Estas dos restricciones permiten patrones de diseño escalables a distintos tamaños de robot y un diseño real en el que partes del cuerpo lejanas no están conectadas (no se puede conectar la cabeza con la cola).

4.5.1. Patrones de conectividad

Para el diseño del CPG a utilizar en este proyecto se presentan una serie de modelos planteados por Fernando Herrero en su tesis [2]. En este apartado se van a mostrar estos modelos, realizando un análisis de sus características, para posteriormente en el siguiente apartado (Apartado 4.5.3) poder elegir el modelo más apto para el CPG que se desea diseñar. Las estructuras de CPG posibles son:

Módulos no acoplados:

Los módulos no están unidos unos con otros, siendo todos los módulos similares y teniendo los mismos parámetros. El comportamiento esperado es que todos los módulos oscilen a la misma frecuencia con una diferencia de fase constante, cuyo valor dependerá únicamente de las

condiciones iniciales. Se observa que esto no es del todo cierto, ya que mientras la fase no es del todo constante, su deriva si lo es fuera del periodo transitorio. Esto muestra que la frecuencia de oscilación de un módulo depende de la diferencia de fase entre las neuronas remotora y promotora dentro del módulo y de las condiciones iniciales particulares de cada uno.

Inhibición simétrica:

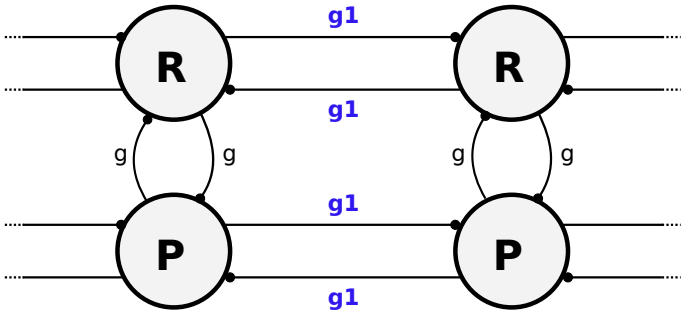


Figura 4.16: Modelo CPG Inhibición simétrica

El primer intento de Fernando fue conectar módulos adjuntos con el mismo esquema con el que se conecta a las neuronas alternas (promotora y remotora) dentro de un mismo módulo, es decir, con dos sinapsis inhibitorias, una en cada dirección (véase Figura 4.16). Se puede dividir en dos tipos:

- **Conductividad baja:** Estas sinapsis tienen una conductividad baja para así influir en la frecuencia de actividad de la neurona post-sináptica pero no inhibirla totalmente. La idea es que las sinapsis inhibitorias prevengan que neuronas adyacentes no se disparen síncronamente, dando lugar a una secuencia de activación entre ellas. El modelo consigue un estado estable pasado un tiempo, pero esto no ocurre siempre. Al tener una conectividad baja, el efecto que se tiene sobre los receptores es bajo.

- **Conductividad alta:** La actividad del receptor es completamente inhibida. El comportamiento ahora dependerá del tiempo de recuperación de una neurona entre sucesivas inhibiciones. Hay tres tipos de actividad que pueden surgir en esta configuración:

  - **Actividad regular:** Una neurona puede dominar a sus vecinos durante unos cuantos periodos y de repente perder el control.

  - **Modo de sincronización de paridad 1:** Este modo aparece cuando hay neuronas que no tienen oportunidad de recuperarse del estado de inhibición y no pueden alcanzar un nivel alto de potencial. De este modo, cuando una neurona es inhibida, permanece de esa manera para siempre.

  - **Modo de sincronización de paridad 2:** Los primeros vecinos se sincronizan en antifase y los segundos vecinos (tienen otra neurona entre medias) se sincronizan en fase.

Inhibición asimétrica

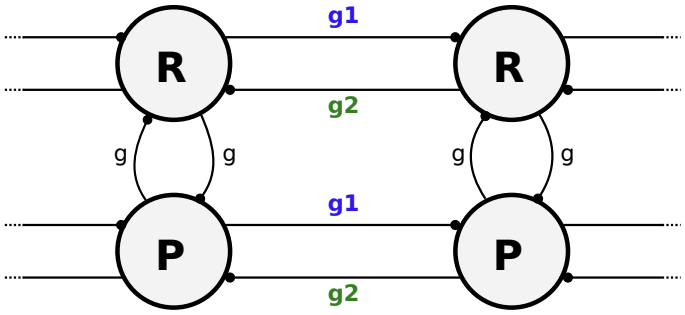


Figura 4.17: Modelo CPG Inhibición asimétrica

Usando una topología de inhibición asimétrica (véase Figura 4.17) es más sencillo prever el sentido de la locomoción e introduce una dependencia de las condiciones iniciales del controlador. De esta manera, estableciendo dos caminos diferentes con dos pesos sinápticos diferentes es una estrategia efectiva para obtener una locomoción efectiva.

Hay un periodo de negociación, al principio de la simulación, en el que los módulos ajustan lentamente su frecuencia hasta encontrar una fase estable. Una vez es encontrado un régimen estable, la diferencia de fase permanece estable hasta el final de la simulación. Claramente la inhibición asimétrica fuerza una dirección específica de desplazamiento. Esto se refleja en que la conexión ascendente dominante promueve diferencias de fase positivas, mientras que conexiones descendentes dominantes promueven diferencias de fase negativas. Esto crea claros desfases entre módulos, que resultan en ondas de propagación de las neuronas.

Es interesante saber que valores altos en las conexiones sinápticas harán que la diferencia de fase tienda hacia los extremos, definidos por π y $-\pi$ radianes. Una diferencia de fase de π radianes (que realmente es lo mismo que $-\pi$, repitiéndose la fase cada 2π radianes) entre módulos del robot le impedirá desplazarse. En ese caso el cuerpo del robot mostrará una onda estacionaria en vez de una onda viajera, requisito indispensable para una locomoción efectiva.

Bucle inhibitorio:

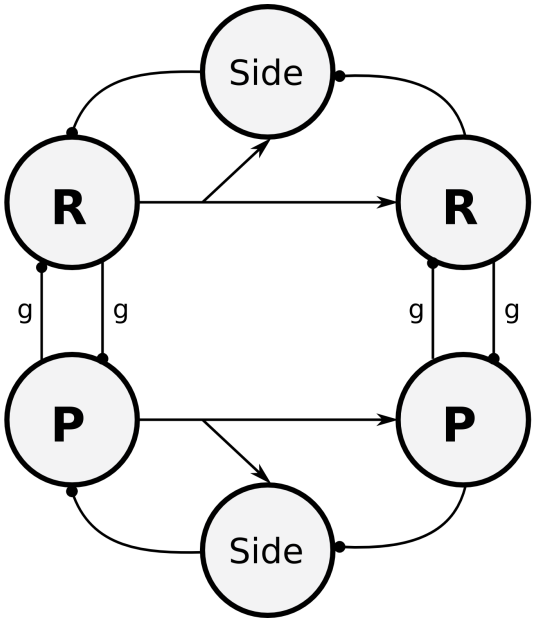


Figura 4.18: Modelo CPG Bucle inhibitorio

El objetivo de este método es conseguir un retardo constante entre módulos adyacentes mediante la introducción de un circuito específico que crea una diferencia de fase constante. Se introduce una neurona intermedia denominada "side" entre cada dos neuronas promotoras y entre cada dos neuronas remotoras (véase Figura 4.18). Esta neurona permanece en régimen silencioso excepto cuando es excitada, en cuyo caso pasa a un régimen de disparo tónico. Cuando cesa la excitación, las neuronas "side" retornan a su estado silencioso.

Este modelo es capaz de mantener un ritmo muy estable con diferencia de fase estable entre módulos. El periodo de las ráfagas es relativamente regular, teniendo menos variación los módulos de los bordes que los centrales. Aunque esto podría conseguirse con tan solo el camino excitatorio, el bucle de inhibición actúa como un detector de fase redundante que teóricamente podría ayudar al robot a recuperarse mejor de las perturbaciones.

Modelo Push-Pull:

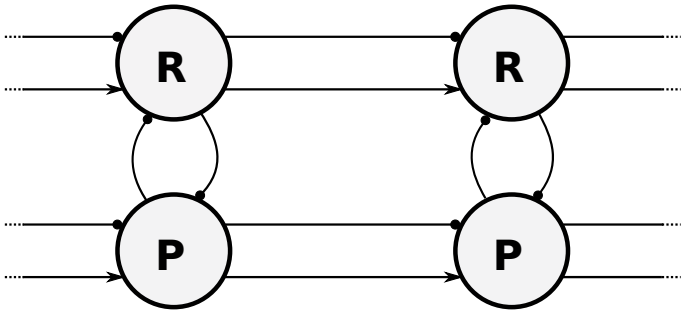


Figura 4.19: Modelo CPG Push-Pull

Es un circuito que muestra excitación e inhibición de manera simultánea, tal y como hacen muchos circuitos del sistema nervioso central (véase Figura 4.19). El resultado combinado de los dos dependerá de si el potencial de la neurona post-sináptica está cerca de la sinapsis excitatoria o de la inhibitoria.

Para los módulos que oscilen a la misma frecuencia existe un punto de equilibrio cuando oscilan completamente en fase. En este caso, se dará el caso de que sólo exista excitación. En cambio, la inhibición aparecerá siempre que la sincronización de fase no sea perfecta.

Modelo de neuronas biestables intermedias:

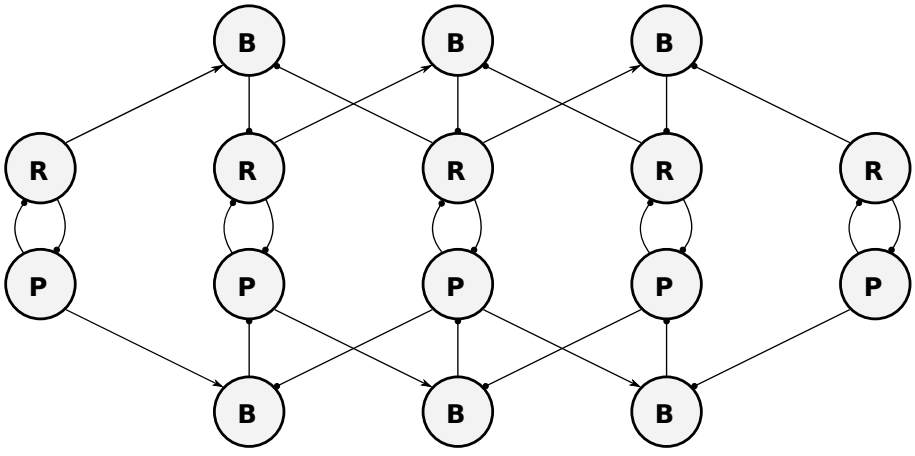


Figura 4.20: Modelo CPG de Neuronas biestables intermedias

Para diseñar un mecanismo que prevenga a una neurona de dispararse en intervalos prohibidos, se necesita un mecanismo que sea consciente del contexto, en el sentido de que sea capaz de discernir que tipo de intervalo es en el que se encuentra.

La figura (véase Figura 4.20) muestra un esquema en el que las neuronas intermedias B han sido añadidas. Estas neuronas pueden estar en dos tipos de estado: en disparo tónico o silenciosas. En que estado se encuentre sólo depende del origen de la última entrada que recibió. Estas nuevas neuronas se basan en el mapa de Rulkov [44] con ciertas modificaciones.

Para ser más claros, se coge una neurona B y se llama a las neuronas de las que recibe entradas *derecha* e *izquierda*, y se llama *medio* a la neurona que inhibe. Cuando *derecha* se encuentra en una ráfaga, B entra en modo silencioso, significando que la neurona del *medio* tiene permiso de entrar en ráfaga entre las activaciones de *derecha* e *izquierda*. Cuando *izquierda* entra en modo ráfaga, B cambia rápidamente al modo de disparo tónico, inhibiendo totalmente la actividad de la neurona del *medio*. De este modo, la neurona del *medio* solo tiene una oportunidad de emitir una ráfaga por ciclo (véase Figura 4.21).

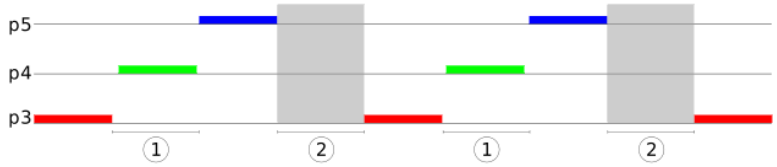


Figura 4.21: Funcionamiento de las Neuronas biestables intermedias

4.5.2. Evaluación de los modelos propuestos

Ahora se va a mostrar las ventajas e inconvenientes de los modelos anteriormente descritos, y así poder elegir el que se considere adecuado para la integración con el robot en la búsqueda de una locomoción efectiva.

La **inhibición simétrica** con conductividad alta no es una buena candidata para el objetivo que se busca. La actividad irregular no encaja en un controlador estable. El primer modo de sincronización de paridad desactiva completamente el control de la mitad de los módulos, y el segundo modo produce una onda estacionaria (en vez de una onda propagadora, como necesitaríamos en el robot). En el caso de la conductividad baja tampoco es un buen modelo, ya que para que ofrezca un estado estable depende mucho de las condiciones iniciales. La inhibición simétrica no tiene un mecanismo por el cual establecer la dirección del movimiento, con lo cual la dirección final que ofrece es imprevisible y además tiene el problema de que la dirección del movimiento se puede invertir de manera repentina debido a la deriva de fase que sufren los módulos.

Al utilizar la **inhibición asimétrica** se está ante un modelo que permite elegir la dirección del movimiento que se quiere realizar. Además ofrece, una vez encontrado el régimen estable, la continuidad de este ritmo hasta el final de la simulación. Como aspectos negativos tiene que si se utilizan valores altos en la conexión sináptica se consigue una onda estacionaria que no permite el desplazamiento del robot, y también que las neuronas necesitan un periodo de negociación largo al comienzo de la simulación en el que ajustan lentamente su frecuencia hasta alcanzar un estado estable.

En el caso del **bucle inhibitorio** se tiene a favor que permite controlar la dirección del movimiento y que ayuda al robot a recuperarse mejor de las perturbaciones. Por contra tiene que para hacer que el robot viaje en la dirección opuesta a la que se encuentre realizando el movimiento, se tendría que hacer un set de conexiones replicado en la dirección opuesta, y activar cada circuito en función de la dirección del desplazamiento.

Para el **modelo Push-Pull** se puede decir que ofrece unas diferencias de fase muy constantes

a lo largo del tiempo. En cambio tiene la desventaja de que no ofrece ningún mecanismo para ajustar la frecuencia del primer módulo a la frecuencia más baja (debido a la actuación del primero) del resto de módulos. Se debe a que el flujo de información solo va en un sentido, ocurriendo que los módulos anteriores no puedan procesar información proveniente de los módulos posteriores. Debido a esto, la necesidad de una topología no abierta, en la que todos los módulos reciban entradas de algún otro módulo, parece evidente.

El **modelo de neuronas biestables intermedias** ofrece un resultado muy bueno en cuanto a la estabilidad de la diferencia de fase entre los distintos módulos y también previene de la inversión de la secuencia. Por contra, sigue ofreciendo los mismos problemas de desajuste de fase en las neuronas de los bordes (aunque esto también pasa en el resto de modelos).

4.5.3. Topología escogida

Ante las distintas soluciones propuestas en el apartado anterior se ha escogido, basándose en los datos ofrecidos por Fernando Hererro-Carrón en su Tesis [2], el **modelo de inhibición asimétrica** para realizar el diseño del CPG. Esto se debe a que, siendo un modelo bastante sencillo, aporta estabilidad en la diferencia de fase entre los distintos módulos y también permite escoger la dirección del movimiento del robot realizando únicamente un par de cambios en el modelo. El detalle más restrictivo que ha hecho elegir este modelo es que, dentro de que varios modelos cumplen las características que se piden, este es el que lo consigue haciendo uso de la topología más sencilla, detalle muy importante para disminuir la posible carga computacional en el microcontrolador.

Aparte, este modelo permite realizar el desplazamiento del robot en ambos sentidos de manera sencilla. Únicamente se debe cambiar las conductancias entre sí (desarrollado en el Apartado 5.5.4) para poder obtener desplazamiento en el sentido contrario al que se esté realizando. Esta es una característica muy importante en este modelo para que se pueda mostrar un cambio de conducta cuando se trabaje en la integración de la función locomotora y la Nariz Electrónica (desarrollado en el Apartado 5.8.1).

Hasta que llega a un estado estable, el modelo pasa por un extenso periodo de negociación entre las neuronas. El tiempo que dura esta negociación resulta ser un problema, ya que hasta que no se ponen de acuerdo el robot no comienza a moverse y si se tiene que esperar a que esto ocurra cada vez que cambiemos de sentido se tendría el robot parado demasiado tiempo. Existe una solución ante este problema. Se deben ejecutar simulaciones en el ordenador con condiciones iniciales aleatorias e ir guardando los parámetros de las neuronas en medio de la ejecución en el caso de que la simulación haya llegado a un estado satisfactorio. Estos parámetros se almacenan en el robot, y cada vez que se encienda o cambie la dirección del movimiento se cogerá un set de estos parámetros, se asignarán a las neuronas correspondientes y así el movimiento comenzará instantáneamente al haber saltado el periodo de negociación inicial. Gracias a esta solución no debe preocupar la longitud del periodo de ajuste inicial que tiene este modelo.

Otro de los inconvenientes de este modelo es la tendencia a ajustar la diferencia de fase a su valor máximo o mínimo cuando se utiliza un valor de inhibición demasiado alto. La solución a este problema es sencilla: no utilizar altas conductancias sinápticas. Por medio de simulaciones se irá probando hasta conseguir ajustar el valor de estas conductancias y así obtener el desfase entre módulos que se desea.

Un aspecto en el que tanto este modelo, como el resto de modelos propuestos, falla es en la diferencia de fase que se obtiene en los módulos de los extremos. Al no recibir el mismo número de sinapsis que el resto de neuronas, estos actúan de manera algo diferente, mostrando un desfase algo diferente al resto de módulos. Fernando Herrero ya se encontró con este problema [2], y se dió cuenta de que una vez introducido el modelo en el robot, que los extremos presentasen un desfase algo diferente al resto de módulos no tenía mucha repercusión en la consecución

del desplazamiento por parte del robot. Aún así, existe una posible solución (propuesta por Fernando Herrero en su Tesis), que consiste en introducir conexiones sinápticas de los módulos de los extremos con sus segundos vecinos. En un principio no se hará uso de esta solución y que así el modelo sea lo más sencillo posible (teniendo menor carga computacional), pero si se observan problemas en el desplazamiento debidos a este problema, se incluirán las nuevas conexiones.

4.6. Adquisición de olores

4.6.1. Introducción a Olus2

Para la realización de este Proyecto Final de Carrera se va a utilizar la Nariz Electrónica desarrollada por David Yáñez, denominada Olus2 (véase Figura 4.22). El sensor encargado de captar la presencia de odorantes es un TGS2600 [86] con un único sensor interno.

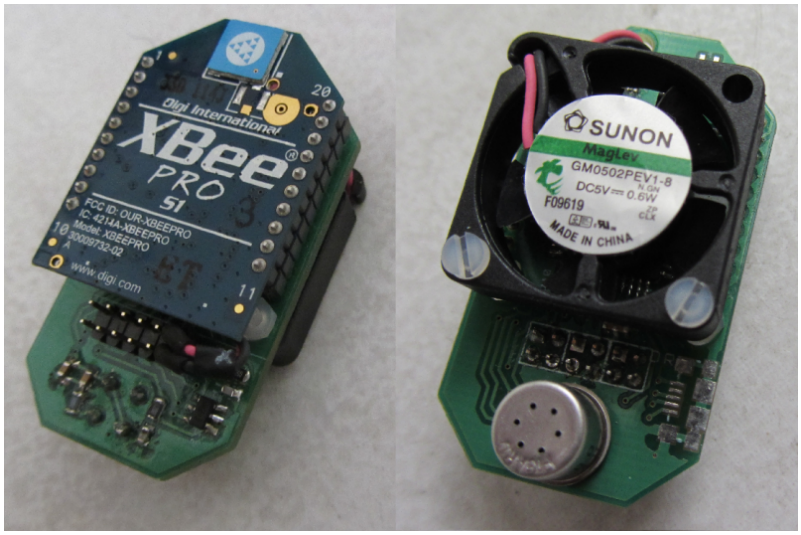


Figura 4.22: Fotografía de Olus2

En esta fotografía se puede apreciar la Nariz Electrónica y varios de sus componentes. Entre otros, se puede ver el ventilador que retira las partículas del sensor, el sensor de odorante y el módulo de comunicaciones inalámbricas XBEE (no se utiliza en este proyecto). La descripción de estos módulos se puede encontrar en el Apartado 4.6.3, en el que también se incluye la razón de por qué no se utiliza el módulo XBEE. Esta nariz es el resultado de la evolución de características de la nariz original que desarrolló David Yáñez para su Trabajo Fin de Máster [4].

La estructura básica de una nariz artificial comprende los siguientes elementos (véase Figura 4.23):

- **Sensor/es:** Para permitir la separabilidad de olores complejos es necesario contar con varios elementos de adquisición. En muchos casos éstos están encapsulados en una envolvente individual.
- **Acondicionador de señal:** En la mayoría de los casos los sensores traducen la concentración de diferentes partículas olorosas a señales eléctricas generalmente débiles y poco robustas. El acondicionamiento de señales lo conforman el conjunto de elementos

amplificadores y filtros que ajustan los niveles de señal a valores adecuados para su transmisión y lectura.

- **Sistema de preprocesamiento:** Herramienta para la compresión de información o su normalización.

- **Sistema de clasificación de las señales registradas:**

  1. **Técnicas estadísticas:** Métodos lineales de calibración, sistemas de análisis lineales de discriminación, análisis de componentes principales y análisis de clusterización.
  2. **Técnicas de inteligencia artificial:** Redes neuronales multicapa feedforward o competitivas, análisis de patrones basado en lógica difusa, sistemas neuronales difusos, etc.

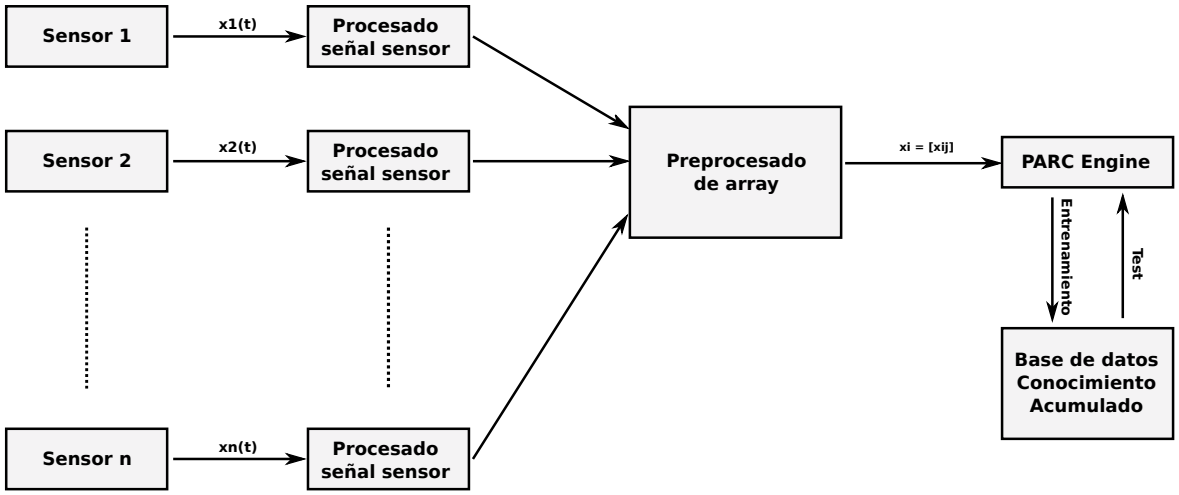


Figura 4.23: Estructura básica de una Nariz Electrónica

Esta imagen permite observar la estructura que suelen presentar las Narices Electrónicas. Es un diseño general a partir del cual se puede hacer un diseño propio como el que realizó David Yáñez.

Olus2 tiene únicamente los tres primeros elementos: Sensor para captar la presencia de odorante, acondicionador de señal para ajustar la señal obtenida y preprocesamiento con el que normalizar la señal acondicionada. El último elemento, la clasificación de señales registradas, se debe realizar fuera de la nariz electrónica. Esta clasificación se puede realizar tanto en un PC como en otro microcontrolador, que estarán conectados a la nariz electrónica mientras esta se halla captando olores. Olus2 no realiza la función de clasificación debido a que David Yáñez buscó crear una nariz portátil con unas características muy sencillas para que, de este modo, se pudiese integrar de manera barata un amplio número de ellas y, que un centro de control se encargase de gestionar y clasificar las señales obtenidas por los sensores.

4.6.2. Sistema funcional de Olus2

Si pensamos en la nariz como un conjunto de elementos diferentes que se pasan información unos a otros y donde la función de cada uno de ellos está claramente diferenciada del resto, podemos observar un esquema como el siguiente (véase Figura 4.24):

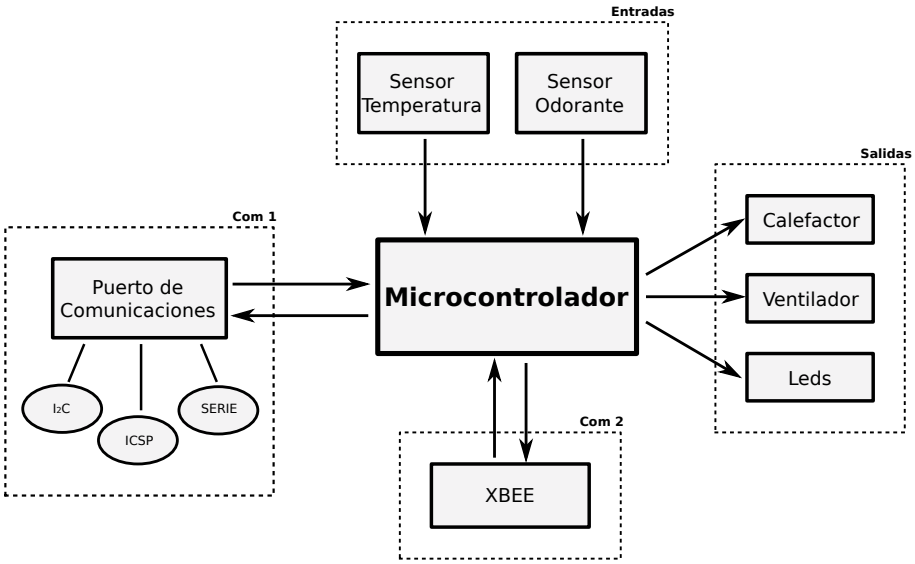


Figura 4.24: Diagrama de sistema de Olus2

En el centro del sistema tenemos al microcontrolador, que se encarga de gestionar el funcionamiento del resto de componentes. Recibe las lecturas, del medio en que se encuentra la Nariz Electrónica, que realizan los sensores, tanto el de olor como el de temperatura. Por otro lado, ajusta los valores que deben mostrar el calefactor, el ventilador y los LEDs. También dispone de un módulo de comunicaciones, accesible a través de una serie de pines, que ofrece los protocolos I2C, ICSP y SERIE. Por último, dispone de un módulo XBEE que ofrece la posibilidad de comunicarse de manera inalámbrica a otra nariz electrónica o a cualquier otro dispositivo.

4.6.3. Funcionamiento de los componentes de Olus2

Ahora pasamos a explicar más en detalle la función de cada uno de los módulos enumerados en el apartado anterior:

Microcontrolador:

Es el cerebro de la nariz electrónica. Se encarga de obtener el valor que devuelven los sensores, de controlar los valores de los elementos de salida y además gestiona las comunicaciones de la nariz con elementos del exterior. Se trata de un PIC18LF4550 [76], un microcontrolador de la casa MICROCHIP (véase Figura 4.25).

Figura 4.25: Fotografía del PIC18LF4550

Es un microcontrolador de 8 bits de montaje superficial, que consta de 32 KB de memoria de programa, 2 KB de datos RAM y una frecuencia máxima de 48 MHz. En el montaje de

OLUS2 el microcontrolador funciona a una frecuencia de 32 MHz, aunque podría subirse hasta 48 MHz si fuese necesario un funcionamiento de mayor velocidad.

Sensor de Odorante:

Sensor encargado de captar las partículas que se encuentran en el aire en la presencia del odorante que se quiere medir. Se trata de un sensor TGS2600 de la casa FIGARO (véase Figura 4.26), especializado en la detección en el aire de partículas contaminantes. Se utiliza habitualmente en aplicaciones tales como limpieza del aire, control de ventilación y en monitorización de calidad del aire.

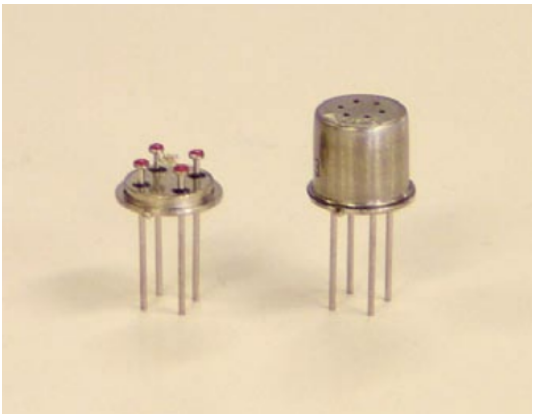


Figura 4.26: Fotografía del sensor TGS2600

El elemento sensor está compuesto por una capa de semiconductor de óxido metálico formado por sustrato de aluminio. En la presencia de un gas detectable, la conductividad del sensor aumenta dependiendo de la concentración del gas en el aire. Un sencillo circuito eléctrico (véase Figura 4.27) convierte el cambio de la conductividad en una señal de salida que corresponde a la concentración de gas.

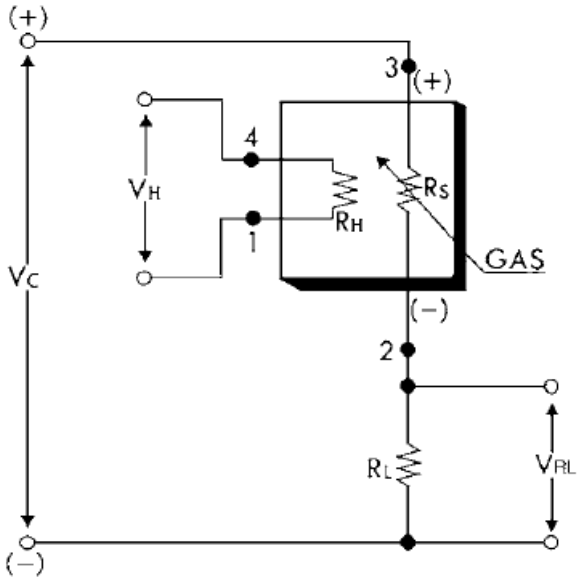


Figura 4.27: Circuito de acondicionamiento del sensor TGS2600

Atendiendo al circuito eléctrico, el sensor requiere de dos entradas de voltaje: voltaje de calefacción (VH) y voltaje de circuito (VC). El voltaje de calefacción es aplicado al calefactor integrado para así mantener al elemento sensor en una temperatura determinada, la cual es

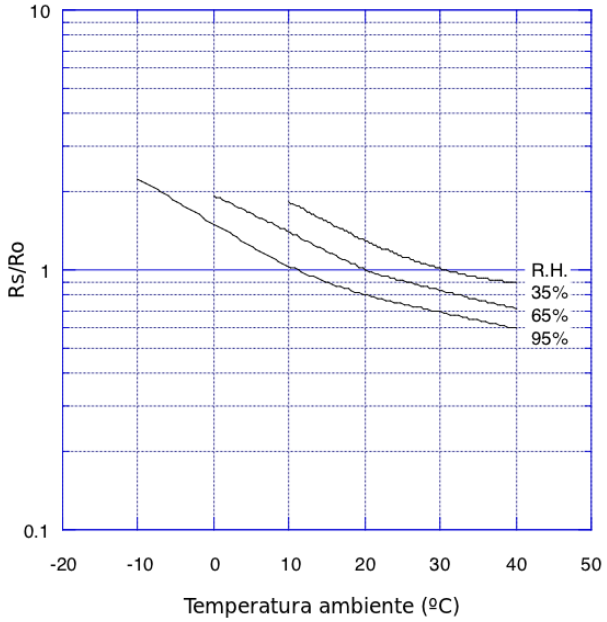


Figura 4.28: Comportamiento TGS2600 con la temperatura de calefacción

óptima para poder medir (véase Figura 4.28). El voltaje de circuito es aplicado para así poder medir el voltaje (VOUT) a través de una resistencia de carga (RL), la cual está conectada en serie con el sensor. El valor de la resistencia de carga debe ser escogido para optimizar el valor del umbral de alarma, manteniendo el consumo de energía en el semiconductor por debajo de 15 mW. El consumo de energía (PS) será máximo cuando el valor de RS sea igual al de RL durante la exposición al gas.

Sus principales características son:

- Bajo consumo de energía.
- Alta sensibilidad a contaminantes gaseosos del aire.
- Larga vida y bajo coste.
- Usa un sencillo circuito eléctrico.
- Reducidas dimensiones.

Sensor de Temperatura:

Para medir la temperatura en las proximidades del sensor utilizamos un TC1047A. Se trata de un sensor de temperatura con salida lineal de voltaje, cuyo voltaje de salida es directamente proporcional a la temperatura medida (véase Figura 4.29). Sus aplicaciones más típicas son en los siguientes sistemas: apagado por temperatura de fuentes de alimentación, ventiladores controlados por temperatura, reguladores de temperatura, equipos de baterías portables y teléfonos móviles.

Es capaz de medir el valor de la temperatura con precisión desde $-40°C$ hasta $+125°C$. Su alimentación puede variar desde 2,5 a 5,5V.

Calefactor:

El sensor de odorante TGS2600 dispone de un calefactor integrado (véase Figura 4.27) que se utilizará para poder situarse en los valores de temperatura donde se facilite la detección de las partículas de gas que se esté intentando medir. El calor que emita el elemento dependerá de la tensión (V_H) que le facilite el microcontrolador.

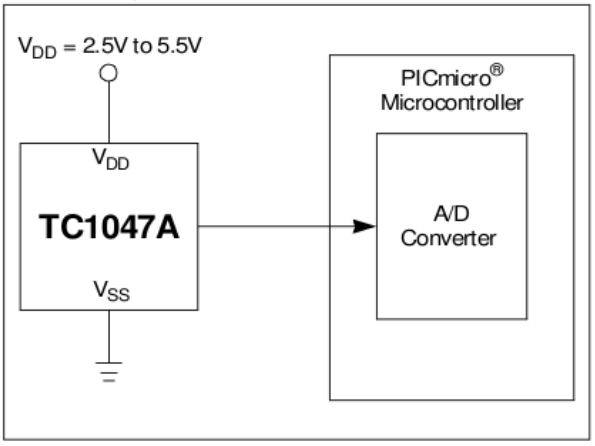


Figura 4.29: Estructura de montaje sensor de temperatura TC1047A

Ventilador:

La Nariz Electrónica dispone de un ventilador axial SUNON que hace posible el movimiento del gas en las cercanías del sensor de odorante. Esto permite tanto que el odorante llegue con mayor facilidad al sensor como que no se quede en las proximidades de el durante demasiado tiempo. Su utilidad es mayor si la nariz se encuentra dentro de la caja diseñada para ella (véase Figura 4.30), ya que el efecto del ventilador será mayor al tener que manejar un volumen de aire menor (únicamente el que se encuentre dentro de la caja). La caja está hecha de Acrilomitrilo Butadieno Estireno (ABS) y tiene unas dimensiones de 25x47x18 mm.

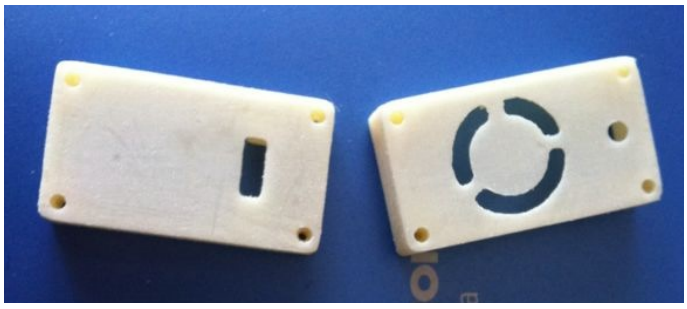


Figura 4.30: Caja de Olus2

El caudal que ofrezca el elemento dependerá de la tensión que le facilite el microcontrolador. Se maneja desde una salida analógica que puede variar desde 0 a 5V, correspondiendo los 5V al caudal máximo del ventilador.

LEDs:

Módulo de cuatro LEDs, modelo 0603 de luz roja con longitud de onda dominante de 638nm, controlados desde el microcontrolador, y cuya función es informar del estado en el que se encuentra la nariz. Su utilidad reside en facilitar al usuario de OLUS2 la labor de monitorización de la misma.

Con los cuatro LEDs se pueden realizar hasta 16 combinaciones, con las que se pueden mostrar distintos estados, e incluso indicar otros estados utilizando la velocidad de variación de un LED a otro (por ejemplo para indicar la velocidad de rotación del ventilador).

Puerto de comunicaciones:

La Nariz Electrónica Olus2 dispone de una serie de pines que se pueden utilizar para comunicar la nariz con el exterior mediante el uso de cableado. La distribución de los pines es la siguiente (véase Figura 4.31):

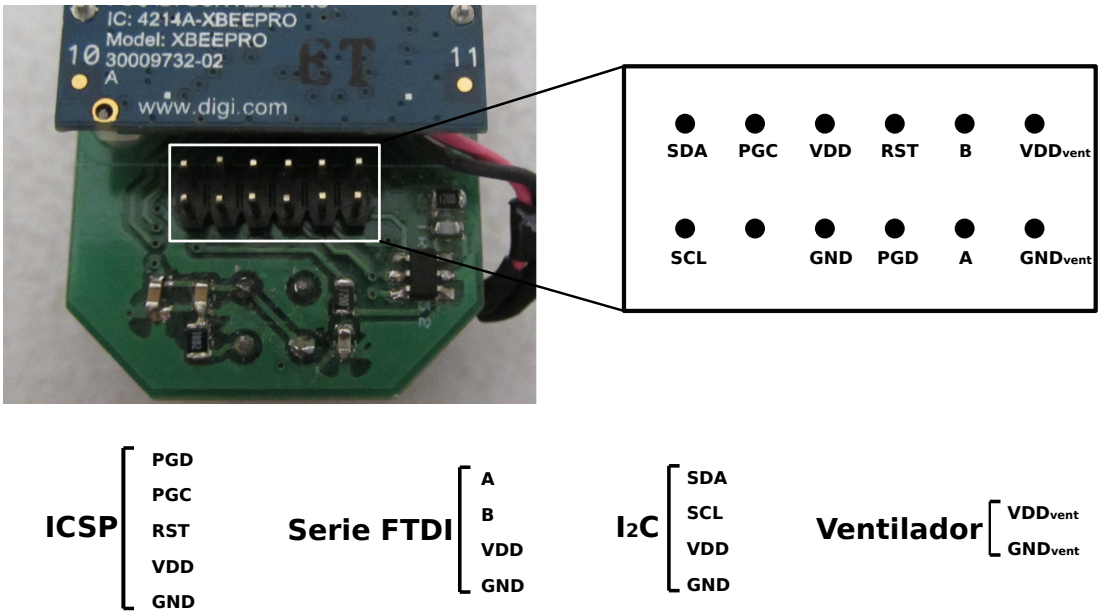


Figura 4.31: Puerto de comunicaciones de Olus2

En este puerto de comunicaciones se concentran tres tipos de protocolos, teniendo cada uno de ellos una función concreta. Son los siguientes:

- **Protocolo I2C:** Utilizado para comunicar a la Nariz Electrónica con otros microcontroladores. Esta comunicación es del tipo Maestro-Esclavo, pudiendo ejercer Olus2 los dos papeles, es decir, controlando otros dispositivos o siendo controlada externamente. Es la comunicación que se va a utilizar para comunicar el robot con Olus2. Establecer la comunicación I2C entre los dos elementos es uno de los principales objetivos en este proyecto y por tanto se debe observar en detalle las características de este protocolo de comunicación (desarrollado en el Apartado 4.7).
- **Protocolo ICSP:** Estos pines se utilizan para programar el PIC18LF4550, o lo que es lo mismo, para programar el comportamiento de la Nariz Electrónica. Mediante este protocolo se cambiará el programa que se encuentra alojado en la memoria de programa del PIC, pudiendo cambiar el tratamiento interno que hace de las señales, el valor que ofrece al ventilador y el calefactor, las tramas que envía mediante las distintas comunicaciones, etc.
- **Protocolo SERIE FTDI:** Comunicación utilizada para poder leer en un ordenador los datos captados por la nariz. David Yáñez desarrolló un programa para poder visualizar estos datos en on-line, mediante una interfaz gráfica muy amigable, y ofreciendo la posibilidad de cambiar los valores del caudal del ventilador y de la temperatura del calefactor a través de esta comunicación.

XBEE:

Este módulo se encuentra en Olus2 para poder realizar comunicaciones inalámbricas. En concreto es el XBEE PRO S1 (véase Figura 4.22), modelo que utiliza el protocolo ZigBee para

realizar las comunicaciones. Ofrece un área de comunicaciones de 10 metros con una tasa de transferencia de 250 kbps. Entre los aspectos que ofrece, los más importantes son la adecuación de su uso para tiempo real por medio de slots de tiempo garantizados, evasión de colisiones por CSMA/CA y soporte integrado a las comunicaciones seguras. También incluye funciones de control del consumo de energía como calidad del enlace y la detección de energía.

Su objetivo, por tanto, son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías. Debido a esto es una buena elección para poder situar varias narices electrónicas autónomas en una habitación y poder realizar medidas en varios puntos de la misma, y así poder hacer un seguimiento del flujo del odorante.

De los módulos que se ha hablado a lo largo de este apartado, este es el único que no se va a utilizar en este Proyecto. No se necesita hacer conexiones inalámbricas ya que se va a trabajar con una única nariz y la conexión con el robot es mediante I2C. No tiene sentido utilizar el módulo XBEE como sustituto del módulo I2C ya que, tras quitar el cable al hacer la comunicación inalámbrica, se seguiría teniendo que utilizar un cable entre la nariz y el microcontrolador que controle el robot para poder alimentar la electrónica de Olus2.

Aún así, en futuros trabajos podría ser útil, para localizar la fuente de odorante, aprovechar la señal recibida a través de este módulo de otras narices localizadas fuera del robot, situadas en puntos estratégicos de la sala en la que se realice el experimento.

4.6.4. Principales características de Olus2

Atendiendo a las características eléctricas, el consumo máximo que presenta Olus2 es de 171mA con un potencial de 5V, con lo que ofrece una potencia máxima de 0,86W. Este cálculo está realizado utilizando el módulo xBee, por lo que el consumo necesario por la nariz para funcionar en nuestro proyecto será menor. El peso que tiene es de 26,82 gramos introducida en la caja y 15,57 gramos sin ella, por lo que no se trata de un peso que vaya a impedir a el robot moverse correctamente.

Por último, se va a presentar los puntos fuertes y débiles de esta nariz electrónica.

Sus ventajas son:

- Gran portabilidad, debido a sus pequeñas dimensiones y bajo peso.
- Ofrece una gran variedad de maneras de comunicarse con otros dispositivos, siendo cada una específica para una función determinada.
- Contiene una electrónica muy sencilla.
- La repetitibilidad de experimentos a corto plazo es muy aceptable.
- El sensor responde claramente a variaciones en el flujo de aire circulante, y especialmente a temperaturas de calefacción diferentes.

Por contra, sus desventajas son:

- Si bien la repetitibilidad de los experimentos es aceptable a corto plazo, hay una molesta dependencia del histórico de densidades del olor adquirido. Se ha observado que con olores muy concentrados cuesta horas revertir la línea base de las señales a valores previos. Dada la influencia en el comportamiento de los sensores de la historia pasada, parece importante introducir atributos que caractericen el olor analizado con anterioridad como un parámetro más a tener en cuenta para la valoración del olor actual.
- En algún caso, en unos meses de uso se ha detectado un cierto desgaste o amortiguamiento de la sensibilidad de los sensores, debiendo realizar en alguno de ellos una nueva calibración del equipo para contrarrestar este efecto.

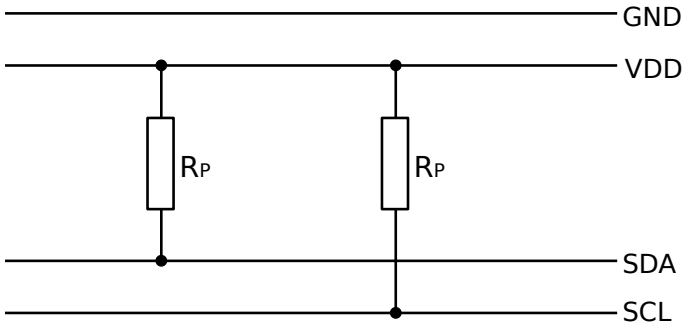


Figura 4.32: Esquema resistencias PULL-UP en la comunicación I2C

4.7. I2C

En Olus2, al contrario que el resto comunicaciones, del protocolo I2C sólo están disponibles sus salidas al exterior pero la Nariz Electrónica no dispone del código necesario para que se pueda establecer este tipo de comunicación con otros dispositivos. Debido a ello, se debe entender el protocolo I2C para poder implementar las líneas de código que faltan en Olus2.

I2C o I2C, cuyo significado en inglés es Inter-Integrated Circuit, fue diseñado por Philips en el año 1992 [82] y se trata de un bus de comunicaciones serie. Su velocidad estándar es de 100 kbps pero puede llegar a alcanzar los 3,4 Mbps. Es un bus muy usado en la industria, principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados, y para comunicar circuitos integrados entre sí que normalmente residen en un mismo circuito impreso.

La principal característica es que utiliza dos líneas para transmitir la información: una para los datos (SDA) y otra para la señal de reloj (SCL). Estas salidas son del tipo drenaje abierto, con lo que necesitan resistencias de PULL-UP para funcionar (véase Figura 4.32). El valor de esta resistencia se suele encontrar entre los 1.8 y 10k. También es necesaria una tercera línea, pero esta sólo es la referencia (masa). Como suelen comunicarse circuitos en una misma placa que comparten una misma masa, esta línea no suele ser necesaria (en este proyecto sí que se va a utilizar, ya que se conectarán elementos de placas diferentes).

Los dispositivos conectados al bus I2C tienen una dirección única para cada uno, y tienen la posibilidad de ser maestros o esclavos. El dispositivo maestro inicia la transferencia de datos y además genera la señal de reloj, pero no es necesario que el maestro sea siempre el mismo dispositivo, esta característica se la pueden ir pasando de uno a otro los distintos dispositivos conectados al bus. Esta característica de la comunicación hace que al bus I2C se le denomine bus multimaestro.

Actualmente los principales fabricantes de dispositivos semiconductores ofrecen circuitos que implementan un bus I2C para su control. El Comité I2C es el encargado, entre otras cuestiones, de asignar las direcciones I2C a cada tipo de circuito. De esta manera cada función implementada, independientemente del fabricante, posee la misma dirección; es decir, circuitos que realicen funciones equivalentes deberán poseer la misma dirección oficial I2C independientemente del fabricante.

4.7.1. Protocolo de Comunicación del bus I2C

Habiendo varios dispositivos conectados sobre el bus, es lógico que para establecer una comunicación a través de él se deba respetar un protocolo. Existen dispositivos maestros y dispositivos esclavos, siendo los maestros los únicos que pueden iniciar una comunicación.

Para realizar la comunicación el bus I2C sólo define dos señales, además de la señal de masa común:

- **SDA.** Es la línea de datos serie (Serial DAta, en inglés), semibidireccional. Eléctricamente se trata de una señal a colector abierto. Es gobernada por el emisor, sea éste un maestro o un esclavo.
- **SCL.** Es la señal de sincronía (reloj serie, o Serial CLock en inglés). Eléctricamente se trata de una señal a colector abierto. En un esclavo se trata de una entrada, mientras que en un maestro de una salida. Un maestro, además de generar la señal de sincronía, suele tener la capacidad de evaluar su estado; el motivo es poder implementar un mecanismo de adaptación de velocidades. Esta señal es gobernada única y exclusivamente por el maestro; un esclavo sólo puede retenerla para forzar al maestro a ralentizar su funcionamiento.

Evidentemente, al ser las señales a colector abierto, un enlace I2C necesitará sendas resistencias de PULL-UP en las respectivas líneas SDA y SCL. De esta manera un excitador I2C realmente sólo gobierna el estado 0 lógico en las líneas I2C, mientras que el estado 1 lógico no es suministrado por el excitador directamente sino por medio de la resistencia de PULL-UP.

El bus I2C puede encontrarse en distintos estados, que la norma define como los siguientes:

- **Libre:** La condición inicial, de bus libre, es cuando ambas señales (SDA y SCL) están en estado lógico alto. En este estado cualquier dispositivo maestro puede ocuparlo, estableciendo la condición de inicio (Start).

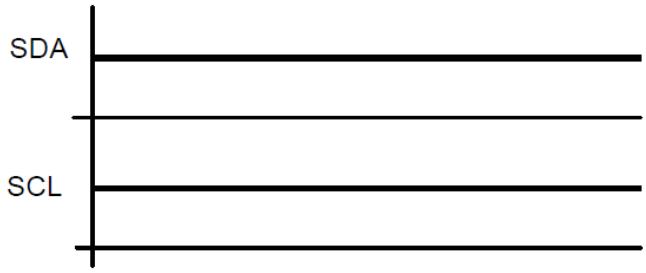


Figura 4.33: Condición de bus libre de I2C

- **Inicio:** Se produce una condición de inicio cuando un maestro inicia una transacción. En concreto, consiste en un cambio de alto a bajo en la línea SDA mientras SCL permanece en alto. Esto puede apreciarse en el cronograma mostrado en la figura 4.34. A partir de que se dé una condición de inicio se considerará que el bus está ocupado y ningún otro maestro deberá intentar generar su condición de inicio.

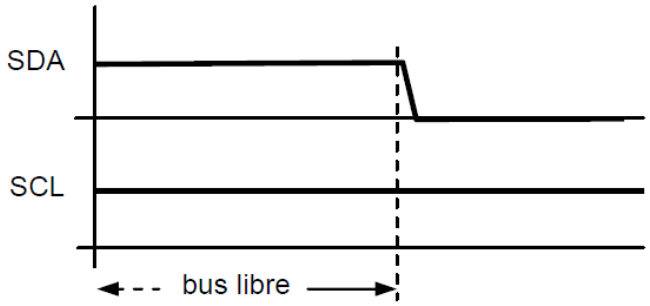


Figura 4.34: Condición de arranque de I2C

- **Cambio:** Se produce una condición de cambio cuando, estando en bajo la línea SCL, la línea SDA puede cambiar de estado. En la transferencia de datos por un bus I2C éste es el único instante en el que el sistema emisor (que podrá ser tanto un maestro como un esclavo) podrá poner en la línea SDA cada bit del carácter a transmitir.

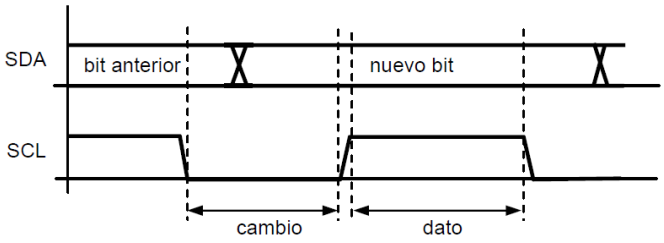


Figura 4.35: Condición de cambio de I2C

- **Dato:** Este estado es el que, una vez iniciada una transacción, queda definido por la fase alta de la señal de sincronía SCL. En este estado se considera que el dato emitido es válido, y no se admite que pueda cambiar. Recuérdese que, ya iniciada una transferencia, el único instante en que la línea de datos puede cambiar es en el estado de cambio.
- **Parada:** Se produce una condición de fin o parada cuando, estando la línea SCL en alto, se produce un cambio de bajo a alto en la línea SDA. Obsérvese que esto es una violación de la condición de dato, y es precisamente por esto por lo que se utiliza para que un maestro pueda indicar al esclavo que se finaliza la transferencia. Tras la condición de parada se entra automáticamente en el estado de bus libre.

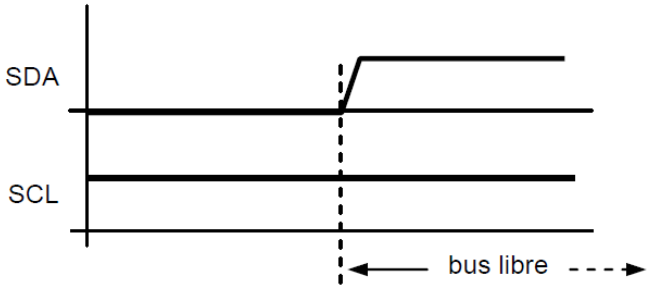


Figura 4.36: Condición de parada de I2C

Al estar inmerso en una comunicación, el primer byte que se transmite después de la condición de inicio (véase Figura 4.34) contiene siete bits que componen la dirección del dispositivo que se desea seleccionar, y un octavo bit que corresponde a la operación que se quiere realizar con él (lectura o escritura).

Si el dispositivo cuya dirección corresponde a la que se indica en los siete bits (A0-A6) está presente en el bus, éste contesta con un bit en bajo, ubicado inmediatamente luego del octavo bit que ha enviado el dispositivo maestro (véase Figura 4.37). Este bit de reconocimiento (ACK) en bajo le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse. Aquí la comunicación se establece en firme y comienza el intercambio de información entre los dispositivos.

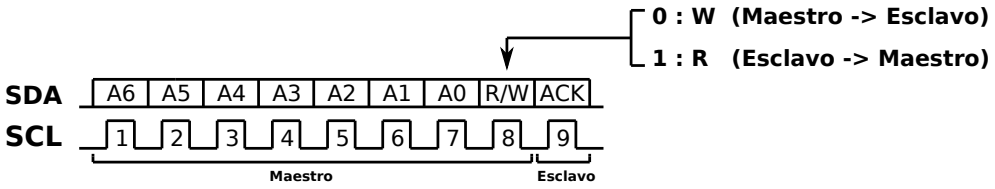


Figura 4.37: Trama para seleccionar dispositivo con el que comunicarse y establecimiento R/W

Si el bit de lectura/escritura (R/W) fue puesto en esta comunicación a nivel lógico bajo (escritura), el dispositivo maestro envía datos al dispositivo esclavo. Esto se mantiene mientras

continúe recibiendo señales de reconocimiento, y el contacto concluye cuando se han transmitido todos los datos.

En el caso contrario, cuando el bit de lectura/escritura está a nivel lógico alto (lectura), el dispositivo maestro genera pulsos de reloj para que el dispositivo esclavo pueda enviar los datos. Después de cada byte recibido, el dispositivo maestro (quien está recibiendo los datos) genera un pulso de reconocimiento.

El dispositivo maestro puede dejar libre el bus generando una condición de parada o secuencia de parada (véase Figura 4.36). Las secuencias de inicio y parada son especiales, ya que son los dos únicos casos en que se permite que la línea de datos (SDA) varíe su valor mientras la línea de reloj (SCL) se encuentra en estado lógico alto.

Si se desea seguir transmitiendo, el dispositivo maestro puede generar otra condición de inicio en lugar de una condición de parada. Esta nueva condición de inicio se denomina inicio reiterado y se puede emplear para direccionar un dispositivo esclavo diferente o para alterar el estado del bit de lectura/escritura.

Lo más común en los dispositivos para el bus I2C es que utilicen direcciones de 7 bits. Una dirección de 7 bits implica que se pueden poner hasta 128 dispositivos sobre un bus I2C, ya que un número de 7 bits puede contener desde el 0 hasta el 127 en decimal. Cuando se envían las direcciones de 7 bits, de cualquier modo la transmisión es de 8 bits. El bit extra se utiliza para informarle al dispositivo esclavo si el dispositivo maestro va a escribir o va a leer datos desde él. La dirección de 7 bits se coloca en los 7 bits más significativos del byte y el bit de lectura/escritura es el bit menos significativo.

4.7.2. Formato de transacción

Se entiende por transacción todos los eventos desarrollados entre una condición de inicio y una de parada. Una vez producido el inicio de una transacción comienza propiamente ésta, y en ella se lleva a cabo la transferencia de uno o varios caracteres, en uno u otro sentido. Una transacción I2C se caracteriza por:

- El carácter de ocho bits es la unidad de transferencia. El orden de emisión de los bits de un carácter es empezando por el más significativo, siguiendo en orden decreciente y terminando por el menos significativo.
- Un carácter puede tener diversos significados en función de quién lo emita y en qué instante lo haga. Será el código que esté tras la comunicación el que defina a que variable pertenece el valor recibido.
- Tras cada carácter se debe producir un reconocimiento por parte del receptor. El motivo es dotar al protocolo de un mecanismo de seguridad.
- El primer carácter transferido lo emite siempre el maestro; sus siete bits más significativos indican la dirección del esclavo al que se dirige, y el bit de menor peso indica el sentido de la transferencia de los subsiguientes caracteres (0=escritura, 1=lectura, siempre desde el punto de vista del maestro).
- Dentro de una transacción es posible cambiar el sentido del flujo, pero para ello hace falta generar una nueva condición de inicio (reinicio) y direccionar de nuevo el mismo esclavo.

Anteriormente se ha indicado que la señal SCL sólo la puede generar el maestro. Si el maestro actúa como emisor entonces la señal de sincronía puede entenderse como una validación de los bits que va volcando en la línea de datos SDA, sean estos bits los de un carácter de dirección o los de un carácter propiamente de datos. Si actúa como receptor entonces SCL sirve no sólo para indicar los instantes en los que el maestro toma los bits que le envía el esclavo, sino también

para indirectamente marcar la velocidad de envío del siguiente bit. El esclavo no debe poner el siguiente bit hasta que se dé una condición de cambio, y ésta no se da hasta que finaliza el estado actual de dato (esto es, hasta que SCL pasa de alto a bajo).

Por otra parte, ya se ha indicado que la unidad de transferencia es el carácter de ocho bits, y que a cada carácter le debe seguir un bit de reconocimiento por parte del receptor. Será siempre el maestro quien validará el bit de reconocimiento, sea éste ofrecido por él mismo (maestro receptor) o por el esclavo (esclavo receptor).

El desarrollo del módulo de comunicaciones I2C para la Nariz Electrónica se describe en el Apartado 5.7, donde se detalla el hardware implicado en la comunicación, las librerías existentes sobre I2C utilizadas y el código desarrollado.

5

Desarrollo e Integración

5.1. Introducción

En este capítulo se van a mostrar los pasos que se han ido realizando en la búsqueda de la consecución de los objetivos marcados para este proyecto.

Se comenzará mostrando los métodos utilizados para simular el sistema neuronal, haciéndoles variar el comportamiento mediante sus parámetros para poder observar la influencia de los mismos. Se procederá a conectar una neurona promotora con una neurona remotora (aspecto comentado anteriormente en el Apartado 4.4), observando la afectación que produce una sobre la otra y se establecerán aquellos parámetros que ofrezcan los resultados que se necesitan para cumplir los objetivos. A partir de las señales de activación y desactivación de las neuronas promotora y remotora se obtendrá la señal de la motoneurona.

Una vez se tenga modelado y caracterizado el comportamiento de cada módulo, se procederá a conectarlos para poder llegar a desarrollar el CPG completo, es decir, el sistema neuronal encargado de general la locomoción. Se irán haciendo pruebas con diferentes parámetros hasta que se encuentren los adecuados para cumplir las características necesarias con las que obtener el desplazamiento del robot. Esto se podrá comprobar introduciendo el modelo en el simulador RoboSim (simulador robótico explicado en el Apartado 5.5.4). Completado este paso se procederá a introducir el programa desarrollado en el robot y así probarlo de manera física, observando los resultados en situaciones en las que no se pueden controlar todas las variables.

Llegado a ese punto se pasarán a desarrollar las comunicaciones con la Nariz Electrónica, teniendo que reprogramarla para poder introducir el módulo I2C en sus rutinas. Establecida la comunicación entre nariz y robot se procederá a integrar un sencillo procedimiento mediante el cual el robot vea afectado su comportamiento debido a la presencia de odorante en el medio. Se realizarán pruebas para poder observar la respuesta ante estímulos olfativos, pero no se pretende realizar la búsqueda de la fuente de odorante ya que se trata de un tarea altamente compleja.

5.2. Modelo neuronal de Rulkov: Simulación de una neurona

Se comienza trabajando con una única neurona. Esto permitirá la comprensión de como afectan los diversos parámetros del modelo neuronal de Rulkov y así, posteriormente, poder ir

ampliando el modelo con los demás elementos que, de manera conjunta, acabarán formando el CPG.

5.2.1. De las ecuaciones al modelo neuronal desarrollado

Apoyándose en las ecuaciones del modelo de Rulkov [44] se ha desarrollado, en el lenguaje de programación C, un sistema que modela neuronas de una forma cercana a su comportamiento real (el código se encuentra en el Apéndice B). Un estracto del código que contiene a estas ecuaciones está representado en los siguientes recuadros que se van a presentar.

Estas ecuaciones se basan principalmente en dos variables que contiene el modelo: El potencial de membrana, x, y la variable de dinámica lenta, y, que no tiene un significado biológico directo. La forma en la que vayan variando estas dos variables afectará en gran medida al sistema, determinando el comportamiento del mismo.

```
void calculo_xy (float *x, float *y, float In)
{
  float xn, yn;

  xn = *x;
  yn = *y;

  *x = f(xn, yn+B_E*In);
  *y = yn - MU*(xn+1) + MU*SIGMA + MU*SIGMA_E*In;

  return;
}
```

Cuadro 5.1: Código que representa a las ecuaciones 4.1 y 4.2

En este recuadro (cuadro 5.1) se puede observar como se asignan los nuevos valores a las variables x e y (corresponde a las ecuaciones 4.1 y 4.2 en el Apartado 4.2). El cálculo del nuevo valor de x se realiza en la función explicada a continuación. En cambio, el cálculo del nuevo valor de y se realiza en esta misma función, dependiendo del valor en ese instante de sí mismo, de x y de la corriente sináptica de entrada a la neurona, I_n.

```
float f (float x, float y)
{
  if (x <= 0)
        return (ALFA/(1-x)) + y;

  else if (0<=x && x<(ALFA+y))
        return (ALFA + y);

  else
        return -1;
}
```

Cuadro 5.2: Código que representa a la ecuación 4.3

Esta segunda porción de código (cuadro 5.2) contiene la función que realiza el cálculo del nuevo valor que le corresponde a x (corresponde a la ecuación 4.3 en el Apartado 4.2). Esta acción se realiza utilizando el valor de x del paso anterior del mapa iterado como umbral de decisión, calculando el nuevo valor de x mediante los valores de x e y del paso anterior.

Cada neurona, dentro del código, queda representada con todos los parámetros que la definen dentro de una estructura propia que se ha desarrollado (descrita en el cuadro 5.3). Esta

estructura permite tener todos los parámetros agrupados y además define la unidad principal de nuestro modelo, que es la neurona. Entorno a las distintas neuronas se realizarán las conexiones entre sí mismas o a las motoneuronas, estableciendo con ellas el diseño final del CPG.

```
typedef struct{
  float x, y, In;
  int entrada[NUM-1], num_entradas;
}NEURON;
```

Cuadro 5.3: Estructura de la neurona utilizada en el código del modelo neuronal

La neurona tendrá sus variables definidas en la estructura, que son el potencial de membrana (x), la variable de dinámica lenta (y), la corriente sináptica entrante (I_n), y además tendrá el número de entradas por las que recibe la corriente sináptica de otras neuronas ($num\_entradas$) y la neurona de la que procede cada entrada ($entrada[]$).

5.2.2. Comportamiento del modelo neuronal

Para poder observar como afectan ciertos parámetros en este modelo, se ha procedido a realizar diversas simulaciones. A continuación se muestran los resultados de algunas de estas simulaciones en gráficas que representan tanto el potencial de la membrana como la variable de dinámica lenta.

Se empieza variando únicamente los parámetros α (ALFA en el código) y σ (SIGMA en el código), dejando el parámetro μ (MU en el código) con un valor fijo (μ=0.001). Los resultados que se obtienen son los siguientes:

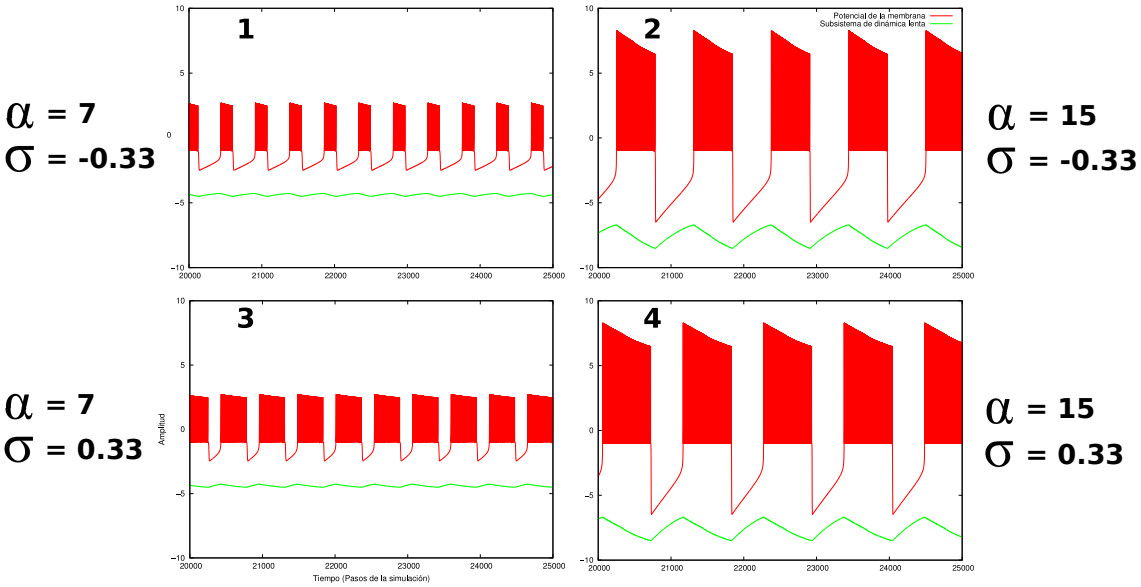


Figura 5.1: Comparativa neuronal en función de alfa y sigma. Muestra el potencial de membrana (rojo) y el susbsistema de dinámica rápida (verde)

Comparando las cuatro gráficas que se acaban de representar se puede obtener alguna conclusión. Haciendo variar el parámetro α se puede modificar tanto la amplitud como la frecuencia de las señales: A mayor α se tiene una mayor amplitud del valor de las señales y la frecuencia se hace menor. Si se modifica el parámetro σ se puede hacer variar la frecuencia y anchura de las señales: A mayor valor de σ se tiene una anchura mayor del periodo de activación del potencial de la membrana, lo que hace que disminuya la frecuencia de la señal resultante.

En cambio, si se dejan los parámetros α, σ y μ fijos ($\alpha = 15$, $\sigma = -0{,}33$ y $\mu = 0{,}001$), y únicamente se hacen variar los parámetros referentes a los estímulos externos de la neurona (I_n, β^e y σ^e), se pueden observar los siguientes resultados:

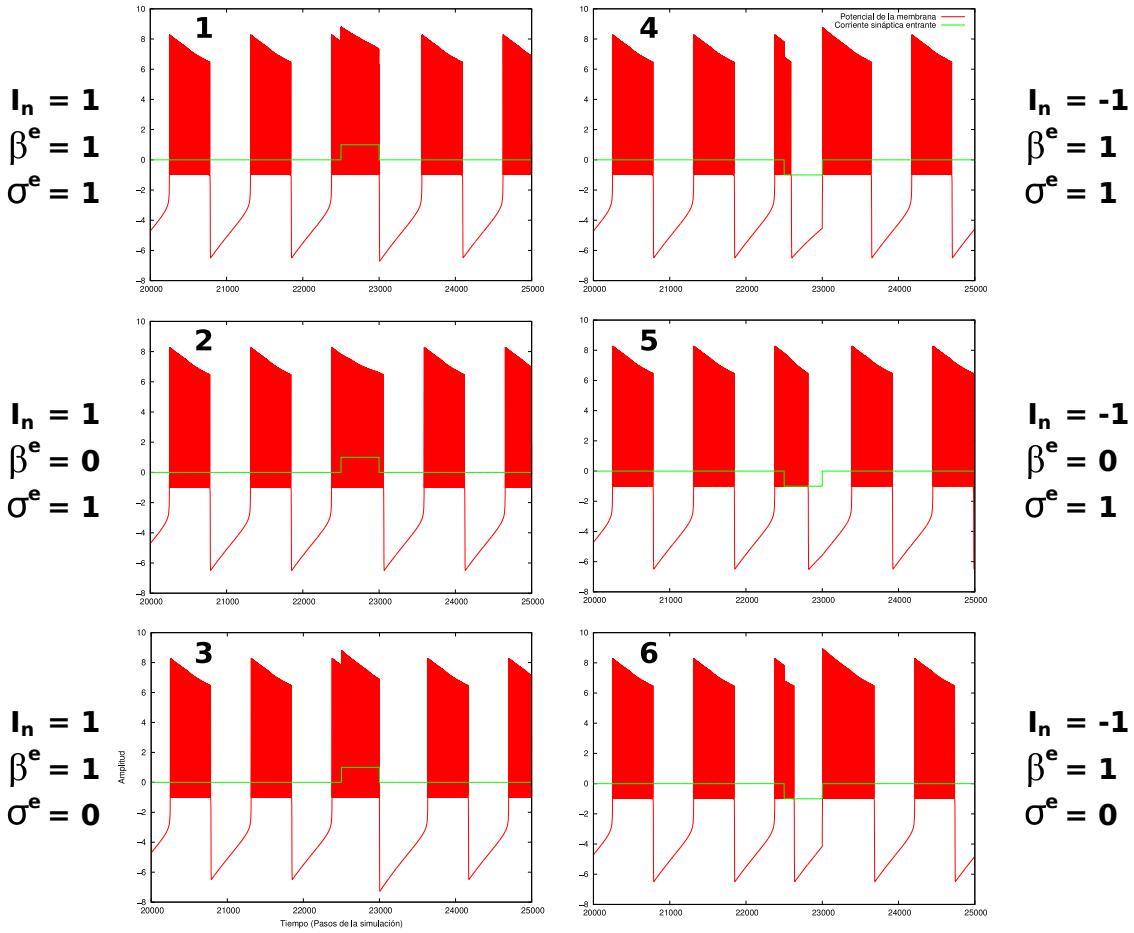


Figura 5.2: Comparativa neuronal en función de la corriente sináptica, beta y sigma. Muestra el potencial de membrana (rojo) y el susbsistema de dinámica rápida (verde)

Tras observar los resultados representados en las gráficas se pueden obtener varias conclusiones. Mediante el uso de I_n se puede activar o desactivar a la neurona: Si I_n pasa a valer $+1$ se procederá a la activación de la neurona, y en el caso de valer -1 la neurona pasará a un estado silencioso. Si se activa el parámetro β^e, en el caso de que se tenga un estímulo externo positivo, hará que la amplitud del potencial de membrana de la neurona aumente hasta su valor máximo, es decir, el que muestra en el comienzo de su periodo de activación. En cambio, activando σ^e se conseguirá que la longitud del periodo de activación de la neurona, para el caso de $I_n = +1$, aumente respecto a su longitud habitual, ocurriendo lo contrario para $I_n = -1$.

Tras observar el comportamiento de la neurona ante la variación de sus parámetros, se ha elegido una combinación de los mismos para así tener el modelo de la neurona debidamente caracterizado y por tanto poderse centrar en el estudio de otras propiedades del modelo como son la interacción con otras neuronas, la generación de la señal de la motoneurona, el funcionamiento de diversos CPGs, etc. Los parámetros fijos de ahora en adelante son:

- $\mu = 0.001$
- $\alpha = 15$
- $\sigma = 0.33$
- $\beta^e = 1$
- $\sigma^e = 1$

5.3. Sinapsis de Rulkov: Conexión de las neuronas

Una vez entendido el funcionamiento de una neurona de manera independiente dentro del modelo propuesto, se procede a aumentar el número de neuronas presente en la simulación. Gracias a esto se puede observar el comportamiento que ofrece cada una de ellas al interactuar con el resto y como varía este comportamiento al cambiar los parámetros que definen las conexiones entre unas neuronas y otras. Mediante la interacción entre neuronas se podrá generar un ritmo locomotor de manera similar a como se hace en los seres vivos mediante CPGs.

Como se comentó en el Apartado 4.3, se va a implementar un modelo sináptico diferente al utilizado en el trabajo realizado por Fernando Herrero [2]. Este modelo sináptico se trata de un modelo de sinapsis eléctricas, modelo con una menor carga computacional que los utilizados anteriormente, y por tanto más adecuado para integrar dentro de un microcontrolador (aspecto introducido en el Apartado 3.10).

5.3.1. De las ecuaciones al modelo sináptico desarrollado

La rutina que define la sinapsis del modelo se basa en las ecuaciones planteadas por Rulkov (Ecuaciones 4.6, 4.7, 4.8 y 4.9), y está compuesta por las siguientes líneas de código (cuadro 5.4):

```
for (i; i<TAM; i++){
  for (j=0; j<NUM; j++)
        x_untouched[j] = n[j].x;

  for (j=0; j<NUM; j++){
        n[j].In = dep_pasado*n[j].In;
        for (k=0; k<n[j].num_entradas; k++)
          n[j].In = n[j].In + g*(-n[j].x + x_untouched[n[j].entrada[k]]);
        calculo_xy(&n[j].x, &n[j].y, n[j].In);
  }
}
```

Cuadro 5.4: Código que define la sinapsis eléctrica entre neuronas

En este cuadro se puede observar como se calcula el siguiente paso de la simulación para cada una de las neuronas. Este nuevo paso de cada neurona depende del estado en el que se encuentren las neuronas con las que estén conectadas. Para ello primero se depende del estado de la corriente sináptica, I_n, en el paso anterior. Esta dependencia será mayor o menor en función del peso que se le dé a la influencia del valor pasado sobre el futuro, es decir, al parámetro *dep_pasado*. El nuevo valor de I_n quedará definido, aparte del valor debido al paso anterior, por la diferencia existente entre el potencial de membrana de la neurona respecto al de sus neuronas vecinas. Esta diferencia de potencial se verá potenciada en mayor o menor medida en función de la conductividad, g, que se le dé a cada una de las sinapsis.

Ahora se procede a realizar pruebas utilizando un par de neuronas y variando entre simulaciones los parámetros que acaban de aparecer en el modelo, es decir, g y *dep_pasado*. Gracias a estas pruebas será posible de discernir que parámetros son adecuados para obtener la comunicación necesaria con la que mantener el ritmo que se necesita entre neuronas para que se desarrolle locomoción.

5.3.2. Comportamiento del modelo sináptico

Primero se procede a elegir el parámetro *dep_pasado*, que determina la dependencia del valor anterior de I_n a la hora de calcular el nuevo valor de la corriente sináptica. Este parámetro sirve

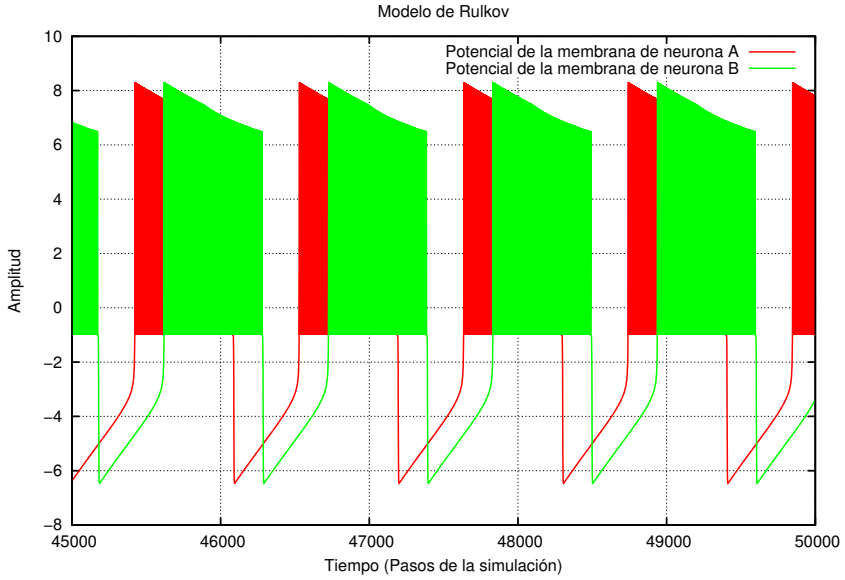


Figura 5.3: Dos neuronas conectadas con *g=0*

para que los valores de la corriente sináptica muestren menos cambios bruscos al hacer depender el nuevo valor del anterior, es decir, aporta robustez . Si *dep_pasado* es igual a 0 se producirán grandes discontinuidades en los valores ofrecidos por I_n. Si *dep_pasado* es muy grande, en cambio, habrá demasiada dependencia del valor anterior e influirá menos el cálculo proveniente de la diferencia de los potenciales actuales de las neuronas. No se tiene una medida clara para objetivar la elección del parámetro, pero al ir probando distintos valores del parámetro e ir observando gráficas se ha elegido finalmente *dep_pasado* = 0,18 como valor a utilizar para ofrecer una mayor robustez a las comunicaciones sinápticas. Se ha escogido porque las gráficas de los resultados de las simulaciones con este valor mostraban mayor suavidad en las transiciones del estado activo al estado silencioso, buena estabilidad en el estado silencioso y no quitaba excesiva relevancia al valor del potencial de las membranas en el cálculo del nuevo valor de I_n.

Una vez elegida la dependencia en la sinapsis del modelo de los valores pasados, se va a establecer la conductancia que se necesita entre las neuronas. Para establecer un valor primero se necesita un objetivo. El objetivo que se busca ahora mismo es encontrar un ritmo estable entre dos neuronas, siendo además π el valor que se busca en la diferencia de fase entre las neuronas en la activación del potencial. ¿Por qué esta diferencia de fase? Es la que se busca debido a que el siguiente paso va a consistir en conectar una motoneurona a este par de neuronas, y, para controlar a la motoneurona se necesita que las neuronas que la manejan se activen alternativamente (como se ve en el Apartado 4.4). Una diferencia de fase de π significa que cuando una neurona se encuentre en su estado de activación la otra se encontrará en estado silencioso.

Para establecer el valor de la conductancia se utilizan los valores que ofrece Rulkov en su trabajo [44]. Estos valores ya fueron presentados en el Apartado 4.2. Se prueba conectando a las neuronas como se puede ver en la Figura 4.13, pero sin realizar de momento las conexiones a la motoneurona.

Primero se prueba con un valor de conductancia $g = 0$ (valor extraído del trabajo de Rulkov [44]). Utilizar esta conductancia es similar a tener las neuronas sin conexión entre ellas (se vió en el Apartado 4.5.1). Debido a ello, las neuronas son totalmente independientes y pueden mostrar cualquier estado como estar sincronizadas en fase, en antifase o con una fase diferente. El estado que ofrezcan se deberá únicamente al punto en el que cada una de la neuronas haya comenzado la simulación.

Si se prueba con una conductancia $g = 0{,}043$ (valor extraído del trabajo de Rulkov [44]) se obtendrá una situación en la que las dos neuronas actúan en fase una con la otra. Esto quiere

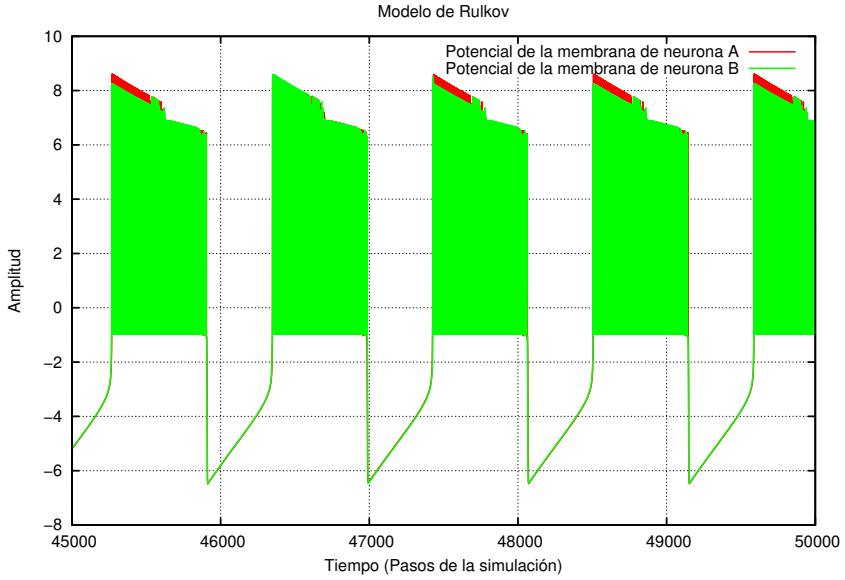


Figura 5.4: Dos neuronas conectadas con *g=0.043*

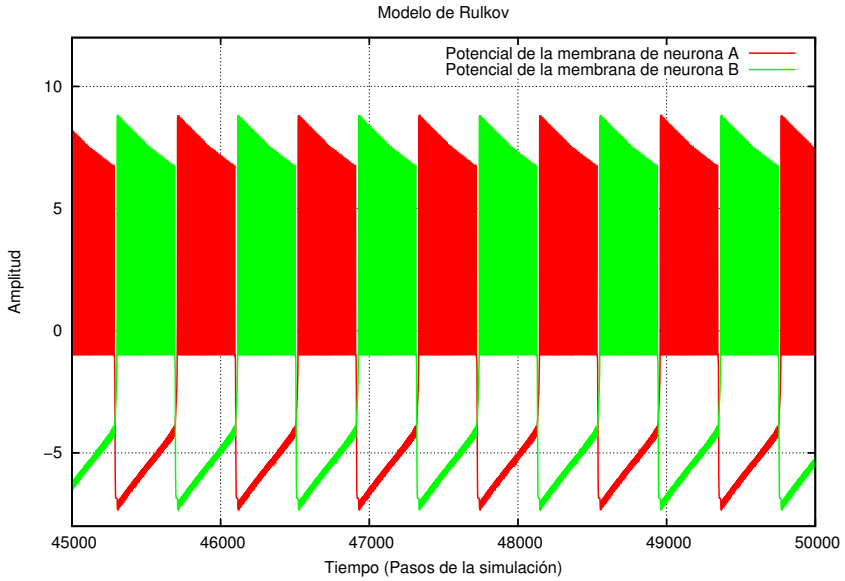


Figura 5.5: Dos neuronas conectadas con *g=-0.029*

decir que en el momento en el que una se encuentre disparando su potencial de membrana la otra hará lo mismo. Por tanto, al conectarlas con esta conductancia, las neuronas se pondrán de acuerdo para ir activando y desactivando su potencial a la par.

En cambio, si se utiliza la conductancia $g = -0{,}029$ (valor extraído del trabajo de Rulkov [44]), se observará como el par de neuronas comienza a trabajar alternadamente, es decir, en antifase. Cuando una neurona esté activada, la otra se encontrará desactivada, y ocurrirá al contrario cuando la primera neurona pase al estado de silencio. Este es el comportamiento que se desea que muestren entre sí el par de neuronas encargadas de controlar a la motoneurona, y por tanto será el valor para la conductancia que se utilizará a la hora de conectar entre sí a las neuronas promotora y remotora (como se puede ver en la figura 4.14).

Por tanto, se ha definido el valor que se utilizará para la sinapsis entre las neuronas remotora y promotora, $g = -0{,}029$. Es decir, se ha definido el funcionamiento de un módulo formado por una neurona remotora y otra promotora. En cambio, este valor no se utilizará para definir la sinapsis entre las parejas de módulos (por ejemplo las conductancias g1 y g2 en la Figura 4.17). La conexión entre módulos, que depende del valor de las sinapsis entre módulos, se definirá en

el Apartado 5.5.

5.4. Modelo de la motoneurona: Generación del movimiento

Una vez establecida la sinapsis entre el par de neuronas se va a ir un paso más allá y utilizar el ritmo establecido entre las dos neuronas para controlar a una motoneurona. Esta motoneurona será la que más adelante se encargará de accionar un servo y posibilitará, por tanto, el movimiento del robot. El conjunto de las dos neuronas, remotora y promotora, junto a la motoneurona y las conexiones que unen unas a las otras conforman un módulo (se puede ver la conjunción de los elementos en la Figura 4.13). Al finalizar el CPG se debe tener 8 módulos, con sus debidas conexiones, donde cada uno de ellos se encargará de controlar uno de los 8 servo-motores que tiene el robot.

5.4.1. De las ecuaciones al modelo de motoneurona desarrollado

Como ya se explicó en el Apartado 4.4, las motoneuronas leen la actividad del oscilador a través de un par de sinapsis (véase Figura 4.13). Estas sinapsis conectan a las neuronas remotora y promotora con la motoneurona.

El papel de la motoneurona es integrar los eventos individuales producidos por cada una de las neuronas. Si la neurona promotora emite un disparo, la motoneurona moverá el servo un poquito en un ángulo positivo. Si emite un segundo disparo lo suficientemente cercano al primero, el servo se posicionará un poco más lejos de manera positiva. Análogamente, la neurona remotora hará que la motoneurona mueva el servo hacia un ángulo negativo. Si ambas neuronas se encuentran en silencio, la motoneurona se encargará de mover lentamente el servo a una posición de descanso en el ángulo 0.

Las ecuaciones que definen a la motoneurona que se ha incluido en el sistema neuronal están representadas por las líneas de código que se muestran a continuación. Están divididas en dos funciones, que son las siguientes:

```
int f_s (float x)
{
  if (x > UMBRAL_M)
        return 1;
  else
        return 0;
}
```

Cuadro 5.5: Código que representa a la ecuación 4.10

Esta primera función (cuadro 5.5), $f_s(x)$, decide si la neurona de la que se le pasa el potencial de membrana se encuentra en estado de activación o en estado silencioso (representa a la ecuación 4.10). La función devuelve un 1 en el caso en el que el potencial de membrana supere el valor del umbral, ν (en el código representado por UMBRAL_M), y un 0 en caso contrario. Por tanto, variando este umbral se consigue cambiar los segmentos de decisión de si la neurona está activada o no, lo que repercutirá en el funcionamiento de la motoneurona como se verá a continuación.

```
float f_m(float Rx, float Px, float m)
{
  float m_nuevo, c;

  c = (f_s(Px) - f_s(Rx));
  m_nuevo = m + (ALFA_M*MU/TAO_M)*(-m + CHI*c);
  return m_nuevo;
}
```

Cuadro 5.6: Código que representa a las ecuaciones 4.11 y 4.12

En esta segunda función (cuadro 5.6), $f_m(Rx, Px, m)$, se tiene el cálculo del siguiente valor que mostrará la motoneurona (descrito por las ecuaciones 4.11 y 4.12). Este cálculo depende tanto del valor en el instante anterior de la motoneurona como del resultado de una ecuación que utiliza a la función $f_s(x)$. Esta ecuación varía en función del estado de activación de las neuronas promotora, Px, y remotora, Rx. Podrá ofrecer 3 tipos de valores: $c = 1$ cuando Px está activa y Rx no, $c = -1$ cuando Px está silenciosa y Rx activa, y $c = 0$ cuando las ambas están al mismo tiempo activas o silenciosas.

El resultado de la introducción de estas ecuaciones en el modelo se puede observar en la siguiente gráfica (véase Figura 5.6). En ella, en la gráfica superior, se aprecia el comportamiento de los potenciales de membrana de las neuronas. Se puede observar como se van activando y desactivando de manera alternada, comportamiento conseguido en la sección anterior. En la gráfica inferior de la imagen se puede apreciar el comportamiento de la motoneurona. En este caso, atendiendo a la secuencia de activación de las neuronas y a las ecuaciones plasmadas en los párrafos anteriores, se puede observar que la neurona que ofrece el potencial de membrana representado en verde se trata de la promotora y el representado en rojo se trata del perteneciente a la remotora. La neurona promotora se encargará de que a lo largo de su periodo de activación la motoneurona ofrezca valores de salida positivos, siendo negativos durante la activación de la remotora (explicado en detalle en el Apartado 4.4).

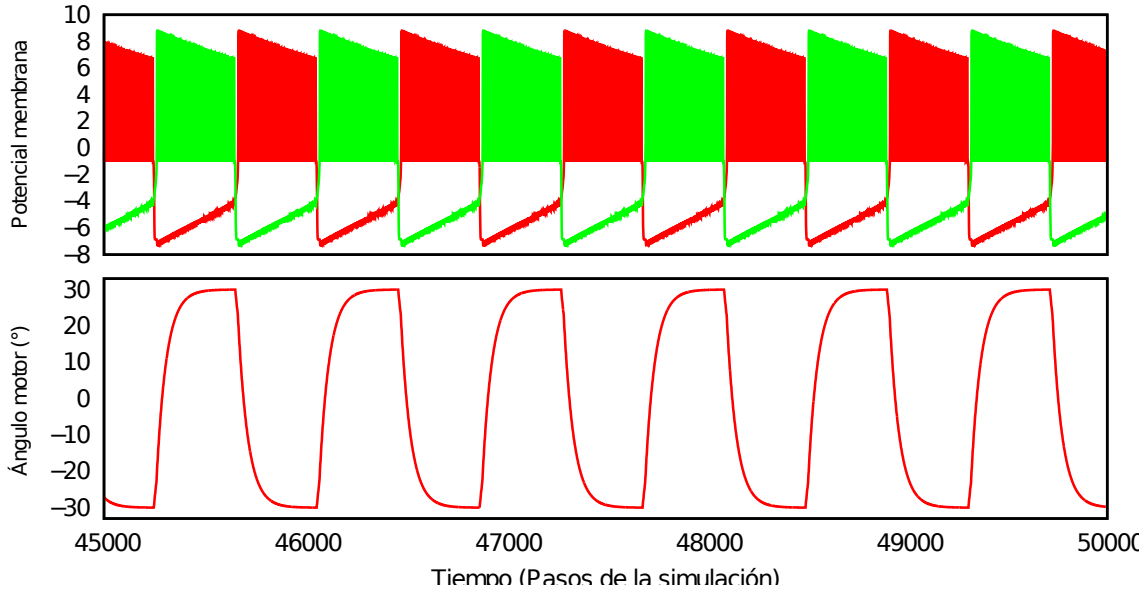


Figura 5.6: Señal de salida de la motoneurona en función del valor de las neuronas Promotora y Remotora

5.4.2. Comportamiento del modelo de la motoneurona

Ahora se va a proceder a variar ciertos parámetros para poder apreciar en que afectan al modelo de la motoneurona.

En primer lugar se va a modificar el parámetro χ (CHI en el código contenido en el Cuadro 5.6). Se cambia su valor a $\chi = 20$ (véase Figura 5.7), pudiendo apreciar cómo al variar su valor se puede modificar la amplitud de la salida de la motoneurona. Su valor representa el valor máximo (y en su vertiente negativa, al ser una señal que oscila, el mínimo) que puede obtener la motoneurona. Si se aumenta su valor, se aumentará el límite al que llegará la señal de salida, traduciéndose esto en hacer mayor el ángulo que puede alcanzar el motor respecto a su punto central durante la simulación.

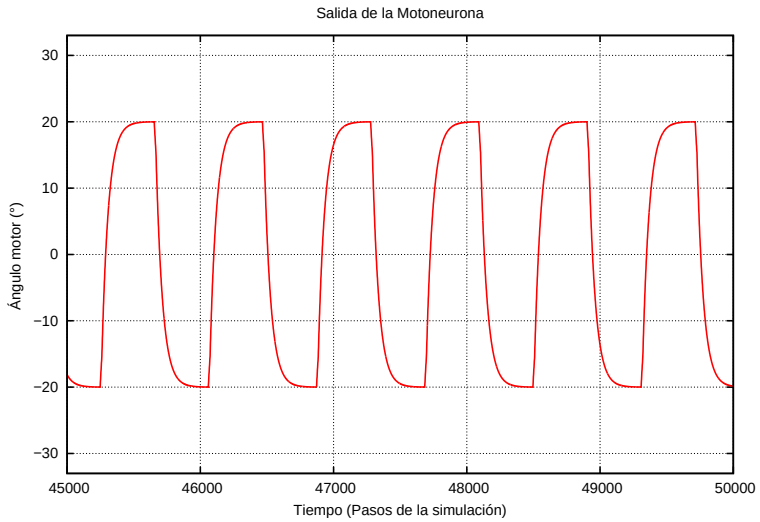


Figura 5.7: Salida de la motoneurona para $\chi = 20$

En segundo lugar se procede a variar el parámetro α_M (ALFA_M en el código contenido en el Cuadro 5.6). Se cambia el valor del parámetro a $\alpha_M = 15$ (véase Figura 5.8), y observando los resultados se puede concluir que gracias a esta variable se puede modificar la pendiente de crecimiento/decrecimiento de la señal. A valores altos de α_M se puede hacer que la señal crezca de manera casi instantánea, reduciéndose esta velocidad al poner un valor más pequeño en este parámetro. Como se verá más adelante (al introducir el CPG en el robot en la sección 5.5), no interesa un valor de crecimiento muy alto ya que la velocidad de cambio de posición de los servos es limitada, y al variar la señal demasiado rápido el servo no será capaz de seguirla. En cambio, si se hace que la pendiente de crecimiento sea muy baja, la señal de la motoneurona no será capaz de llegar al valor marcado por χ ya que no dispondrá del tiempo suficiente para ello durante los periodos de activación de las neuronas. τ_M (TAO_M en el código contenido en el Cuadro 5.6) también marca el valor de la pendiente de crecimiento, pero de manera inversa al que lo hace α_M, siendo la velocidad de crecimiento mayor cuanto menor es el valor de τ_M. Mediante el uso de los dos parámetros se puede ajustar esta velocidad de crecimiento al valor deseado. Se ha elegido $\alpha_M = 30$ y $\tau_M = 1{,}5$ ya que, con estos parámetros, la señal de la motoneurona muestra unos valores de crecimiento aceptables, semejantes a los usados por Fernando Herrero y Damián Zamorano en sus trabajos realizados con el mismo robot [2, 3].

Por último se procede a cambiar el valor de ν (UMBRAL_M en el código contenido en el Cuadro 5.5). Se pone su valor a $\nu = -0{,}5$ y se observan los resultados (véase Figura 5.9). Como ya se dijo anteriormente, este parámetro fija el valor mínimo con el que una de las dos neuronas, la remotora o la promotora, moverá el motor hacia su lado. Si el potencial de membrana de la promotora supera el valor de ν, el motor comenzará a moverse en sentido positivo, alcanzando (si dispone de tiempo suficiente para ello) el valor máximo en positivo. La neurona remotora realizará el mismo cometido, salvo que moverá el servomotor hacia ángulos negativos. Al variar este parámetro se deteriora la señal de salida de la motoneurona. En el caso de aumentar el valor del parámetro se degradará la señal cuando se sitúe en sus valores máximos, ya que al situarse la neurona en su ráfaga de disparos tónicos, su potencial varía rápidamente, bajando en ciertos instantes por debajo del umbral, lo cual hará el valor del ángulo del servo tienda, por

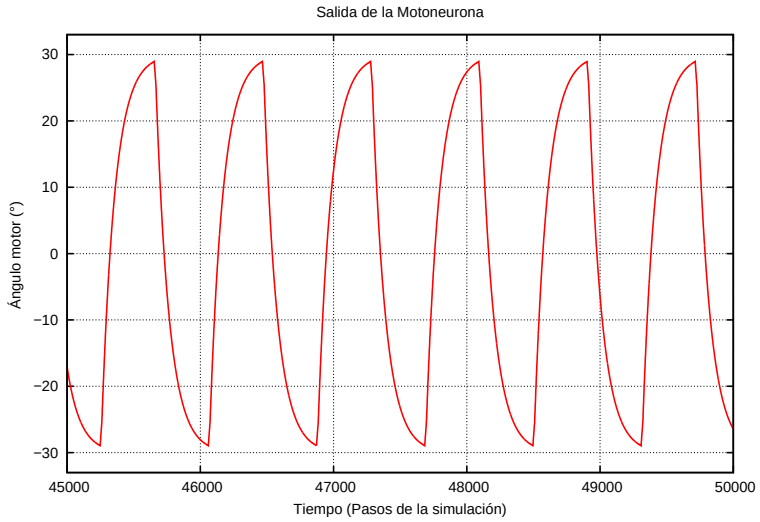


Figura 5.8: Salida de la motoneurona para $\alpha_M = 15$

momentos, al valor central. En cambio, si utilizamos un valor muy bajo para ν, al entrar las neuronas en estado silencioso no se situarán por debajo de este umbral y no dejarán de tirar nunca del servomotor. En ese caso la señal de salida de la motoneurona será igual a cero en todo momento.

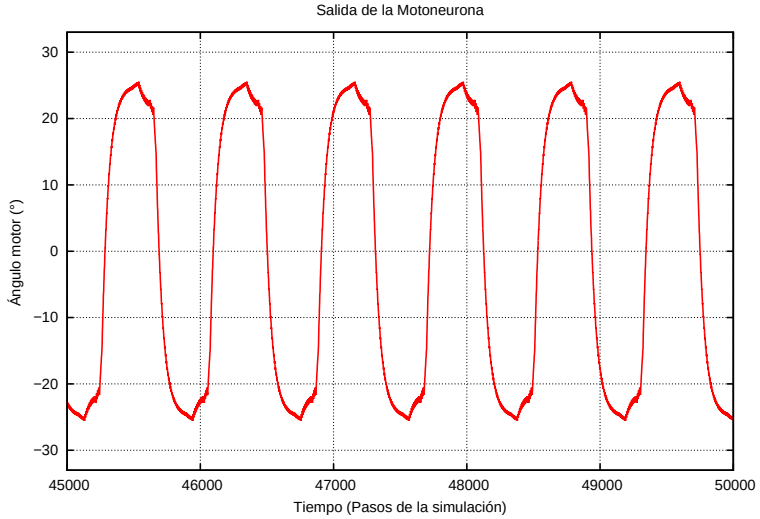


Figura 5.9: Salida de la motoneurona para $\nu = -0{,}5$

Una vez comprendida la función de cada uno de los parámetros de las ecuaciones de la motoneurona, y fijados a unos valores idóneos para cumplir los objetivos (los valores utilizados en la Figura 5.6, que son $\alpha_M = 30$, $\tau_M = 1{,}5$, $H = 0{,}001$ y $\chi = 30$), se puede proceder al siguiente paso de este proyecto. Llega el momento de unir varias unidades del módulo introducido en este apartado. Para ello se tendrán que estudiar las conexiones entre los distintos módulos, con el fin de poder establecer un desfase entre la señal de salida de las motoneuronas válido para el movimiento del robot.

5.5. Construcción del CPG completo

Tras haberse presentado y estudiado los módulos que, al ser juntados conforman un CPG, se procede a combinarlos (a nivel de módulo) para establecer una red neuronal que sea válida para satisfacer los objetivos que se han marcado. Para ello, se probaran distintas estructuras hasta encontrar una que sea correcta.

En el proceso de búsqueda de la estructura del CPG se tendrá que valer de herramientas que faciliten tal misión. Se utilizarán scripts bash para realizar simulaciones que realicen amplios barridos paramétricos (en la Sección 5.5.3), se desarrollará un calculador de fases que permita discernir las diferencias de fase entre los distintos módulos (en la Sección 5.5.2) y también se utilizará un simulador del robot, RoboSim, que permitirá hacerse una idea del movimiento obtenido antes de introducir el CPG en el robot (en la Sección 5.5.4).

Como ya se ha comentado en capítulos anteriores, el Generador Central de Patrones, CPG, es el elemento encargado de generar de manera coordinada los movimientos de los servo-motores que conforman al robot. Se debe buscar la estructura y parámetros válidos para que se generen señales que permitan mover el robot, para lo cual se necesitará alcanzar un desfase entre módulos estable y de determinado valor [59]. A continuación se describen los pasos realizados hasta conseguir encontrar un CPG que aune a todas estas características y sea capaz de conseguir que el robot se desplace.

Para que sean más claras las explicaciones que vienen a continuación, en el siguiente diagrama (Figura 5.10) se pueden apreciar las nomenclaturas y numeraciones que se van a utilizar:

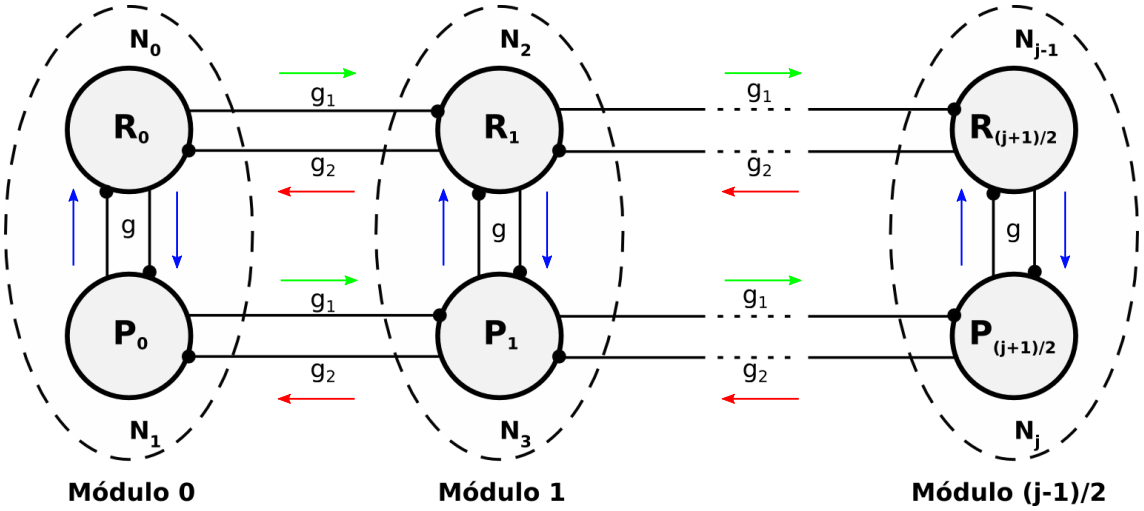


Figura 5.10: Diagrama de un CPG con la nomenclatura de neurona y módulo que se utiliza en secciones posteriores

5.5.1. Inhibición Simétrica entre módulos

En primer lugar se va a proceder a realizar la conexión entre módulos más sencilla que hay. Se trata de utilizar un sólo tipo de conductancia entre los módulos, siendo el mismo valor para las sinapsis en ambos sentidos. Esta estructura de CPG está explicada en detalle en el Apartado 4.5.1.

Conexión de dos módulos

Se procede a conectar dos módulos y se utiliza una conductancia $g = -0{,}029$ (valor que se estableció en el Apartado 5.3.2) para todas las sinapsis del modelo (tanto las que conectan

a los módulos entre sí como las que se encuentran dentro de los propios módulos, es decir, $g = g1 = g2 = -0{,}029$) (Véase Figuras 5.11 y 5.12). Con esto se busca que todos los módulos se encuentren desfasados π radianes respecto a sus vecinos, realizando acciones opuestas unos de otros.

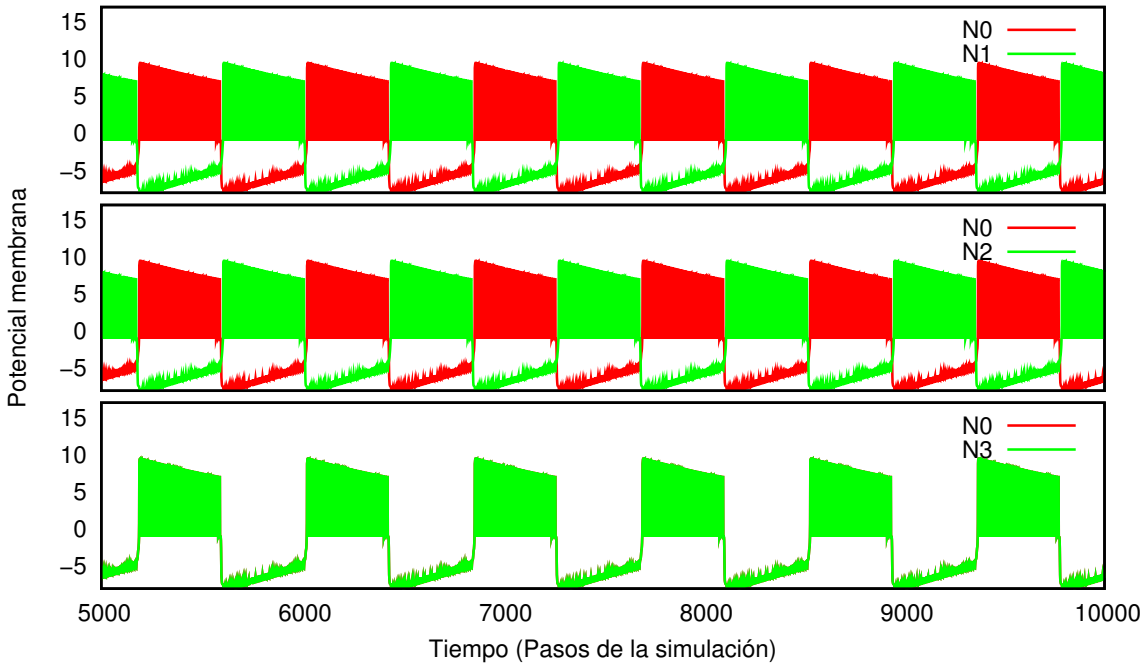


Figura 5.11: Comparativa de la Neurona #0 frente a las Neuronas #1, #2 y #3

En la figura se puede apreciar el potencial de membrana de la neurona #0 en todas las gráficas y el de las otras 3 neuronas repartido cada uno en una gráfica. La neurona #0 es vecina de las neuronas #1 y #2, y como se puede ver en las dos primeras gráficas si que se encuentran las distintas señales con el desfase deseado entre ellas (π radianes). En la tercera gráfica se aprecia que se encuentran en fase las neuronas #0 y #3. Esto entra también dentro del comportamiento esperado ya que entre las dos neuronas se encuentra un salto, siendo este la neurona #1 o la neurona #2 (hay dos caminos posibles). La neurona #1 y la #3 son vecinas así que están desfasadas π radianes, ocurriendo lo mismo con las neuronas #0 y #1, por lo que combinándolas se queda una diferencia de fase de 2π radianes, que es lo mismo que estar en fase (la fase se repite cada 2π).

Se procede a representar las motoneuronas correspondientes a los dos módulos que se han conectado (véase Figura 5.12). En esta imagen se puede apreciar como las señales se encuentran sincronizadas en antifase, ya que cuando una se encuentra en su valor máximo la otra está en el mínimo y continúan variando sus valores alternativamente. Esto se puede entender desde la perspectiva de la Figura 5.11, ya que las neuronas promotoras (también las remotoras), que son las neuronas #0 y #2, se encuentran también desfasadas π radianes, realizando su acción de tirar de la motoneurona hacia valores positivos en instantes opuestos.

Se ha comprobado que aunque las neuronas no estén directamente conectadas, las sinapsis que une unas con otras hace que les llegue la información de las neuronas que afectan a sus vecinas. Esta información se irá propagando de una neurona a otra, viajando desde un extremo del CPG al otro. En el caso de tener dos módulos se ha visto cómo parece que se mantiene constante el desfase. En el siguiente subapartado se va a comprobar si sucede lo mismo cuando se incrementa el número de módulos.

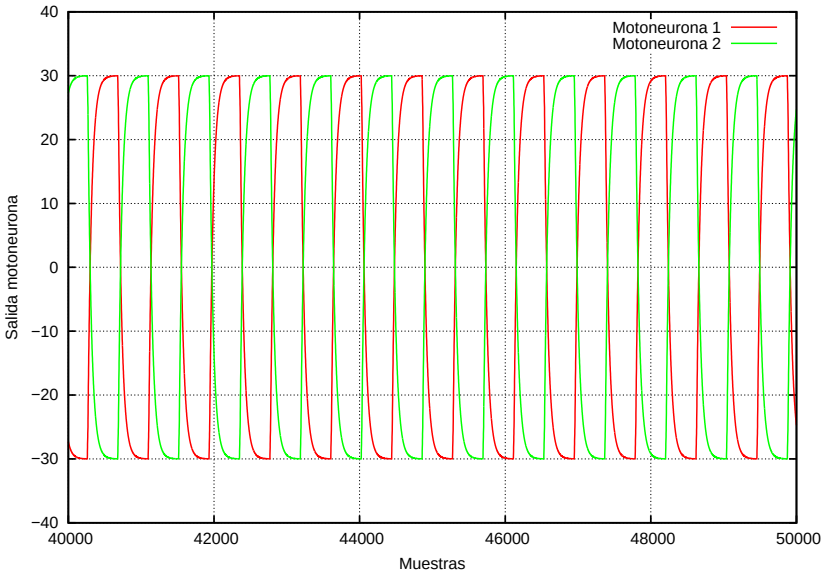


Figura 5.12: Señal de las motoneuronas en el Modelo con Inhibición Simétrica

Aumento del número de módulos del CPG

Se procede a ir añadiendo módulos al modelo para observar de que manera le afecta esto y si es capaz de mantener de manera estable el ritmo deseado. Tras realizarse simulaciones con 4, 6 y 8 módulos se representan los resultados obtenidos:

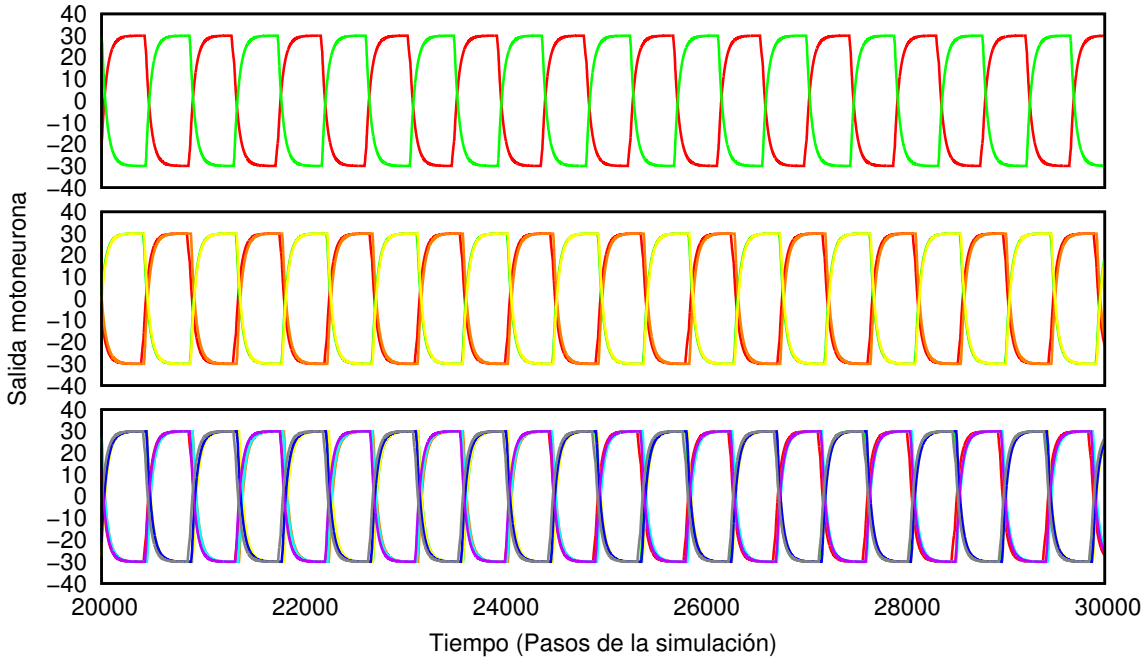


Figura 5.13: Señal de las motoneuronas para 4, 6 y 8 módulos respectivamente

En la figura se puede apreciar la señal de salida de las motoneuronas del modelo con inhibición simétrica. En la gráfica superior se utilizan 4 módulos, en la intermedia 6 y en la inferior 8. Se puede ir viendo cómo al aumentar el número de módulos la coordinación entre señales va disminuyendo, aunque esto no es realmente apreciable en estas gráficas. Esto se debe a que con esta sinapsis los módulos que se encuentran distanciados (por ejemplo el primero y el último) no llegan a establecer un acuerdo en la generación de sus ritmos. Esto, añadido al resto de desventajas (no siempre se alcanza un estado estable y el desfase que posteriormente

definirá el sentido de la locomoción no es siempre el mismo) de este modelo comentados en la Sección 4.5.1, significa que se tendrá que buscar otra estructura sináptica entre módulos para poder cumplir nuestro objetivo.

Además, se encuentra otro problema. Como se puede apreciar en la Figura 5.13 no se pueden analizar fácilmente las gráficas cuando se introduce un número alto de módulos. Para ello, y poder estudiar de una manera más objetiva los resultados, se va a desarrollar a continuación un programa que permita calcular el desfase entre los distintos módulos.

5.5.2. Cálculo del desfase entre módulos

En la búsqueda de poder analizar las señales de salida de las motoneuronas y así poder medir el desfase entre módulos se ha desarrollado el siguiente programa. Se trata de un solución diseñada específicamente para resolver el problema que nos ocupa y se ha tenido que realizar debido a la imposibilidad de medir visualmente el desfase que aparece entre los módulos del CPG.

A continuación se relata su funcionamiento. Una vez realizada la simulación neuronal, el código implementado se encarga de leer el fichero con los valores que van adquiriendo las motoneuronas a lo largo de la simulación y mediante el cálculo del paso por un punto común es capaz de calcular la diferencia existente entre las diferentes señales.

El pseudocódigo (el código está publicado en el Anexo B) que define la parte principal de este programa es el siguiente:

```
Bucle hasta que acaba el fichero con los datos de las motoneuronas:

        -En el caso que M1 pase por 0 (de manera ascendente):
                -Almacenamos el instante en la simulación en el que ocurre
                -En el caso que M1 ya ha pasado por 0 anteriormente:
                        -Calculamos longitud respecto al anterior paso por cero
                        -En el caso que M2 también haya pasado al menos una vez
                            por 0:
                                -Diferencia entre los pasos por 0 de ambas
                                    motoneuronas.
                                -Calculo desfase = 2*(dif_motoneuronas/
                                    longitud_M1)

        -En el caso que M2 pase por 0 (de manera ascendente):
                -Almacenamos el instante en la simulación en el que ocurre
```

Cuadro 5.7: Pseudocódigo del programa que se encarga de calcular el desfase

El programa primero se fija si la señal de las motoneuronas pasa por 0 de manera creciente, es decir, viene de valores negativos para ir hacia positivos, y cuando esto pase se queda con el instante de simulación en el que ha superado el punto 0. Cuando la señal de la motoneurona que haya pasado por el eje sea otra distinta a la del primer módulo, se calcula la distancia que separa este punto del último punto en el que la motoneurona #0 cruzó el eje (Véase Figura 5.14). Con esto, y la longitud del periodo de la primera motoneurona, se tendrá el desfase entre cada módulo y el primero de los módulos.

Lo que realmente se quiere es el desfase entre módulos vecinos con lo que se realizará un último cálculo. Para calcular el desfase de un módulo respecto a su anterior (véase Ecuación 5.1) se resta la diferencia de fase que separa al módulo menos lejano respecto al primer módulo (i-1,0) del más lejano respecto al primer módulo(i,0), y así se obtendrá el desfase entre módulos vecinos (i,i-1).

$$\varphi = \varphi_{i,0} - \varphi_{i-1,0} \tag{5.1}$$

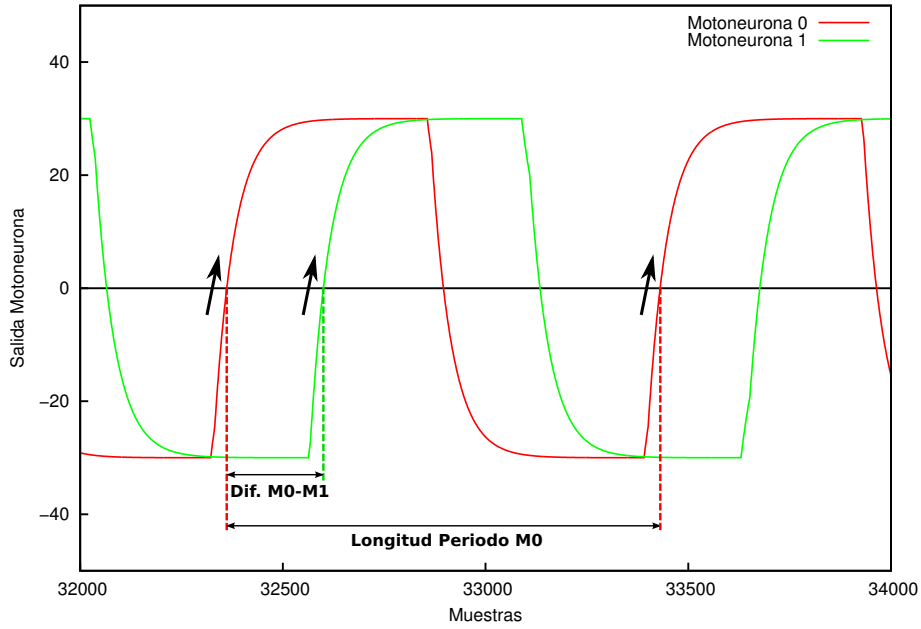


Figura 5.14: Diagrama explicativo para el cálculo del desfase

El último ajuste que se realiza se trata de acotar las fases entre π y $-\pi$ radianes, ya que aunque se obtengan valores superiores, se trata de valores contenidos en ese rango debido a que la fase es periódica con periodo 2π. Por tanto, para acotar el valor de las fases se debe sustraer o añadir 2π radianes en función del rango en que se encuentre la fase calculada (Véase Ecuación 5.2).

$$\varphi_{i,i-1} = \begin{cases} \varphi_{i,i-1} - 2\pi, & \varphi_{i,i-1} > +\pi \\ \varphi_{i,i-1} + 2\pi, & \varphi_{i,i-1} < -\pi \end{cases} \tag{5.2}$$

Se procede a realizar una prueba conectando dos módulos y se observa la diferencia de fase que muestra el programa que se ha diseñado. El resultado se muestra en la siguiente Figura:

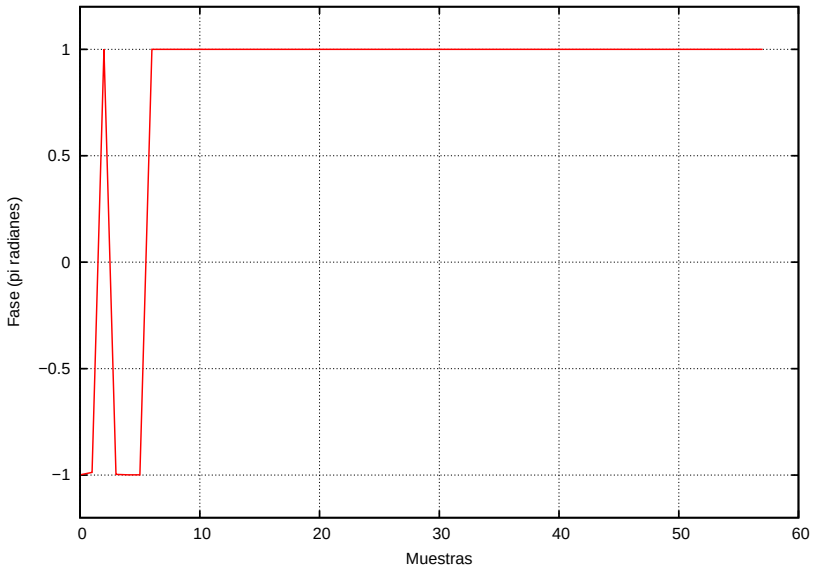


Figura 5.15: Diferencia de fase entre la salida de dos motoneuronas

Al analizarse la imagen, se puede apreciar cómo, al principio de la simulación, el modelo pasa por un estado en el que sus variables se encuentran estabilizándose. Una vez alcanzado ese

estado estable la diferencia de fase entre las motoneuronas pasa a ser π radianes, valor que se esperaba al haber conectado los módulos utilizando conductancias con valor $g = -0{,}029$. Esta gráfica corresponde a la diferencia de fase entre las señales de salida de dos motoneuronas, cuya salida se mostró en la Figura 5.15, y que se encuentran en anti-fase.

Tras el desarrollo de este calculador del desfase entre señales, ya se puede medir la diferencia existente en el comportamiento de los distintos módulos. Por lo tanto ya se puede continuar en la búsqueda de un CPG que permita hacer que el robot se desplace.

5.5.3. Inhibición Asimétrica entre módulos

Una vez realizadas conexiones básicas de prueba entre las neuronas y tras haberse conseguido una herramienta que permita profundizar en el análisis de la diferencia existente entre la fase de la señal de salida de las distintas motoneuronas, se va a comenzar a buscar los parámetros adecuados para desarrollar el CPG que permita cumplir los objetivos buscados.

A partir de ahora se va a buscar un valor para los parámetros *g1* y *g2*, manteniendo g con el valor encontrado anteriormente ($g = -0{,}029$) para poner en antifase a las neuronas remotora y promotora de cada módulo. Se buscará el valor de esos dos parámetros con la intención de encontrar un CPG que ofrezca una fase entre módulos estable y cuyo valor sea válido para conseguir desplazamiento por parte del robot.

El valor necesario para conseguir el desplazamiento del robot es un desfase del orden de $\varphi_{i,i-1} \approx 0{,}5\pi$ radianes. Al situarse próximo a este valor entre los diferentes módulos se conseguirá que el robot presente una ondulación que sea capaz de producir desplazamiento. Esto es así para los módulos internos del robot. Los módulos que se sitúan en el exterior del mismo, es decir, la cabeza y la cola del robot, no es necesario que estén sincronizados con tanta precisión como el resto de módulos entre sí. Esto es así porque su aportación a la generación del movimiento es pequeña al estar en los extremos. Este detalle es interesante ya que en nuestro modelo, los módulos de los extremos reciben un número menor de conexiones que el resto de módulos con lo que es normal que presenten un desfase diferente a ellos. Por tanto, aunque en las gráficas que se presentarán a continuación la diferencia de fase entre los módulos primero y segundo, $\varphi_{1,0}$ y antepenúltimo y último, $\varphi_{7,6}$, no será parecida a la del resto, no debe tomarse en consideración ya que no tendrá apenas repercusión en el resultado final.

Como se pudo apreciar en una sección anterior (en la Figura 5.10), *g1* se trata de la conductancia de la sinapsis que va desde la neurona remotora $n-1$ a la remotora n, o bien de la neurona promotora $n-1$ a la promotora n. Con *g2* pasa al revés, se trata de la conductancia de la sinapsis que vaya del módulo n al $n-1$. Por tanto, la sinapsis que se mueva hacia los módulos de la derecha llevará una conductancia *g1* y la que se mueva hacia los módulos de la izquierda tendrá *g2*. Antes solo se tenía una conductancia en la simulación, pero ahora, al tener varias, se aumentará el número de variables dedicadas a su valor. Son las siguientes:

```
g = -0.029;
g_aux = g;
g1 = ??
g2 = ??
```

Cuadro 5.8: Variables que almacenan las conductancias del modelo

Todas las conductancias se encuentran explicadas en la Figura 5.10 menos una, g_{aux}. Esta variable se utiliza para almacenar el valor inicial de g ($g = g_{aux} = -0{,}029$), ya que en g se va poniendo el valor de la conductancia de la sinapsis que se esté calculando en cada momento (g original, *g1* o *g2*). Una vez elegida la conductancia, se realizará el cálculo de la sinapsis tal cual se estaba realizando en las simulaciones anteriores.

Definido el funcionamiento de los elementos en esta estructura de CPG se procede a buscar valores en los parámetros que resulten válidos para cumplir el objetivo principal de conseguir desplazamiento en el robot.

Inicio de la búsqueda paramétrica

Se empieza observando el comportamiento del sistema ante una variación manual de sus parámetros. Se van probando diferentes valores y observando la probabilidad de ocurrencia que muestran los desfases entre los módulos en la señal de salida. Para ello se escoge un valor para los parámetros *g1* y *g2* y se realizan varias repeticiones de la simulación, en las que lo único que variará serán los valores iniciales de los parámetros x e y de cada neurona. Además, se introduce un retardo entre cada simulación para que no se repitan los valores iniciales de los parámetros (aunque se eligen de manera aleatoria, esta elección la realiza el PC utilizando para su cálculo, entre otras cosas, la hora del sistema). Se va a prestar atención a la diferencia de fase entre los módulos #4 y #5, que son los módulos que se encuentran en el centro de la estructura (véase Figura 5.28). Si el desfase entre estos dos módulos es el deseado, habrá una gran probabilidad de que el resto de módulos se encuentren también en el desfase adecuado.

Por ejemplo, realizando una prueba con $g1 = -0{,}019$ y $g2 = 0{,}035$, se pueden observar los siguientes resultados:

Desfases mostrados	**Porcentaje de ocurrencia (por cada simulación)**					
0.27π	90%	85%	80%	80%	90%	85%
2π	10%	15%	20%	20%	10%	15%

Cuadro 5.9: Ocurrencia de fases para *g1=-0.019* y *g2=0.035*

Este recuadro contiene los resultados de varias ejecuciones (6 ejecuciones) del programa que se acaba de comentar. En cada una de las ejecuciones se ha simulado el sistema neuronal 100 veces (con condiciones iniciales aleatorias) utilizando los valores para *g1* y *g2* indicados, y se ha calculado el porcentaje de ocurrencia de la fase obtenida al llegar a un estado estable en el sistema. En este caso se ve que el sistema tiende a dos fases estables, siendo estas dos opciones $0{,}27\pi$ o 2π radianes, variando la ocurrencia de estas fases a lo largo de las 6 ejecuciones del programa. Se puede apreciar como en la primera ejecución se tienen 90 casos de $0{,}27\pi$ radianes y 10 casos de 2π radianes, obteniendo en el resto de ejecuciones otros resultados. Los resultados se repiten en algún caso aunque suelen ser diferentes. En este caso no se observa mucha variación entre ejecuciones, pero hay otros casos en los que se produce una gran variación en la distribución de las fases. Por ejemplo, utilizando $g1 = -0{,}02$ y $g2 = 0{,}0197$, se obtienen los siguientes resultados:

Desfases mostrados	**Porcentaje de ocurrencia (por cada simulación)**					
0.41π	100%	75%	70%	45%	65%	70%
1.15π	0%	0%	10%	50%	25%	5%
2π	0%	25%	20%	5%	10%	25%

Cuadro 5.10: Ocurrencia de fases para *g1=-0.02* y *g2=0.0197*

En este caso se ve mucha variación en los resultados entre cada ejecución del programa. Atendiendo a los componentes del sistema, se puede afirmar que esta molesta variación en el

porcentaje de posibles desfases entre módulos se debe únicamente a la variación de los valores iniciales aleatorios. Además de este detalle, no se pueden obtener más conclusiones de estas simulaciones ya que la variación entre diferentes ejecuciones es tan grande que no permite unir los resultados y observar algo en común que apunte a valores válidos.

A continuación se va a tratar de ajustar el valor de esas condiciones iniciales intentando conseguir una mayor estabilidad en los sistemas.

Ajuste de las condiciones iniciales

Se ha observado que debido a los valores aleatorios generados, se están obteniendo resultados en la simulación que difieren en gran medida unos de otros. Estudiando el método que se utiliza para generar estos valores iniciales se hace evidente que, dentro del rango que se utiliza, se permite un número muy pequeño de valores posibles.

El valor inicial de los parámetros se calcula de tal manera que se permite que x e y solo puedan tomar los valores -1, 0 y $+1$. Debido al número tan reducido de opciones, los valores de neuronas vecinas suelen coincidir y al tener un estado similar ambas, no se acaban poniendo de acuerdo nunca ya que al intentar comunicarse las dos a la vez ninguna llega a escuchar a la otra.

Se va a continuar utilizando una generación aleatoria del valor inicial de los parámetros x e y de cada una de las neuronas involucradas en el modelo, aunque se procede a aumentar el número de posibles valores iniciales dentro del rango que se dispone. Gracias al cambio en la generación aleatoria de estos valores, se ha ampliado el rango de valores posibles, pudiendo obtener números desde -2 a $+2$. Además, ahora no solo se tienen los valores enteros comprendidos en ese rango, sino que se tiene la posibilidad de que el número tenga hasta 3 decimales (incremento de 0.001 entre posibles números). Por tanto, se ha pasado de poder tener 3 valores diferentes a la capacidad de generar 4000 valores aleatorios diferentes. La probabilidad de que se repita el estado entre neuronas vecinas ahora es muy remoto con lo que se ha solucionado el problema que se presentó en el apartado anterior.

Solucionado el asunto de la generación de los valores iniciales de las variables de la neurona, se va a continuar buscando el valor que se necesita en las conductancias del sistema. El procedimiento utilizado en el apartado anterior no es práctico a la hora de buscar en profundidad ya que se trata de la búsqueda manual de estos parámetros. A continuación se va a desarrollar un método que, mediante la automatización del proceso de búsqueda, explore de manera exhaustiva los posibles valores de las conductancias y encuentre los valores que cumplan los requisitos que se necesitan.

Búsqueda del desfase óptimo

En este apartado se continúa también en la búsqueda de un desfase adecuado entre los módulos centrales de la estructura. Se utilizaran varias estructuras, variando únicamente el número de neuronas presentes (2, 4, 6 y 8 motoneuronas), y se verá como va evolucionando la presencia de conductancias válidas.

En primer lugar se ha calculado el desfase medio existente entre los módulos centrales (valor medio de los desfases entre módulos) a lo largo de toda la simulación. Esto se hace para todas las posibles conductancias *g1* y *g2* incluidas en el rango -0.1 a 0.1 y con un incremento de 0.005 entre cada posible conductancia. Una vez obtenidos estos valores se han representado en los siguientes mapas, donde se puede apreciar las zonas (delimitado mediante un círculo de líneas discontinuas) en las cual se podrán obtener los resultados deseados.

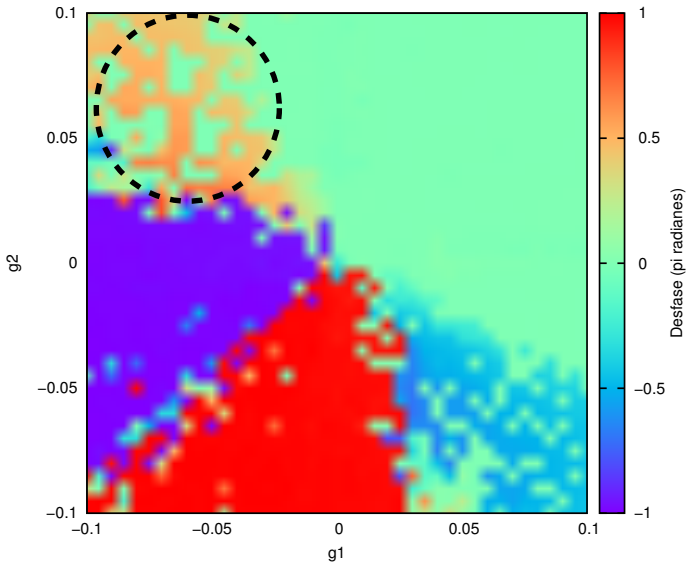


Figura 5.16: Mapa de conductancias para 2 módulos

Este primer mapa (véase Figura 5.16) representa los resultados obtenidos en una estructura en la que únicamente se han utilizado 2 módulos (4 neuronas). Es un modelo muy sencillo pero que luego permitirá ir observando la evolución de los parámetros válidos según se va aumentando el número de neuronas en la red neuronal. Los resultados válidos son los que ofrecen un desfase próximo a $0{,}5\pi$ radianes, siendo también válidos los que se encuentran en $-0{,}5\pi$ radianes ya que ofrecen el mismo movimiento válido de los módulos pero desplazando el robot en el sentido contrario. Se puede apreciar cómo tiene una zona, acotada por $-0{,}1 \leq g1 \leq -0{,}025$ y $0{,}025 \leq g2 \leq 0{,}1$, en la que se encuentra una alta probabilidad de obtener un desfase entre módulos cercano a $0{,}5\pi$ radianes. Se encuentra, en los parámetros $0{,}025 \leq g1 \leq 0{,}1$ y $-0{,}1 \leq g2 \leq -0{,}025$, otra zona similar pero que ofrece el resultado negativo válido, con un desfase cercano a $-0{,}5\pi$ radianes. Estas dos zonas comentadas se corresponden con la esquina superior izquierda y la esquina inferior derecha.

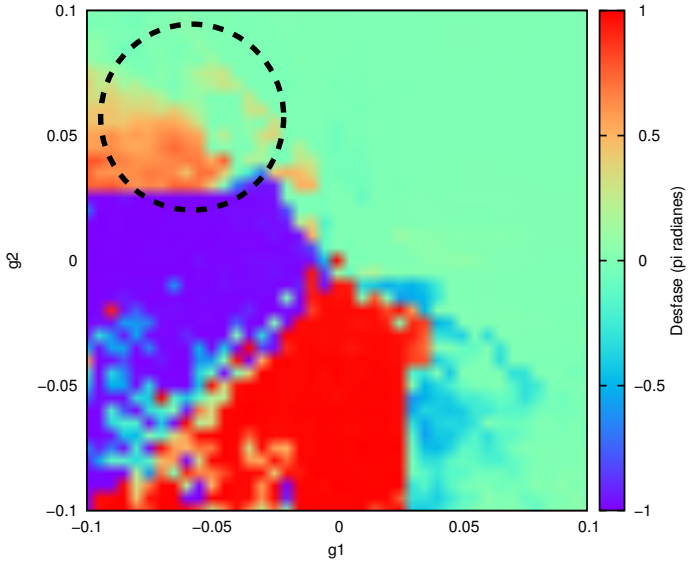


Figura 5.17: Mapa de conductancias para 4 módulos

Al observar los resultados de utilizar 4 módulos (8 neuronas) (véase Figura 5.17) se puede apreciar que las zonas en las que encontramos los parámetros válidos continúan estando en el mismo lugar que al utilizar únicamente 2 módulos. La cantidad de parámetros válidos dentro de esa zona disminuye, aunque también esos puntos se encuentran menos dispersos que en el

caso anterior. Además, pueden encontrarse puntos válidos en torno a la diagonal que va de la esquina inferior izquierda al centro de la gráfica. Estos valores válidos se encuentran aislados entre puntos que dan valores muy diferentes, los cuales no son válidos, con lo que se van a desechar esas posibles soluciones y se escogen únicamente con las zonas homogéneas comentadas anteriormente.

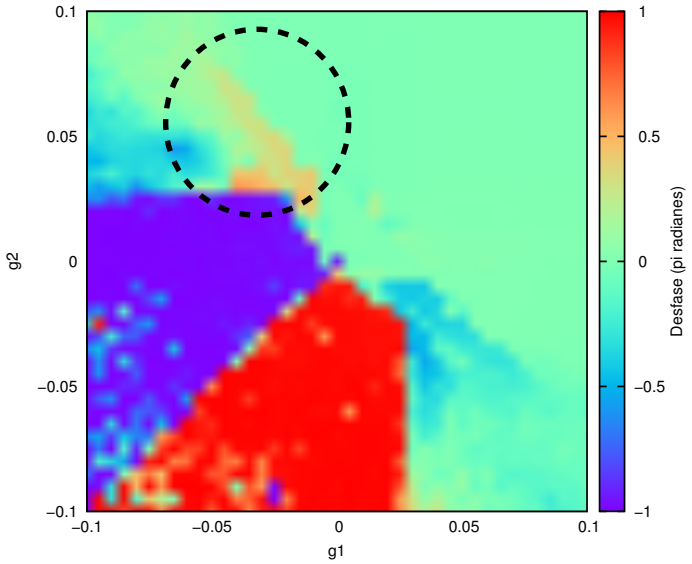


Figura 5.18: Mapa de conductancias para 6 módulos

En este caso (véase Figura 5.18), ya con 6 módulos (12 neuronas), se observa como la densidad de puntos que incluyan las características deseadas continúa disminuyendo. Los resultados siguen encontrándose en las mismas zonas paramétricas comentadas en las gráficas anteriores.

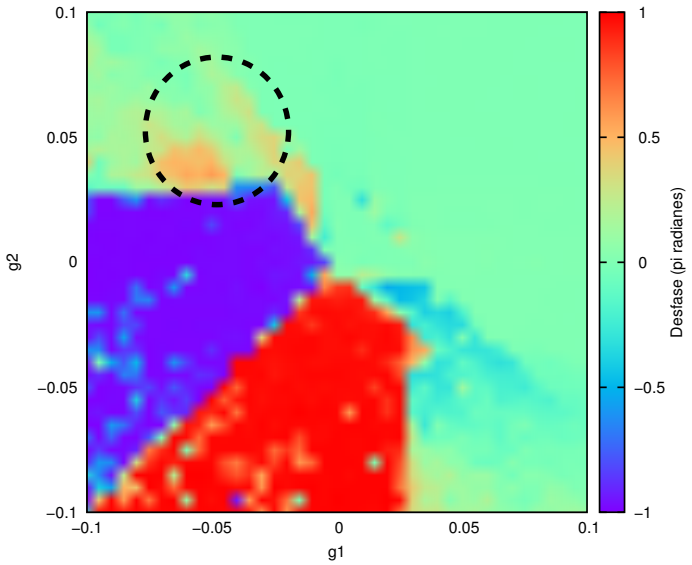


Figura 5.19: Mapa de conductancias para 8 módulos

Por último, se realiza el mapa correspondiente a la estructura con todos los módulos que se necesitan (véase Figura 5.19), es decir, con 8 módulos (16 neuronas). En la imagen se aprecian unos resultados similares a lo visto en casos anteriores en los que se utilizaba un número menor de módulos. Entre los mapas de 6 y 8 módulos apenas se muestra diferencia en cuanto a la presencia y distribución homogénea de los puntos válidos. Puede decirse que se ha alcanzado a un punto en el que, aún aumentando el número de módulos del CPG, no se alteran los resultados

ofrecidos en relación al desfase de los módulos internos.

Para poder apreciarse en mayor detalle cuales son los parámetros óptimos dentro de la zona señalada, se vuelve a realizar el mapa de fases para 8 módulos, solo que ahora analizándose únicamente el área comprendida por $-0{,}1 \leq g1 \leq -0{,}024$ y $0{,}024 \leq g2 \leq 0{,}1$ (véase Figura 5.20). Se analiza únicamente el área con valores cercanos a $-0{,}5\pi$ radianes ya que el otro área mostrará una situación similar sólo que centrada en $0{,}5\pi$ radianes. En este caso se reduce el incremento entre posibles conductancias a 0.002.

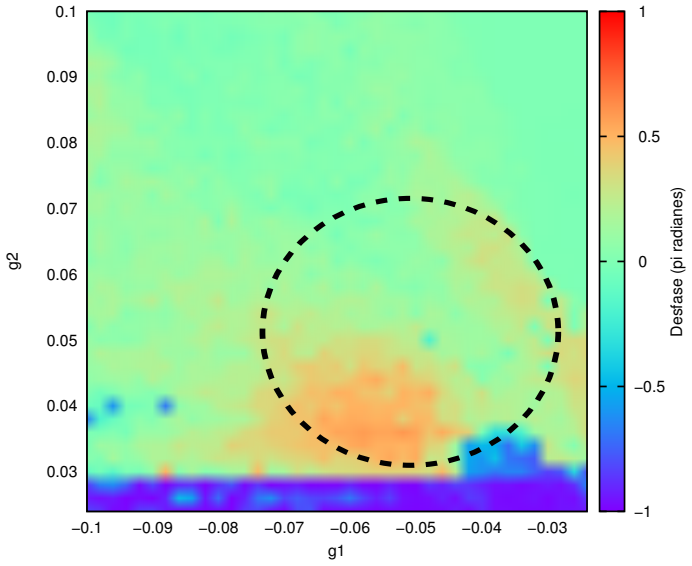


Figura 5.20: Mapa de conductancias ampliado para 8 módulos

En la anterior imagen se puede apreciar el desfase existente entre los módulos centrales (en este caso el cuarto y el quinto módulo), pero no se tiene información acerca del desfase entre el resto de módulos. Se ha repetido la simulación anterior pero en este caso también observando el resto de desfases de los módulos internos. No se introduce en este análisis al primer módulo ni tampoco al último ya que, como se ha comentado anteriormente, ofrecen unas características diferentes. Para ello se obtiene el valor medio de la suma de los desfases de los módulos internos, como se muestra en la ecuación a continuación:

$$\varphi_{\text{interna media}} = \frac{\varphi_{23} + \varphi_{34} + \varphi_{45} + \varphi_{56} + \varphi_{67}}{5} \tag{5.3}$$

En el siguiente mapa se muestra el valor obtenido para el área de interés:

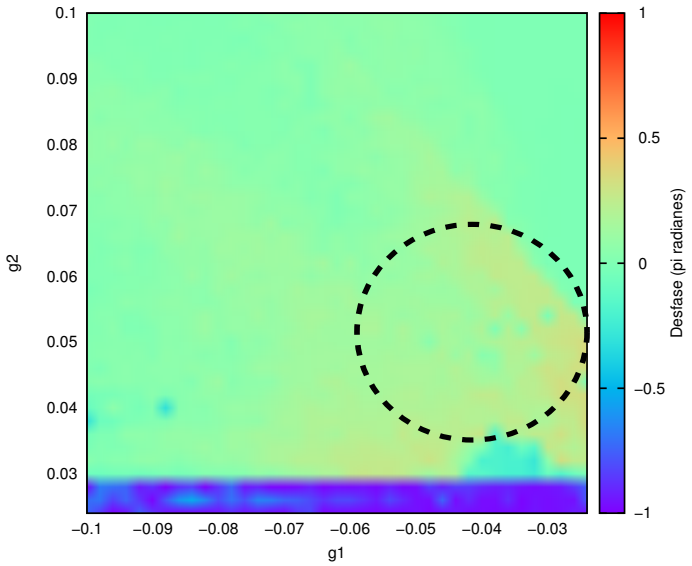


Figura 5.21: Mapa de conductancias ampliado para 8 módulos y con el valor medio de los desfases internos

En esta última imagen se observa como, al filtrar en función del valor de todos los módulos centrales, las soluciones posibles se reducen en gran medida. Se analiza el fichero del que se sacan los datos para hacer esta gráfica y se extrae el mejor resultado que se encuentra en su interior. Los parámetros que ofrecen el mejor resultado (un desfase más cercano al valor deseado que el mostrado por el resto de conductancias) son $g1 = -0{,}024$ y $g2 = 0{,}048$, que presentan un desfase medio cercano a -0.5 radianes. Estas conductancias dan lugar a la siguiente gráfica, en la que se muestra la diferencia de fase entre los módulos que conforman al robot:

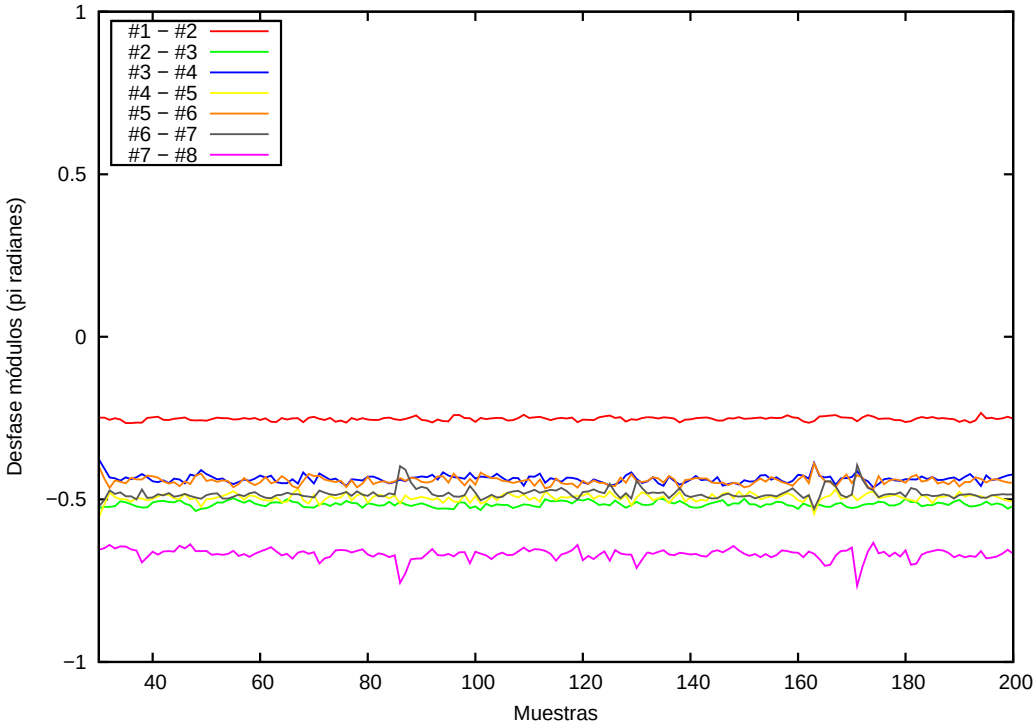


Figura 5.22: Desfase mostrado con 8 módulos utilizando conductancia óptima

Los resultados obtenidos cumplen los requisitos que se impusieron para encontrar un CPG que permitiese hacer que el robot se desplace. Se tiene la necesidad de probar los parámetros ya que, aunque atendiendo a las fuentes en las que se ha basado el trabajo son correctos, realmente

no se sabe si son válidos o no, o bien cuales son mejores que otros, ya que nunca se han probado en el robot. En el siguiente apartado se va a simular el comportamiento del robot para poder probarse la validez de los resultados.

5.5.4. Simulación del comportamiento del robot

Para simular el comportamiento del robot se prueba a utilizar el modelo desarrollado por Juan González para el programa ODE [87]. Este modelo simula de manera realista al robot gracias a la fiel recreación que hace ODE de las leyes de la física que se ven implicadas (rozamiento, gravedad, fuerza). Se comprueba que el modelo desarrollado por Juan no es compatible con la versión actual de ODE, ni tampoco es compatible con las versiones actuales de alguno de los programas a los que recurre ODE para funcionar. Debido a esto, y a la carga adicional que supondría modificar los ficheros fuente del proyecto de Juan González, se decide buscar una alternativa.

Fernando Herrero suministra un simulador desarrollado por él mismo, RoboSim, que utilizó para observar el comportamiento simulado de CUBE REVOLUTIONS. Este programa está escrito en lenguaje C++, y utiliza BULLET [88] para implementar las físicas y OGRE [89] para implementar el apartado gráfico en la simulación. El simulador no incluye unas características físicas tan completas como ODE, aunque este aspecto no es un problema en este caso ya que lo único que lo que se quiere es verificar que el robot puede desplazarse. Sería un problema si se quisieran sacar detalles precisos de la simulación, ya que no responde tan fidedignamente como ODE ante una situación real, pero basta con observar al robot alejarse del punto en el que se encuentre situado inicialmente.

El programa trae por defecto la generación de la secuencia de movimiento en tiempo real mediante el uso de una función seno (código en el Anexo B), siendo esta desplazada para cada módulo respecto a su módulo vecino anterior. Tras observarse que el robot se desplaza de manera óptima cuando sus módulos se encuentran desfasados $\pi/2$ radianes, se procede a modificar la generación del movimiento. Se hace que el programa obtenga la información de como debe mover los módulos a partir de un fichero generado en una simulación del CPG que se habrá obtenido previamente.

La primera prueba que se realiza no es satisfactoria. Según comienza la simulación, el robot se despega del suelo y comienza a realizar movimientos violentos sin una muestra clara de coordinación entre sus módulos. Comparando la señal que se obtiene de la simulación del CPG y la que utiliza Fernando Herrero como ejemplo en RoboSim, se aprecia que hay una gran diferencia entre los periodos y amplitudes de ambas señales. Al ser la señal de salida del CPG mucho más rápida, los módulos comienzan a moverse muy deprisa y generan una inercia en el movimiento que hace que el robot comience a realizar movimientos anormales. Esto no podría ocurrir en la realidad. La velocidad de movimiento que alcanzan los módulos simulados no pueden alcanzarla los módulos reales ya que el servo-motor tiene una velocidad de giro limitada, y por tanto se vería un comportamiento no deseado al no poder ofrecer el robot los valores dados por las motoneuronas en los márgenes de tiempo solicitados.

Para solucionar este problema se intenta ajustar lo máximo posible el periodo y amplitud de la señal que se genera mediante el CPG, tratando de hacerla lo más parecida posible a la señal que venía como muestra (Véase Figura 5.23). Tras varios ajustes se consigue implementar un código que cumpla los requisitos (código en el Anexo B). Se ha tenido que muestrear la señal para quedarse únicamente con 1 de cada 13 valores que se leen del fichero (posteriormente se modificará el CPG para que no haya que hacer este muestreo) y también se ha tenido que reducir la amplitud 10 veces su valor.

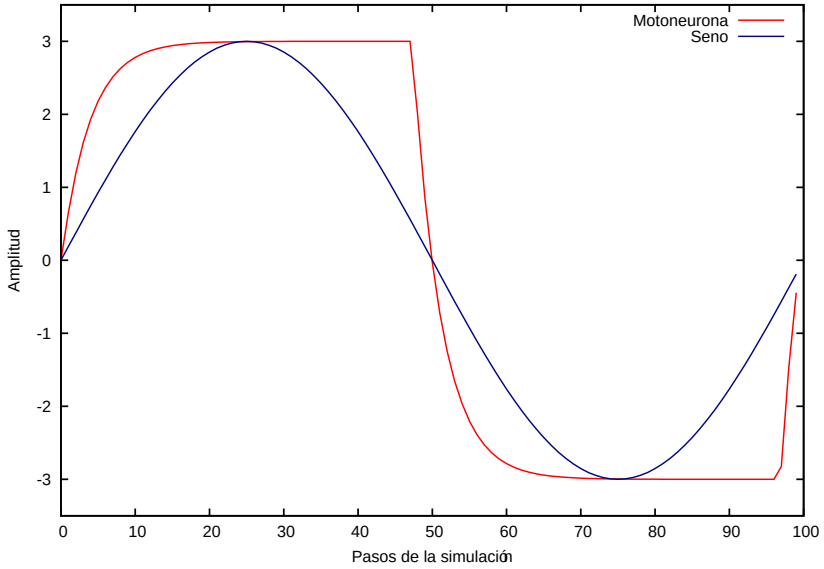


Figura 5.23: Comparativa señal motoneurona con función seno

Al probarse esta modificación de la señal en el simulador puede observarse como el robot se desplaza adecuadamente, de manera muy parecida a como lo hacía con la función seno proporcionada por defecto en el simulador. Se puede apreciar el movimiento en la siguiente secuencia de imágenes tomadas de una ejecución en el simulador:

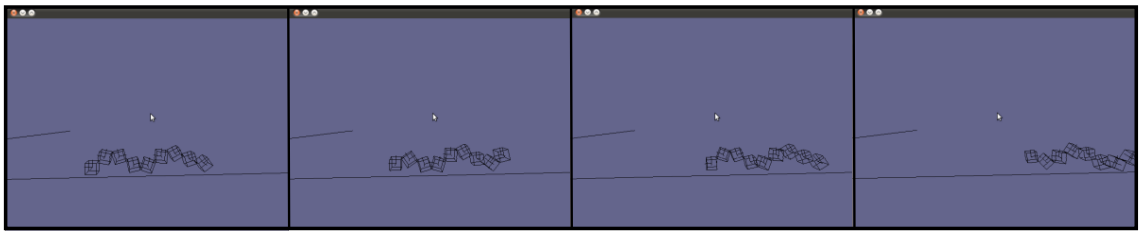


Figura 5.24: Robot desplazándose en RoboSim

Por otro lado prueba a hacerse un cambio entre las conductancias g1 y g2 (como ya se comentó en la Sección 4.5), y se aprecia cómo ahora el robot se desplaza en el otro sentido manteniendo el mismo desfase absoluto entre módulos. Al hacer un cambio entre estos dos parámetros, la simulación atraviesa un nuevo periodo de tiempo durante el cuál sus señales no se encuentran estables. Pasado este tiempo, las señales vuelven a ajustarse pero presentando el desfase necesario para que el robot se mueva en el sentido contrario. El periodo de ajuste es muy largo así que después de obtener el fichero de la simulación del CPG, y antes de introducirlo en RoboSim, se procede a sustraer la parte del archivo correspondiente a esta situación momentánea de inestabilidad (Véase Figura 5.25). Una vez eliminados estos valores y ajustadas las condiciones iniciales de las neuronas el cambio del sentido del desplazamiento se realizará de manera instantánea.

Por último, se adapta el código (en el Anexo B) para hacer que la simulación del CPG se ejecute en tiempo real respecto a RoboSim. Se realiza mediante el uso de tuberías MKFIFO [90] de los sistemas UNIX. Se utilizan para poder comunicar al proceso encargado de simular el CPG con el proceso principal del simulador, comunicándose ambas partes al escribir mensajes a través de la tubería. Se ha corregido la duración del periodo de las motoneuronas y su amplitud mediante la modificación de los parámetros del CPG, lo que evita tener que muestrear y escalar los datos. Con esto se comprueba que el sistema neuronal que hemos diseñado se puede ejecutar en tiempo real ofreciendo los mismos resultados en RoboSim que los obtenidos leyendo el fichero generado tras la simulación del CPG.

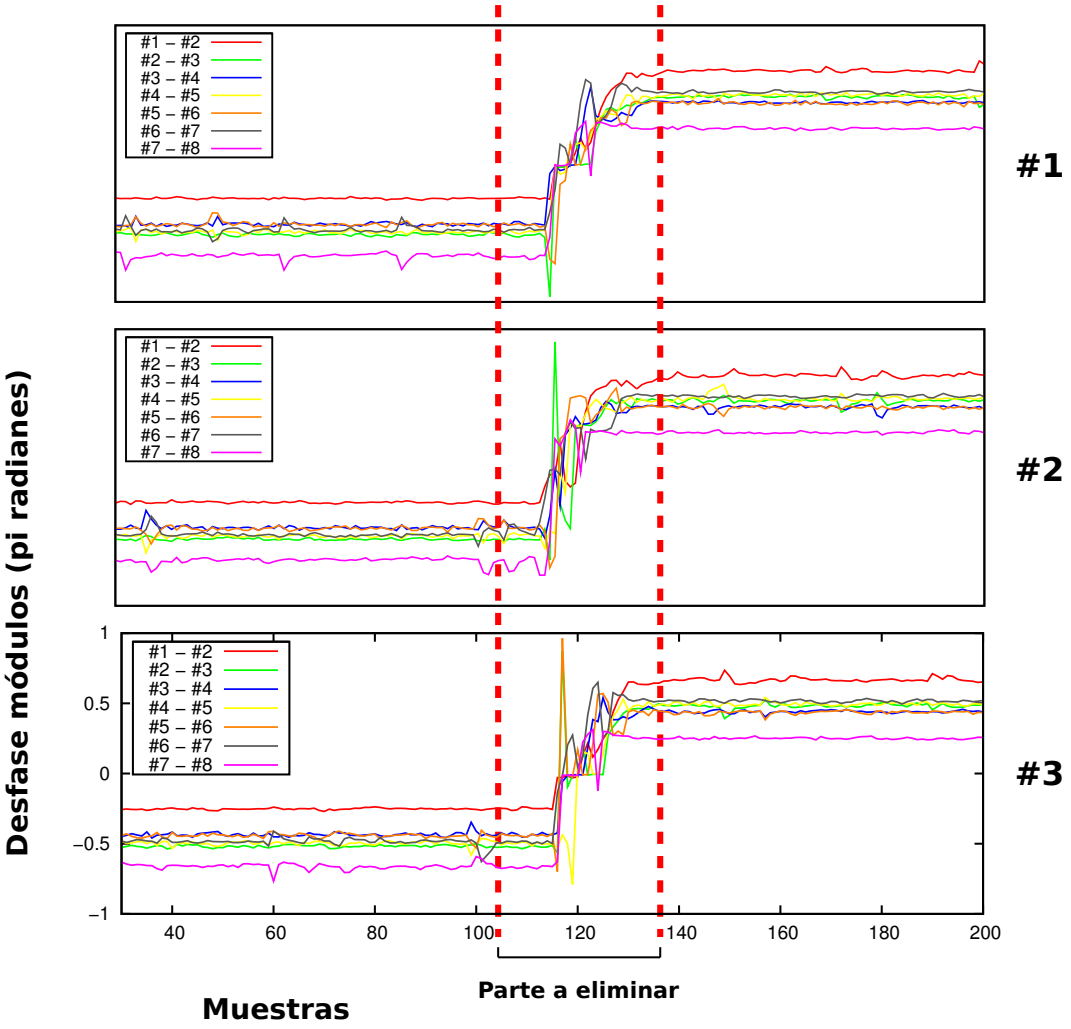


Figura 5.25: Swap de las conductancias

5.6. Integración del CPG en el robot

Una vez ya se tiene un CPG válido y que ha sido probado en ROBOSIM, donde se ha podido verificar que mueve correctamente al robot simulado, se va a introducir este CPG en el robot y así poder comprobar que ofrece buenos resultados en un entorno no simulado.

5.6.1. Introducción del CPG en el microcontrolador: SkyPIC

Desde el inicio del proyecto se dispone de la tarjeta SkyPIC [91], ya que se trata de la tarjeta (introducida en el Apartado 3.10) con la que Fernando Herrero, Damián Zamorano e Ionut Urziceanu realizaron sus respectivos estudios [2, 3, 42]. Ellos utilizaban la tarjeta a modo de cliente y el PC a modo de servidor. De esta manera, la simulación del CPG la realizaban en el PC e introducían una conexión entre los dos elementos para que el ordenador le informase a la SkyPIC de las posiciones de los diversos motores. Primero realizaban la simulación del CPG, almacenaban en un fichero toda la información relevante generada a lo largo de la simulación, y una vez terminada pasaban estos datos a la tarjeta. La tarjeta únicamente se encargaba de recibir los valores de las posiciones de cada uno de los motores y los movía en función de estos valores.

Ahora se va a utilizar la SkyPIC pero de una forma distinta. Solo se quiere una conexión PC-SkyPIC para programar a la tarjeta y que esta de manera autónoma sea capaz de simular el CPG, haciéndolo además en tiempo real.

Lo primero que se va a hacer es introducir un programa, star-servos8 [92], realizado por Juan González expresamente para su tarjeta. Este programa funciona de forma CLIENTE-SERVIDOR, y permite cambiar desde un cuadro de mando (véase Figura 5.26) situado en el PC el ángulo de giro de los servos controlados por la SkyPIC. El motivo por el cual se quiere utilizarlo es que permite saber a que servo se corresponde cada señal enviada desde el microcontrolador (la matriz de interconexión de Damián Zamorano está oculta y no se tiene certeza de que esté conectado todo como indica en su PFC) y también permite ajustar el valor central de los módulos (el error presente se mostró en la Tabla 4.1).

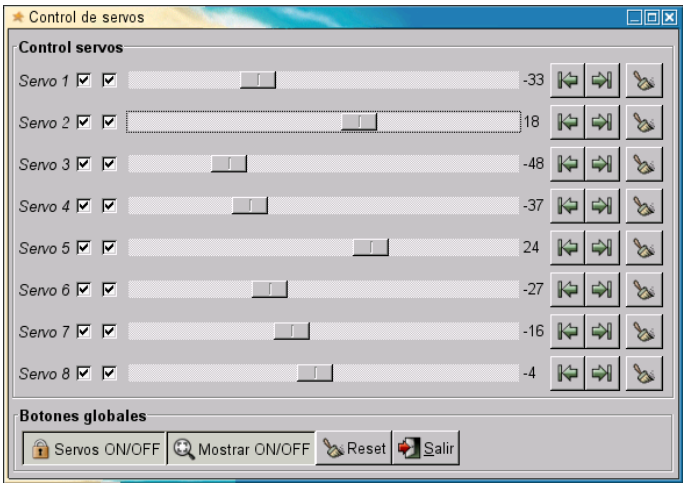


Figura 5.26: Cuadro de mando de servos8

Para poder utilizar este programa, y también para poder descargar programas a la SkyPIC, se han tenido que utilizar los siguientes programas (todos proporcionados en la web de Juan González [1]):

- **libStargate:** Librería que permite realizar programas que puedan hablar con clientes SKYPIC a través del proyecto Stargate desarrollado por Juan González [93].
- **libIris:** Librería en python para descarga de programas en la SkyPIC [94].
- **SDCC:** Compilador para poder traducir el código C en un fichero en hexadecimal, que será el archivo que introduciremos en la SkyPIC y de donde obtendrá la información de que acciones debe realizar[95].
- **Pydownloader:** Aplicación para descarga de programas en la SkyPIC. Mediante su uso introduciremos el fichero en hexadecimal en el microcontrolador[96].

Se procede a introducir en la SkyPIC el cliente del programa star-servos8, denominado servos8, mediante el uso de Pydownloader (Véase Figura 5.27). Utilizando el servidor en el PC, se modifica el valor de los servos y se observa que la disposición de los módulos no está relacionada directamente con la numeración de los servos en el servidor (el módulo 1 en el robot no se corresponde con el servo 1 en el programa-servidor). No se conoce la razón de esta no correspondencia en la numeración ya que se está trabajando con el robot que utilizó Damián Zamorano en su Proyecto Final de Carrera y no incluye en la documentación ningún comentario acerca de este asunto. Debido a eso se debe obtener esta numeración de manera experimental probando a mover cada uno de los servos mediante el programa star-servos8.

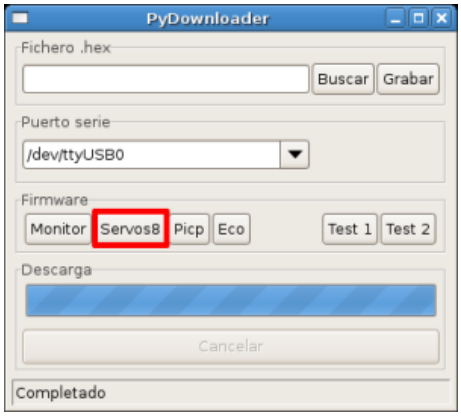


Figura 5.27: Descarga de programas a SkyPic mediante Pydownloader

La correspondencia entre la numeración de los módulos en el robot y la numeración de los servos en el servidor es la siguiente:

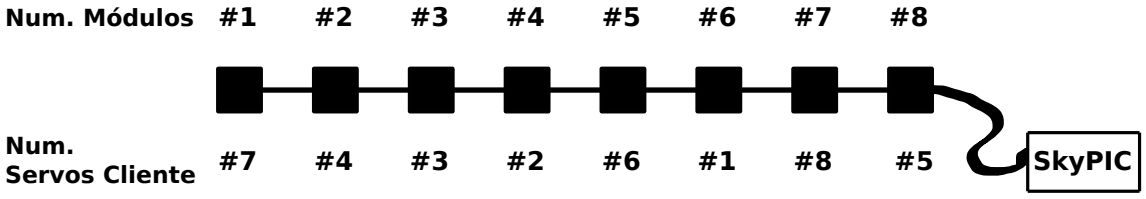


Figura 5.28: Correspondencia de numeración de los módulos con la de los servos en el programa servos8

Por otro lado, si se introduce el valor central para cada servo, se observa como el robot se dispone en forma de arco. Se van probando valores en cada uno de los servos hasta encontrar el punto en el cual el valor del ángulo del servo se corresponda con el punto central del módulo (es decir, poco a poco el robot se queda horizontal respecto al suelo). Una vez encontrados todos estos valores, los cuales se muestran en el Apartado 4.1, se anotan como valores centrales. Este valor central-simulado será el punto alrededor del cual el servo gire, estando limitado el ángulo máximo de giro a los 90 iniciales (los 90 grados que puede girar el servo desde su punto central a cualquiera de los dos lados) menos el ángulo que se haya corregido.

Una vez conocidos estos datos es hora de introducir el CPG en la SkyPIC. Para ello, se parte del código fuente de servos8. Se retira del código la parte que se encarga de escuchar al servidor situado en el PC, quedándose únicamente con la parte que se encarga de manejar a los servos. Se realizan pruebas en las que, sin haber llegado a introducir el CPG, prueba a hacerse que la SkyPIC haga variar el valor de los servos de manera autónoma, obteniendo un resultado satisfactorio.

Al realizar el análisis de la memoria necesaria para introducir el CPG en un microcontrolador, se observa que la memoria de datos que incluye el PIC16F876 (microcontrolador de la SkyPIC de la casa Microchip [76]) no es suficiente para almacenar el número de variables que se necesita a lo largo de la simulación.

Las características, en cuanto a su memoria, del PIC16F876 son:

- Memoria de programa FLASH: 8 kBytes
- Memoria RAM de datos: 368 Bytes
- Memoria EEPROM de datos: 256 Bytes

Las variables necesarias en la simulación (sin haber incluido aún las referentes a la nariz electrónica) y sus respectivos tamaños se pueden ver en el Cuadro 5.11. Estas variables se

utilizan para conformar a las neuronas, las sinapsis que las conectan, las motoneuronas, las variables intermedias, los contadores y las posiciones de los servos. Su uso se puede observar en el código incluido en el Anexo B.

Variable	Tipo Dato	Tamaño
g	float	4 Bytes
g1	float	4 Bytes
g2	float	4 Bytes
g_aux	float	4 Bytes
x[16]	float	64 Bytes
x_untouched[16]	float	64 Bytes
xn	float	4 Bytes
y[16]	float	64 Bytes
yn	float	4 Bytes
yx	float	4 Bytes
In[16]	float	64 Bytes
m[8]	float	32 Bytes
mn	float	4 Bytes
num_entradas[16]	unsigned int	64 Bytes
entrada[16][3]	unsigned int	192 Bytes
c	unsigned int	4 Bytes
i	unsigned int	4 Bytes
j	unsigned int	4 Bytes
k	unsigned int	4 Bytes
pos_servo[8]	unsigned char	16 Bytes
centro_servo[8]	unsigned char	16 Bytes
		TOTAL = 624 Bytes

Cuadro 5.11: Tamaño de las variables necesarias para simular el CPG

Como se puede apreciar en la anterior tabla, no sirve el PIC16F876 a la hora de poder simular el CPG de forma autónoma ya que no se tiene espacio suficiente y por tanto no se dispone de todas las variables necesarias para realizar los cálculos internos. Aún optimizando el número de variables a utilizarse, es imposible reducir hasta una cantidad de variables válida para poder introducir el modelo en la SkyPIC. Ahora mismo se necesitan 624 Bytes de memoria RAM de datos, quedando todavía por introducir en el modelo las variables que se encargaran de tratar los aspectos concernientes a la nariz electrónica y en la SkyPIC únicamente se dispone de 368 Bytes en total. Debido a esto, se debe buscar otro microcontrolador que permita realizar este cometido disponiendo de una memoria RAM de mayor tamaño.

5.6.2. Búsqueda de soluciones ante la falta de espacio: Arduino

Como posible solución a este problema, se presenta la ocasión de utilizar una placa Arduino [97] para el cometido que en un principio iba a realizar la SkyPIC.

El Arduino (presentado en el Apartado 3.10) incluye un microcontrolador ATmega328 de la marca de chips Atmel [77]. Las características, en cuanto a memoria, que presenta este microcontrolador son:

- Memoria de programa FLASH: 32 kBytes

- Memoria RAM de datos: 2 kBytes
- Memoria EEPROM de datos: 1 kByte

Como se puede apreciar, la memoria que presenta este dispositivo es mucho mayor que la que presentaba la SkyPIC. Esto se debe a que el microcontrolador ATmega328 es bastante más reciente que el microcontrolador PIC16F876. Con una memoria RAM de 2 kBytes, como tiene esta placa, no hay ningún problema a la hora de almacenar las variables necesarias para la ejecución del CPG dentro del microcontrolador. Además, sobra mucho espacio en el que es posible almacenar las variables relacionadas con la comunicación con la Nariz Artificial. Queda libre el doble de la memoria utilizada por el CPG, memoria más que suficiente para almacenar las variables que se van a utilizar para trabajar con Olus2 (se van a necesitar pocas variables con este fin, muchas menos que con el CPG).

Superado el aspecto de la memoria necesaria para almacenar las variables, queda abordar otros aspectos. En primer lugar, no se sabe si Arduino es capaz de suministrar la corriente necesaria a los servos del robot para que estos puedan realizar movimientos. Observando la tarjeta se aprecia que la entrada de alimentación de la placa es la misma que presentaba la SkyPIC. Tras comprobar las características eléctricas de los dos elementos, vemos que son muy similares, y procedemos a conectar el Arduino con la fuente de alimentación con la que suministrábamos la corriente a la anterior placa. La nueva placa funciona sin ningún tipo de problema con esta alimentación con lo que continuamos teniendo una placa que recibe el suficiente amperaje como para poder manejar a los 8 servo-motores a la vez.

Se procede a conectar los servos al Arduino y se hacen pruebas para verificar que les llega suficiente corriente para poderse mover a la vez. Tras conectar la alimentación positiva de los servos a la salida +5V el resultado es negativo, mostrando los servos un movimiento defectuoso provocado por una alimentación insuficiente. Leyendo la documentación de la plataforma se observa que la alimentación que proporciona el Arduino a través de su salida de +5V es de tan solo 500 mA. Esta corriente es insuficiente para alimentar los 8 servos ya que si todos se encuentran en movimiento pueden llegar a necesitar una alimentación de 1,6 Amperios (introducido en el Apartado 4.1) para funcionar. Al mirar en detalle la documentación se aprecia que hay otro pin en el Arduino, V_{in}, que también tiene como finalidad proporcionar alimentación a otros componentes y en el que no se limita la corriente que sale. V_{in} está directamente conectado a la fuente de alimentación del Arduino, con lo que se podrá disponer de toda la corriente que no necesite la placa. Se hace una segunda prueba utilizando este pin para alimentar a los servos, comprobando como ahora sí que tienen la fuerza necesaria para que se puedan mover en toda su amplitud.

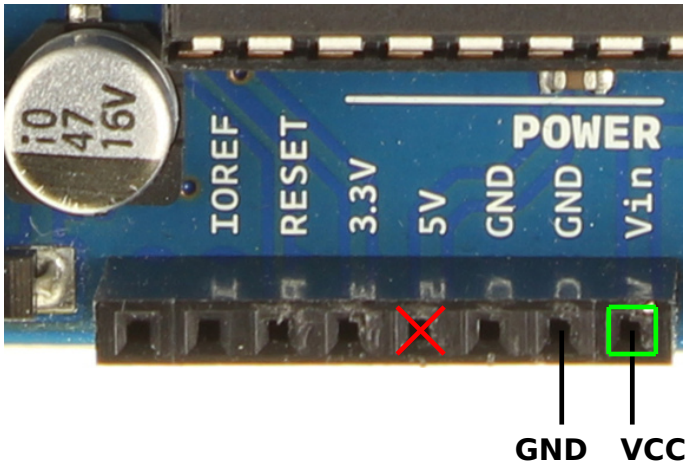


Figura 5.29: Alimentación de los servos

Conexionado Robot-Arduino

Otro problema que se tiene ahora es el conexionado entre el robot y la placa, ya que no se tiene en Arduino el conector del cable como el que tenía la SkyPIC. De momento se están realizando las conexiones entre el robot y la placa utilizando varios cables (Véase Figura 5.31) y montando y desmontando cada vez que se hacen pruebas. Es algo temporal, ya que si se establece este elemento de control como la plataforma final, se construirá un adaptador fiable y que mantenga todo bien unido. Las conexiones realizadas entre los pines de la placa y el robot son:

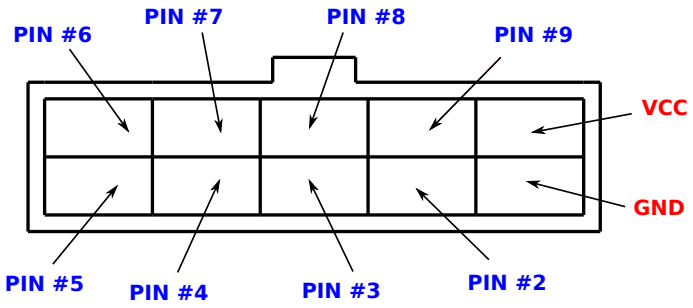


Figura 5.30: Correspondencia de los pines de Arduino con el conector del robot

En el diagrama se puede apreciar cómo no se utilizan los pines 0 y 1 de Arduino, sino que se empiezan a realizar las conexiones a partir del pin 2. Esto se debe a que los dos primeros pines están relacionados con la comunicación serie que incluye el microcontrolador. Esta comunicación serie se puede realizar tanto a través de estos dos pines como del cable USB, pero al activarla se bloquean ambas conexiones. Se va a utilizar la comunicación serie a través del USB para así poder monitorizar, cuando se desee, el estado de la simulación desde una consola en el PC, con lo que se tiene que dejar libre tanto el pin 0 como el pin 1 ya que no se podrá manejar ningún servo a través de ellos.

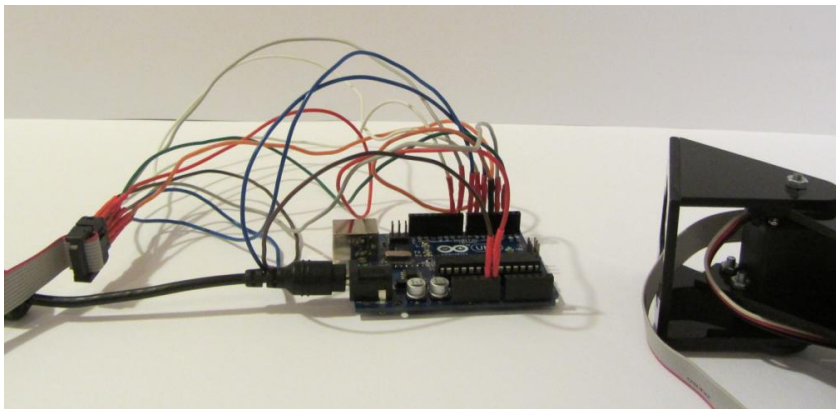


Figura 5.31: Arduino y robot conectados mediante cables independientes

El problema de conectar cada pin con un cable independiente es que hay una alta probabilidad de que alguno de ellos se desconecte y se pierda el control. De momento se continúa utilizando este método pero, si decide utilizarse Arduino debe realizarse un cable unificado ya que si no será imposible realizar pruebas en las que el robot se desplace más de lo que mide el bus de conexión y pueda tirar de la placa, provocando probablemente la desconexión de alguno de los módulos.

Una vez tratados esos aspectos, se comienza a portar el código del CPG, que ya está implementado en el PC, al Arduino. Cabe destacar que este paso no resulta muy difícil ya que el lenguaje, wiring [80], en el que se programa esta plataforma es muy similar al lenguaje

C, que es el que se ha utilizado para realizar hasta ahora las simulaciones en el PC. Se realiza una simplificación del código ya que alguna de las estructuras o funciones que incluía no están soportadas, pero sin realizar muchos cambios se consigue simular el CPG en Arduino. Quizás se podría volver a incluir alguna función o estructura, pero se dejará esa tarea para el final, una vez estén ya fijados todos los parámetros de la simulación.

En el código también se ha incluido alguna librería propia del entorno Arduino, como por ejemplo la librería Servo [98], que permite manejar a los servos haciendo transparente el PWM que hay que ejecutar para poder controlarlos. Esta librería permite manejar hasta 12 servos en una placa como la que se utiliza en este proyecto con lo que no se tendrán problemas para mover los 8 servo-motores que se necesitan. Se incluye un ejemplo de uso de esta librería en el que se hace girar un servo de manera continua:

```
#include <Servo.h>

Servo myservo;  // Objeto servo

int pos = 0;    // Variable que almacena la posición del servo

void setup()
{
  myservo.attach(9);  // Fijamos el pin 9 al objeto servo creado
}

void loop()
{
  for(pos = 0; pos < 180; pos += 1){ // Movemos el servo de 0 a 180 grados
    myservo.write(pos);              // incrementando de grado en grado
    delay(15);
  }
}
```

Cuadro 5.12: Ejemplo de la librería Servo de Arduino

Además, en el programa se ha incluido un módulo de comunicación serie [99] que permite monitorizar el estado de la simulación. Se trata de un módulo temporal en el código, no estará incluido en la versión final del CPG ya que se utilizará para detectar posibles fallos o para entender el comportamiento de la simulación. Permite ver por donde va el flujo del programa, el valor de ciertas variables o incluso obtener gran cantidad de valores con los que realizar gráficas que informen acerca del comportamiento de la simulación en la placa Arduino.

Al terminar de integrar el código en Arduino y probarlo, se hace visible que ahora el tiempo que existe entre cada paso de la simulación es mayor. Esto se debe a que la capacidad de procesamiento del microcontrolador frente a la de la CPU del PC es muchísimo menor. Mientras un PC fácilmente puede tener una velocidad del procesador mayor a 1 Ghz, el Arduino sólo alcanza una velocidad de 16 MHz. Es cierto que la CPU del PC tiene otras tareas que realizar aparte de la simulación del CPG mientras que el Arduino solo se dedica a calcular los pasos de la simulación, pero aún así, la velocidad de procesamiento de la CPU es muchísimo mayor a la que puede ofrecer el microcontrolador que se está utilizando. Debido a esto, al ejecutar la simulación en el Arduino con los mismos parámetros que se utilizaban en el PC, y donde se veía que el robot se desplazaba adecuadamente en tiempo real, ahora puede apreciarse como el robot apenas se mueve ya que cada incremento en la simulación le lleva demasiado tiempo. Debe hacerse que con cada paso en la simulación la variación del sistema sea mayor, es decir, se va a proceder a acelerar la velocidad a la que se ejecuta la simulación. Con esto se conseguirá que el periodo de las señales de cada una de las motoneuronas sea menor.

Para saber el número de pasos que sería adecuado para que la simulación en Arduino fuese a la velocidad necesaria se va, primero, a probar el resultado con una función seno. Se simula

a cada motoneurona como un seno y va probándose con distintos incrementos hasta que se encuentra la velocidad adecuada. Además, en cada paso del bucle de la simulación con el seno, se introduce un pequeño retardo para que así simule el tiempo que emplea Arduino en calcular el valor de las variables de cada una de las neuronas en la simulación del CPG. Para ello se desarrolla el siguiente código, en el que además se puede apreciar como se maneja al conjunto de los 8 servos (este manejo es similar en la simulación que incluye el CPG):

```
void setup()
{
  /*********** Configuramos centro de cada servo ************/
  centro_servo[0] = CENTRO -22;
  centro_servo[1] = CENTRO -30;
  centro_servo[2] = CENTRO -23;
  centro_servo[3] = CENTRO -25;
  centro_servo[4] = CENTRO -29;
  centro_servo[5] = CENTRO -23;
  centro_servo[6] = CENTRO -25;
  centro_servo[7] = CENTRO -29;

  for (i=0; i<NUM; i++){
    pos_servo[i] = centro_servo[i]; // Centramos cada uno
    myservo[i].attach(i+2);         // de los servos al
    myservo[i].write(pos_servo[i]); // principio
  }

  pi = 3,14159265;
  t = 0;
}

void loop()
{
  for (i=0; i<NUM; i++){
    pos_servo[i] = centro_servo[i] + 40*sin(2*pi*t + i*(pi/2));
    myservo[i].write(pos_servo[i]);
  }

  delay(15); // Retardo que simula tiempo calculo variables del CPG
  t = t + 0.001; // Incrementamos el tiempo
}
```

Cuadro 5.13: Locomoción del robot mediante osciladores

Gracias a este código, ajustando el retardo que simula el calculo de las variables del CPG y jugando con el incremento de tiempo que hay entre cada paso, se ha llegado a la conclusión de que para el uso de 16 neuronas en la simulación debe tenerse una longitud del periodo de señal de cada motoneurona de unos 1000 pasos de simulación. Una vez conocido este detalle sólo queda ajustar las constantes de la simulación para que se den esos resultados.

Probando con distintos valores y analizando los resultados, se observa que si aumenta tanto el valor de H como el valor de TAO_M se consigue que la longitud del periodo de la señal de salida de las motoneuronas sea menor. Tras completar numerosas pruebas se obtiene un valor (hay otros valores pero se elige este por criterio personal) para estas dos constantes que permite tener un periodo para las señales de una duración próxima a las 1000 iteraciones, tal y como se quería. El valor de las constantes pasa a ser $H = 0{,}005$ (antes 0.0005) y $TAO\_M = 4$ (antes 0.5).

Una vez ajustadas las constantes toca otra vez buscar las conductancias adecuadas para obtener un desfase entre módulos como el que ya se había encontrado y con el que se había conseguido el desplazamiento del robot en el simulador. Como es una tarea que ya ha sido realizada previamente y ya se tienen unas herramientas preparadas para tal misión, no resulta difícil encontrar unas conductancias válidas para estos nuevos parámetros. Mediante el uso de

mapas paramétricos (como los utilizados en el Apartado 5.5.3) se encuentra rápidamente de manera visual las zonas en las que se encuentran los resultados más cercanos a los valores buscados. Una vez reducida la zona paramétrica se procede a hacer una búsqueda exhaustiva ejecutada automáticamente por el PC, la cual indica que el resultado con un desfase constante más cercano a $\pi/2$ radianes se trata de $g1 = 0{,}052$ y $g2 = -0{,}023$. Por otro lado, como ya se ha comentado anteriormente, si se cambia una conductancia por la otra se conseguirá que el robot se desplace en el otro sentido.

Un problema encontrado es la excesiva tardanza del sistema en encontrar un estado estable entre las neuronas, y es que hasta que no encuentra este estado, el robot no comienza a desplazarse. Para solventarlo, se realiza una simulación con los mismos parámetros en el PC y que se ejecuta con valores iniciales aleatorios en las variables de las distintas neuronas. Una vez alcanzado un estado estable del sistema (hay varios estados estables en un sistema), se procede a almacenar el valor de las variables de las neuronas. Al introducir estos valores como valor inicial en las neuronas del sistema dispuesto en el Arduino, se estará ahorrando el paso inicial hasta llegar a un estado estable. Como pega se puede decir que ahora los parámetros iniciales no serán aleatorios, ya que estarán prefijados para Arduino. Por una parte es cierto, pero por otra no ya que parten de un valor inicial aleatorio calculado en el PC. Para simular este inicio paramétrico aleatorio en el Arduino, se puede crear una matriz con numerosos valores obtenidos en distintas simulaciones en el PC en las que ya se había alcanzado el estado estable.

Una vez expuestos todos estos detalles se puede afirmar que finalmente se ha conseguido simular el CPG dentro de un microcontrolador. Aun así, aún queda pulir algunos detalles como perfeccionar la conexión entre el robot y la placa que lo controla, la posible introducción del microcontrolador en el chasis del robot y también intentar conseguir otro tipo de movimientos con el robot para así realizar desplazamientos en varias direcciones.

5.6.3. Placa controladora definitiva: SkyMEGA

Como último añadido se presenta una placa desarrollada expresamente para el robot por Juan González. Esta placa es la SkyMega [81], placa basada en arduino y que se puede atornillar a la estructura del robot (introducida en el Apartado 3.10). Puede atornillarse tanto a uno de los dos módulos de los extremos como situarse entre dos módulos cualquiera del robot. Debido a la autonomía que puede ofrecer, se decide probar con esta placa para controlar el robot.

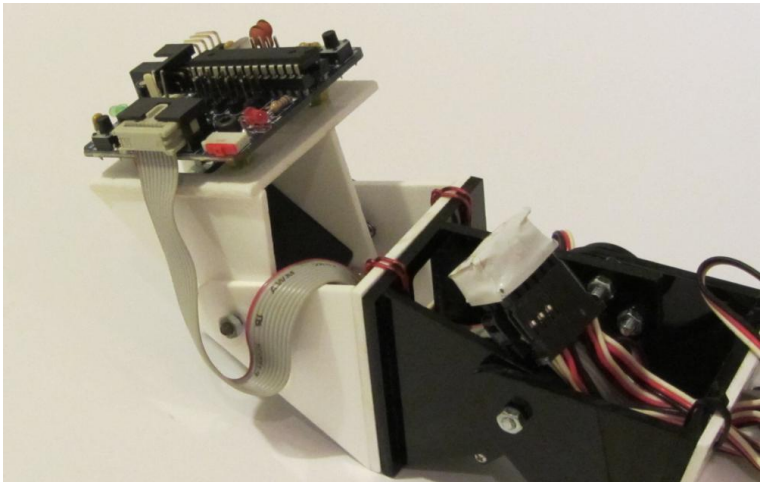


Figura 5.32: Cube Revolutions controlado por SkyMega

Lo primero que se hace es atornillarla al robot (Véase Figura 5.32) y así tener todo lo más integrado posible. Al hacerla pensando en este robot, Juan González incluyó un puerto de entrada como el que tenía la SkyPic, el cual viene perfecto para poder utilizar el cable que lleva

la alimentación y las señales de control a los servos. Se acorta un poco este bus de conexión a los servos ya que ahora, al estar la placa en el robot, se precisa que tenga una longitud mucho menor.

El puerto de expansión que se va a utilizar para conectar el bus es similar al que tenía la SkyPic pero no es exactamente igual. La distribución de pines cambia con lo que se tendrá que hacer una reconfiguración de la distribución de las señales dentro del código fuente. La relación entre los pines del conector del robot y del puerto de expansión de SkyMega es:

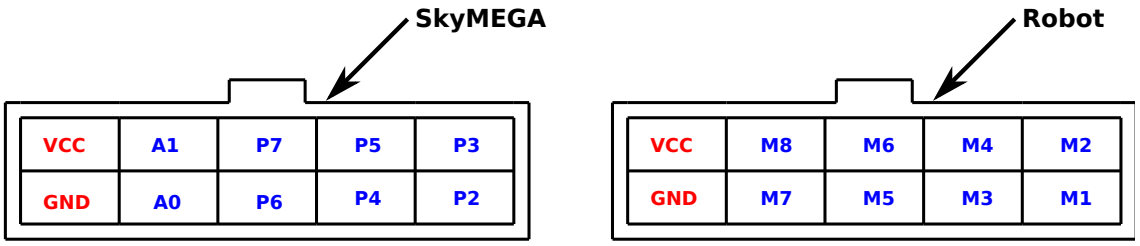


Figura 5.33: Relación pines del conector del robot con el puerto de expansión de SkyMega

La configuración interna de los pines es:

```
myservo[0].attach(2);
myservo[1].attach(3);
myservo[2].attach(4);
myservo[3].attach(5);
myservo[4].attach(6);
myservo[5].attach(7);
myservo[6].attach(A0);
myservo[7].attach(A1);
```

Cuadro 5.14: Configuración de los pines de SkyMega para controlar el robot

Aparte del cambio en los pines de comunicación con los servos, también ocurre que la entrada de la alimentación varía respecto a SkyPIC. Ahora ya no se utiliza un conector jack para alimentar a la placa sino que se utiliza un conector molex de dos entradas. Se fabrica un adaptador de jack a molex (véase Figura 5.34) para poder seguir utilizando la fuente de alimentación que asegura alimentación suficiente con la que poder manejar los servos.

Figura 5.34: Adaptador jack-molex

Ahora que las conexiones entre el robot y la placa son más robustas se comienza a probar el modelo neuronal que permitirá realizar el movimiento comentando en el Apartado 4.1. Este modelo está definido por la combinación de las dos posibles situaciones en las que se pueden disponer los módulos. Se tendrán 6 módulos de la forma cabeceo-cabeceo (movimiento en el plano vertical) y los otros 2 restantes estarán dispuestos en cabeceo-viraje (movimiento en el plano horizontal). La distribución de la disposición de los módulos dentro del CPG es la siguiente:

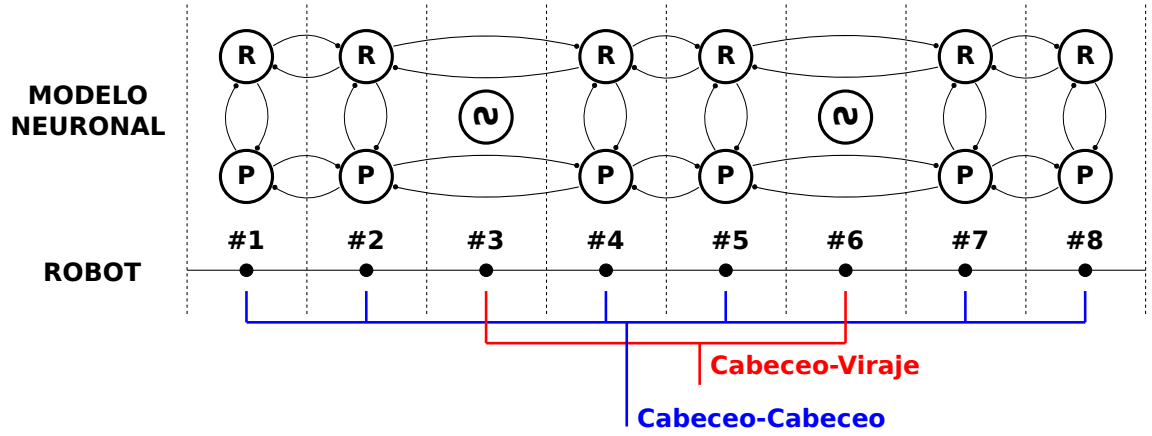


Figura 5.35: Disposición CPG de SkyMega

En la imagen se puede apreciar como los módulos #3 y #6 no están realmente integrados en el CPG. No son parte del CPG, si no que se tratan de un añadido con más parte artificial que el resto de módulos. Esto es así debido a que son los módulos que van a realizar el movimiento en el plano horizontal y, por tanto, ya no tienen que estar sincronizados con el resto de módulos. Estos dos módulos únicamente cambiarán su posición cuando se desee cambiar la dirección del sentido en el que se mueve el robot (introducido en el Apartado 4.1). Si se desea desplazarse en línea recta se debe posicionar a estos dos módulos en su valor central; si quiere hacerse que el robot gire hacia la derecha se debe incrementar a valores positivos el ángulo de los dos módulos; en cambio, si se quiere que gire hacia la izquierda se debe disminuir el valor del ángulo de ambos módulos hasta llegar a ser negativo. Por tanto se tienen tres posibilidades a la hora de realizar el desplazamiento, las cuales quedan representadas en el siguiente diagrama:

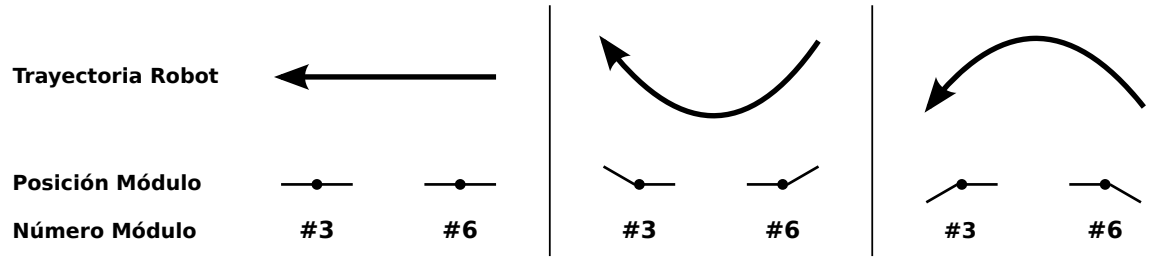


Figura 5.36: Configuración de los módulos #3 y #6 en los 3 modos de movimientos

El ángulo de los módulos #3 y #6 será fijo dependiendo de la situación, pudiendo valer 0, +30 o -30 grados, decidiendo el robot su valor en función de la dirección en la que deba moverse. En cambio, para los otros 6 módulos se continuará utilizando a la motoneurona para establecer su posición. Para poder conseguir un desplazamiento efectivo se necesita que estas 6 motoneuronas estén coordinadas tal y como estaban las 8 motoneuronas del apartado anterior. Para ello va a repetirse la búsqueda de los parámetros óptimos para la consecución de tal fin, que se trata de conseguir otra vez un desfase entre módulos vecinos lo más estable posible y cercano a /2 radianes. Además se debe tener en cuenta, tal y como se hizo con Arduino, la longitud del periodo de señal de la motoneurona para que el robot realice sus movimientos de manera fluida.

Si se representa el mapa paramétrico con 12 neuronas y las mismas constantes que se definieron en el apartado anterior, se observa la siguiente imagen:

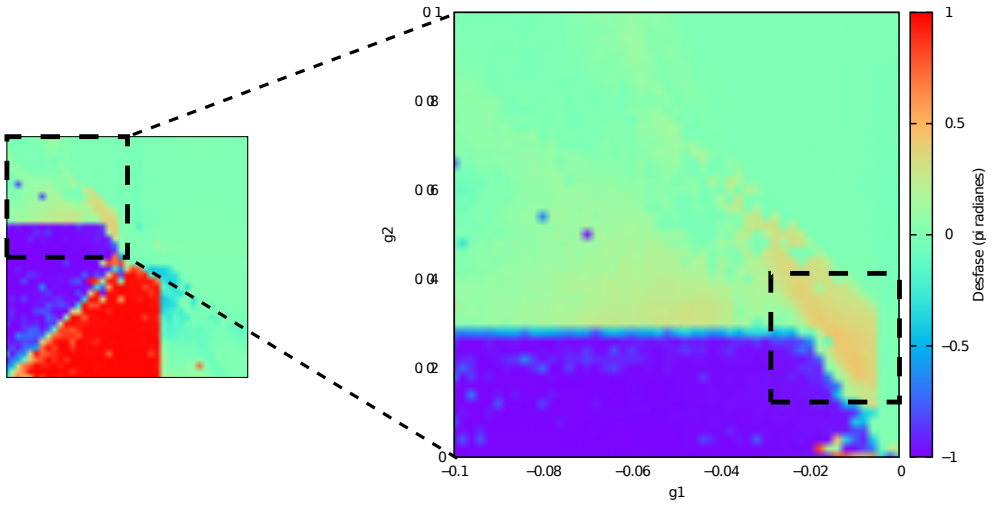


Figura 5.37: Mapa paramétrico de las conductancias, con zoom en la zona de interés

En esa imagen se puede apreciar la situación de los parámetros que se podrían utilizar para hacer desplazarse el robot óptimamente. Debe volverse a analizar la zona paramétrica, esta vez centrándonos únicamente en $-0{,}03 < g1 < 0$ y $0{,}01 < g2 < 0{,}04$, ya que es ahí donde se encuentran los mejores resultados (como puede apreciarse en la Figura 5.37. El resultado es el siguiente:

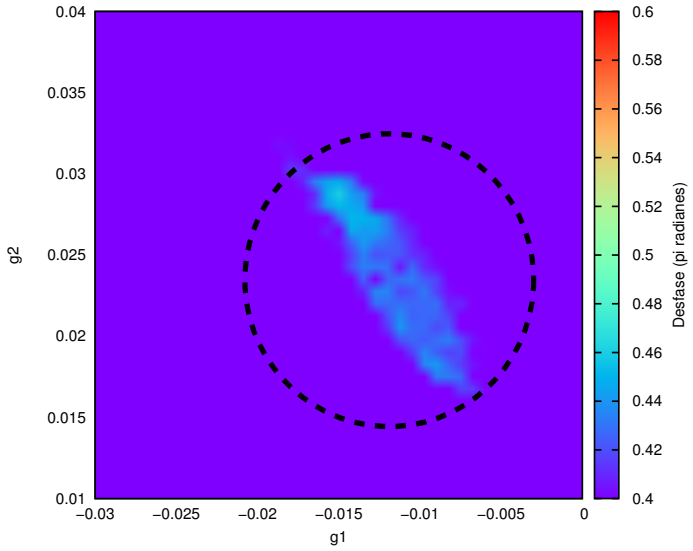


Figura 5.38: Mapa paramétrico de valores válidos filtrando mediante el desfase de los módulos intermedios

El mapa únicamente se centra en los desfases comprendidos entre 0.4 y 0.6 radianes, sacando el resto de resultados posibles del mapa de colores (salen representados en color violeta). Ayudado por esta imagen y analizando el archivo con todos los valores, se obtiene el valor óptimo, el cual ofrece una fase estable y próxima a 0.5 radianes. El valor óptimo que se ha encontrado para los parámetros es $g1 = -0{,}011$ y $g2 = 0{,}024$ (valores escogidos por criterio personal), que ofrecen el desfase entre módulos representado en la Figura 5.39. Se puede apreciar como hay más ruido en estas señales que en las mostradas en páginas anteriores (como por ejemplo la Figura 5.25). Esto se debe al cambio que se ha realizado en el CPG para que se ejecute más rápido (y así pueda introducirse en el Microcontrolador), dando lugar a transicciones más bruscas entre los estados de las neuronas y causándose así de manera más fácil desajustes menores entre los módulos. Este ruido no afecta de manera perceptible el desplazamiento que realiza el robot.

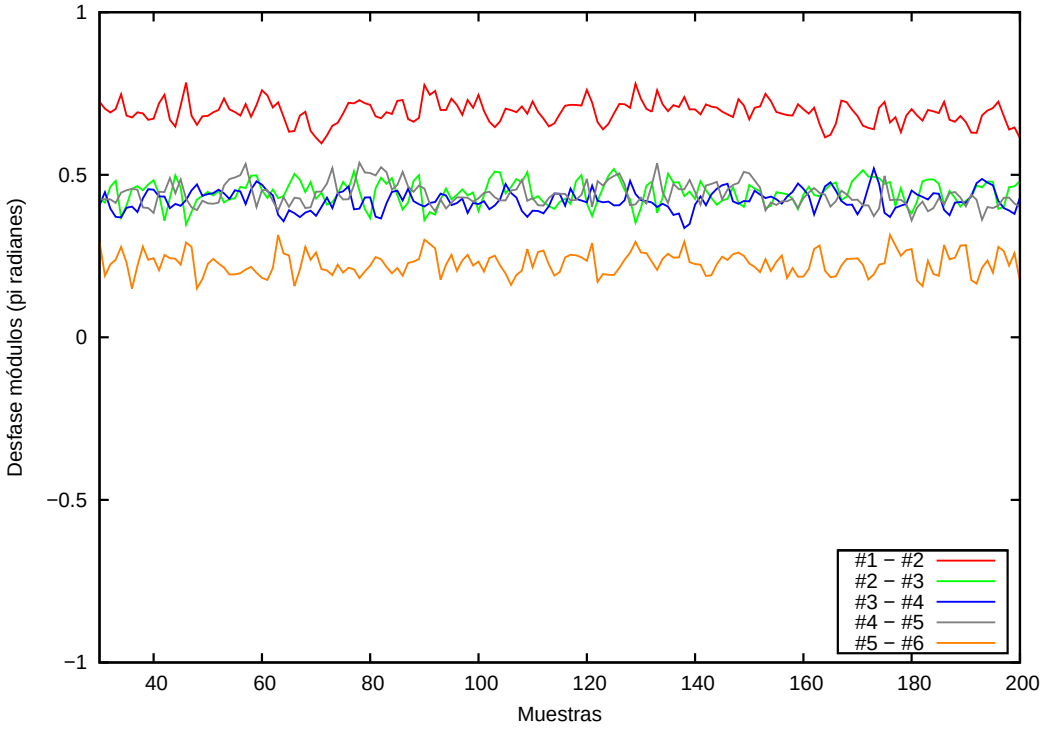


Figura 5.39: Desfase mostrado por los 6 módulos que conforman el CPG a introducir en SkyMega

Tras confirmar en el robot el buen resultado de estas conductancias, se muestra un problema con el conector de la alimentación de SkyMega. Al fabricar el adaptador jack a molex, se utiliza plástico termo-retráctil para proteger los cables y ahora el conector resulta demasiado rígido. Se trata de un problema debido a que al estar unido a la placa y la placa estar atornillada a uno de los extremos del robot, el conector hace palanca al girar el último de los módulos contra el suelo (véase Figura 5.40) y tras unos cuantos movimientos el conector acaba soltándose, con el consecuente apagado repentino del robot.

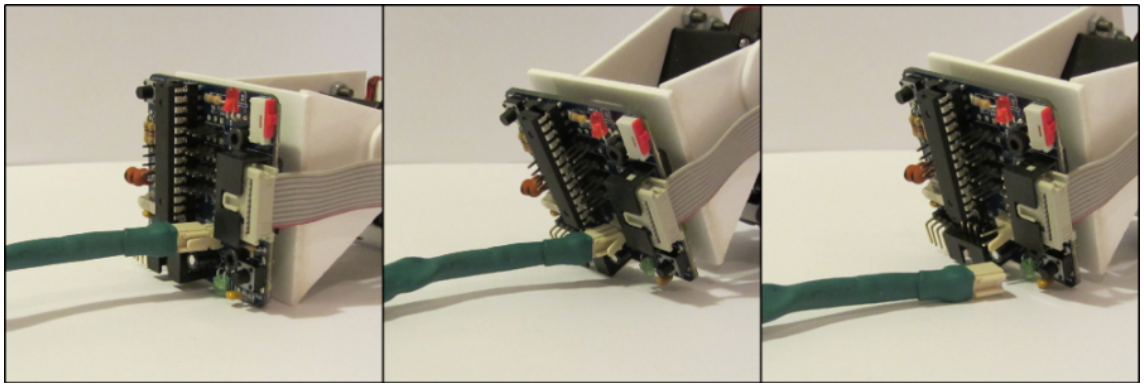


Figura 5.40: Desconexión del conector jack-molex

Para solucionar este asunto se ha hecho una pequeña modificación al robot, incluyendo el adaptador jack a molex en la propia estructura del robot (Véase Figura 5.41). En la parte superior del módulo en el que se sitúa SkyMega se atornilla una pequeña estructura de plástico y se fija en ella al conector jack. Se sueldan los cables pertinentes a la unión del conector molex de la placa con el conector jack hembra de la estructura de madera. Ahora se puede conectar la fuente de alimentación al jack situado en la estructura de madera, el cual al estar situado en la parte superior del módulo no golpea en ningún momento contra el suelo. Mediante esta solución se evita el problema que se estaba produciendo y se continúa manteniendo intacta la

fuente de alimentación por si se quiere volver a utilizarla en algún otro proyecto con SkyPic o Arduino.

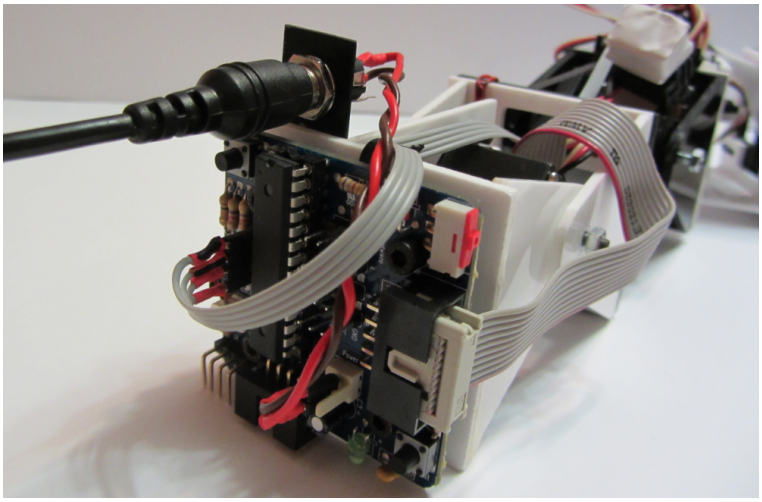


Figura 5.41: Adaptador jack-molex integrado en la estructura del robot

Solucionado este punto ya se tiene un robot totalmente funcional en el que el movimiento se genera mediante la simulación de un CPG de manera autónoma por el microcontrolador que controla el robot. Además, este movimiento se realiza en varias direcciones y, tras haberse integrado a SkyMega en el robot, se pueden realizar los movimientos y recorrer las distancias que se deseen sin miedo a que se produzcan desconexiones en el control de los servos. Como inconveniente ha de señalarse que ahora se ha perdido la simetría del robot, no pudiéndose utilizar el robot cara arriba ya que rompería el adaptador incluido en su estructura. Antes era posible el desplazamiento en cualquier orientación del robot, ahora hay una de las 4 orientaciones posibles que ya no se puede utilizar.

Junto al proyecto se incluye una carpeta que incluye un vídeo en el que se puede apreciar los movimientos que es capaz de ejecutar el robot [100]. En este vídeo se puede apreciar el movimiento en línea recta y en trayectoria circular, ejecutando estas acciones en ambos sentidos del movimiento.

5.7. Comunicación del Robot con la Nariz Electrónica

Tras conseguir que el robot se desplace, se va a trabajar en la integración del mismo con la nariz. Lo primero que se tiene que hacer es conseguir que haya transmisión de información entre Olus2 y SkyMega. Este apartado únicamente se centrará en la comunicación entre ambos dispositivos, dejando para el apartado siguiente la integración de la Nariz Electrónica en el funcionamiento del robot desarrollado.

Para desarrollar este módulo de comunicaciones se parte del código fuente desarrollado por David Yáñez para Olus2 [4], la Nariz Electrónica. Este código incluye todo lo necesario para que la nariz sea capaz de captar señales de odorante presentes en el ambiente en el que se encuentra. Para ello hace uso de la lectura de los sensores, cambia el estado del ventilador y de la resistencia calefactora para adecuar la muestra a medir, hace uso de los LEDs para poder mostrar en el estado en el que se encuentra y utiliza la comunicación serie para transmitir los resultados obtenidos. A partir de esas funciones se desarrollarán las funciones extra necesarias para poder transmitir la información obtenida al Arduino y también para que éste sea capaz de variar ciertos parámetros en la Nariz Electrónica.

5.7.1. Programación de la Nariz Electrónica

Se comienza probando a introducir pequeñas variaciones del código de David Yáñez en la nariz. Para ello, partiendo del código fuente ya modificado, se debe compilar ese código para generar el fichero en hexadecimal (.hex) que será el fichero que se introduzca en el microcontrolador de Olus2. Al utilizar prácticamente la totalidad del código generado por David, y para no tener que modificarlo y así el pueda usar el módulo I2C que se desarrolle, se tendrá que utilizar un compilador compatible con el código generado.

Hasta ahora, todas las herramientas que se han utilizado y que se quieren utilizar en los pasos posteriores del proyecto son herramientas libres y de código abierto. Se trabaja en un sistema operativo libre, se programa, compila y ejecuta las simulaciones por medio de software libre, y se está escribiendo el documento también mediante software libre. En este punto se tiene que utilizar un compilador privativo, de la casa Microchip [76], y además emular las APIs de Microsoft Windows mediante WINE ya que el compilador no está disponible para Linux. Es el único punto negro dentro de un proyecto en el que únicamente se quería utilizar elementos bajo el movimiento de software libre [78]. El compilador en particular que se utiliza es el MCC18 que dentro de los distintos compiladores que existen en la casa Microchip es el que permite compilar código C para el PIC18LF4550.

Para descargar en la Nariz Electrónica el fichero .hex que se ha generado, se utiliza un programador que también es de la casa Microchip. Este programador es el Pickit 2 [101], del que se muestra la distribución de pines a continuación:

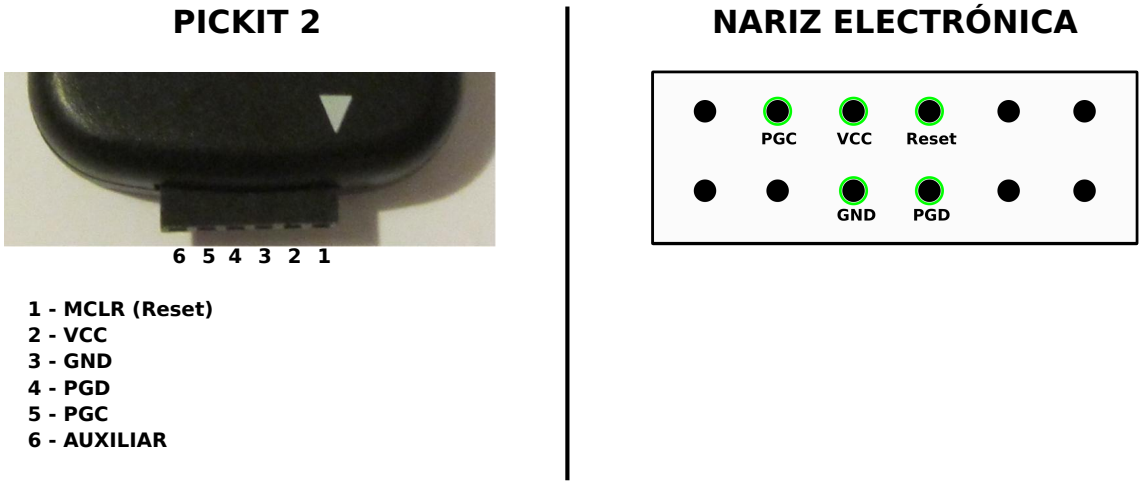


Figura 5.42: Distribución de pines del Pickit2 y de Olus2

En la imagen se puede apreciar también la distribución de los pines que se utilizan en la Nariz Electrónica para comunicarse con el programador. Los pines mostrados son los mismos que se comentaron en el Apartado 4.6.3 al hablar sobre el puerto de comunicaciones. El protocolo utilizado es ICSP que es el protocolo estándar utilizado para programar PICs, así que los nombres de los pines se corresponden con la nomenclatura estándar. Como no se dispone de un conector apropiado, se conecta cada pin a su homólogo en el otro elemento mediante cable independientes.

Utilizando estas herramientas se modifican pequeños detalles en las líneas escritas por David Yáñez para así entender en su totalidad el código y comprender detalles del hardware que conforma a Olus2.Una vez hecho esto, se está preparado para integrar el módulo I2C.

5.7.2. Desarrollo del módulo I2C en la Nariz Electrónica

Hay una librería desarrollada para facilitar la realización de programas que incluyen comunicación I2C para microcontroladores PIC. Se trata de ¡i2c.h¿ y es una librería para el lenguaje de programación C que está incluida en el compilador MCC18. Se comienza a recopilar información sobre sus diversas funciones para poder utilizarla adecuadamente llegando a un punto en el que se descubre que se trata de una librería muy compleja y que habitualmente genera malos resultados debido a su complicado uso. Debido a este hecho se decide no utilizarla, tomándose la idea de hacer un código más cercano a los registros internos del microcontrolador, saltándose las funciones intermedias provenientes de las librerías (se puede ver en el Apéndice B).

Acordado este método de actuación, comenzamos a leernos la documentación del PIC18LF4550 [76], en la que se puede encontrar todos los detalles acerca del funcionamiento interno de las comunicaciones I2C. Lo primero que se observa es que hay un módulo interno, denominado MSSP, encargado de las comunicaciones serie con periféricos u otros microcontroladores. Este módulo puede operar en dos modos, I2C e Interfaz Serie, interesando para este proyecto únicamente el primero. El interfaz I2C del MSSP soporta los modos Máster, Multi-Máster y Esclavo, de los cuales sólo se va a utilizar el último.

La distribución de los pines que se necesita utilizar es (la correspondencia se puede ver en la Figura 4.31):

- PIN RB1: SCL (Línea de reloj)
- PIN RB0: SDA (Línea de datos)
- VCC
- GND

Los pines correspondientes a la línea de reloj y a la línea de datos se tienen que configurar como pines de salida al comenzar el programa. Esto se hace poniendo a 1 los bits correspondientes a estos pines utilizando el método habitual en microcontroladores PIC, habilitando los pines en el puerto al que corresponden, siendo en este caso el puerto B (TRISB):

```
TRISB = 0b00000011;
```

Cuadro 5.15: Configuración de los pines de Olus2

El MSSP, el módulo mencionado, tiene 6 registros con los que puede realizar las operaciones necesarias en la comunicación I2C:

- SSPCON1: Registro de Control MSSP 1
- SSPCON2: Registro de Control MSSP 2
- SSPSTAT: Registro de Estado MSSP
- SSPBUF: Registro del buffer de transmisión/recepción serie
- SSPSR: Registro de desplazamiento MSSP
- SSPADD: Registro de dirección MSSP

El diagrama de flujo que siguen estos registros y sus estados es el siguiente:

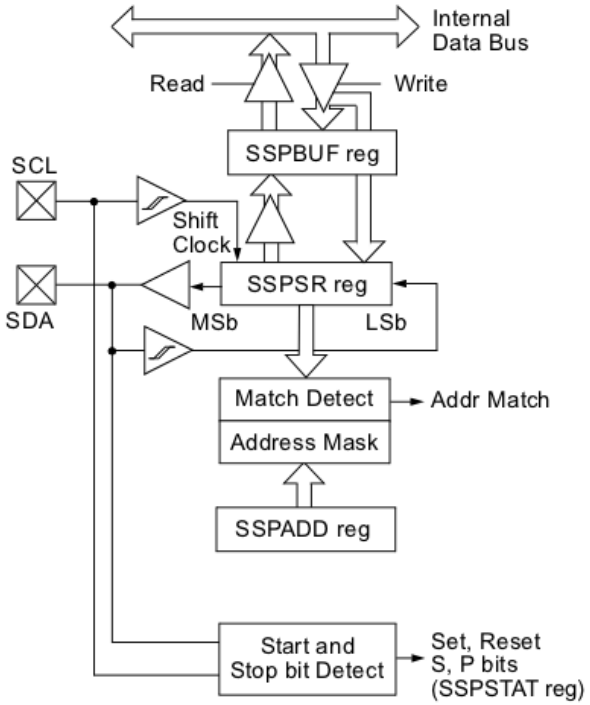


Figura 5.43: Diagrama de funcionamiento del MSSP

Los registros SSPCON1, SSPCON2 y SSPSTAT son los registros de control y estado en la operación I2C. Los registros SSPCON1 y SSPCON2 son de lectura/escritura. Los 6 bits de menor peso del SSPSTAT son de lectura únicamente, siendo los otros dos bits de lectura/escritura.

SSPSR es el registro de desplazamiento utilizado para ir desplazando los datos bien hacia dentro o hacia fuera del módulo. El registro SSPBUF es el buffer donde los datos son escritos o también es de donde se leen, en función de la operación que se esté realizando.

El registro SSPADD almacena la dirección del dispositivo esclavo cuando el MSSP se configura en modo esclavo. Cuando el MSSP se configura en modo maestro, los últimos 7 bits del SSPADD se utilizan para indicar la velocidad que se quiere utilizar en la comunicación.

En operaciones de recepción, los registros SSPSR y SSPBUF son utilizados de manera conjunta para crear un receptor con doble buffer. Cuando el SSPSR recibe un byte completo, se lo transfiere al SSPBUF y la interrupción SSPIF es activada. En cambio, durante operaciones de transmisión, no se utilizará el doble buffer. Al escribir en SSPBUF se estará escribiendo tanto en ese registro como en SSPSR.

Se va a observar en detalle los registros más complejos e interesantes (poniendo únicamente los detalles concernientes al modo esclavo):

SSPSTAT:

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P(1)	S(1)	R/$\overline{W}$(2,3)	UA	BF
bit 7							bit 0

Figura 5.44: Registro SSPSTAT del MSSP

- **SMP:** Bit de control del Slew Rate
  - 1: Deshabilitado
  - 0: Habilitado

- **CKE:** Bit de selección del SMBus
  - 1: Habilitado
  - 0: Deshabilitado
- **D/A:** Bit de selección datos/dirección
  - 1: El último byte recibido/transmitido era de datos
  - 0: El último byte recibido/transmitido era de dirección
- **P:** Bit de parada
  - 1: Una señal de stop fue detectada en la última trama
  - 0: No fue detectada ninguna señal de stop
- **S:** Bit de comienzo
  - 1: Una señal de arranque fue detectada en la última trama
  - 0: No fue detectada ninguna señal de arranque
- **R/W:** Bit de información de lectura/escritura
  - 1: Modo lectura
  - 0: Modo escritura
- **UA:** Bit de actualización de dirección
  - 1: El usuario necesita actualizar la dirección almacenada en el registro SSPADD
  - 0: La dirección no debe ser actualizada
- **BF:** Bit de estado de buffer lleno
  Modo transmisión:
  - 1: Registro SSPBUF lleno
  - 0: Registro SSPBUF vacío

  Modo recepción:
  - 1: Registro SSPBUF lleno (no incluye los bits de ACK y STOP)
  - 0: Registro SSPBUF vacío (no incluye los bits de ACK y STOP)

SSPCON1:

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

Figura 5.45: Registro SSPCON1 del MSSP

- **WCOL:** Bit de colisión en escritura
  Modo transmisión:
  - 1: El registro SSPBUF fue escrito mientras se transmitía la palabra anterior
  - 0: No hay colisión

  Modo recepción: Bit sin uso
- **SSPOV:** Bit de indicador de oferflow en recepción
  Modo transmisión: Bit sin uso
  Modo recepción:
  - 1: Byte recibido mientras el SSPBUF contenía un byte previo
  - 0: No hay overflow
- **SSPEN:** Bit de activación del MSSP

1: Habilita las salidas SDA y SCL

0: Desactiva las salidas SDA y SCL

- **CKP:** Bit de control de liberación del reloj

  1: Libera la línea de reloj

  0: Mantiene la línea de reloj a nivel bajo y asegura la recepción de datos

- **SSMP3-SSMP0:** Bits de selección de modo de MSSP

SSPCON2:

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ADMSK5	ADMSK4	ADMSK3	ADMSK2	ADMSK1	SEN(1)
bit 7							bit 0

Figura 5.46: Registro SSPCON2 del MSSP

- **GCEN:** Bit de habilitación de llamada general

  1: Habilita una interrupción cuando se recibe una llamada a la dirección

  0: Deshabilitado

- **ACKSTAT:** Bit de estado de reconocimiento (Sin uso en modo esclavo)
- **ADMSK5-ADMSK1:** Bits de selección de máscara de dirección del esclavo
- **SEN:** Bit de habilitación extensible

  1: Habilita la transmisión y recepción alternada

  0: Deshabilitado

En la elaboración del módulo I2C se utilizarán la mayoría de los bits que se acaban de ver por lo que es importante comprenderlos. Además, se ha de decir que si que se utiliza la librería ¡i2c.h¿, pero se hace únicamente para abrir la comunicación I2C al encender la Nariz Electrónica. La configuración inicial concerniente al I2C que se realiza es:

```
#define DIR_I2C 0xA0

INTCON = 0b11000000;
INTCON3 = 0b00000000;
PIE1 = 0b00101000;

OpenI2C (SLAVE_7, SLEW_OFF);
SSPADD = DIR_I2C;
SSPCON2bits.SEN = 1;
```

Cuadro 5.16: Configuración de Olus2

Primero se pone en funcionamiento, mediante configuración de PIE1, la interrupción que se genera por recepción de datos a través del MSSP referentes a I2C. Se enciende el módulo I2C, configurándolo en modo esclavo y con el Slew Rate desactivado. Se le asigna al dispositivo la dirección "0xA0" introduciendo este valor en el registro SSPADD y por último se habilita la recepción/transmisión alternada del dispositivo.

Operaciones en modo esclavo

Antes de continuar, se va a ver las operaciones que debe realizar el esclavo para poder tanto recibir como transmitir información con el dispositivo máster del bus I2C. Estos procedimientos ofrecerán una visión más técnica de los aspectos explicados con anterioridad en el Apartado 4.7.

- **Reconocimiento**

  El hardware en el modo esclavo siempre generará una interrupción si ocurre una coincidencia entre la dirección entrante y la dirección almacenada en el SSPADD. Se puede realizar una máscara de direcciones para poder responder a varias direcciones pero esa función no va a ser utilizada en este proyecto.

  Cuando se produce una coincidencia, o bien se recibe una transferencia de datos tras una coincidencia de las direcciones, el hardware se encargará automáticamente de generar un pulso de reconocimiento (ACK) y cargará el SSPBUF con el valor recibido en el registro SSPSR.

  Una vez habilitado el módulo MSSP, este espera hasta que ocurre una condición de Arranque (véase Figura 4.34). Tras la condición de Arranque, los 8 bits recibidos van siendo desplazados al registro SSPSR. Todos los bits entrantes son empleados con el flanco de subida de la línea de reloj (SCL). El valor almacenado en el registro SSPSR es comparado con el valor de SSPADD durante el flanco de bajada del octavo pulso de reloj de la línea de reloj. Si las direcciones coinciden y el bit de overflow no está activado, ocurren los siguientes eventos:

  1. El valor del registro SSPSR se carga en el registro SSPBUF.
  2. Se activa el bit de buffer lleno BF.
  3. Se genera un pulso ACK
  4. El bit de bandera de interrupción del MSSP, SSPIF, es activado en el flanco de bajada del noveno pulso de reloj.

- **Recepción**

  En este estado, el dispositivo esclavo recogerá la información que le transmita el dispositivo maestro. Cuando el bit R/W es 0 (Observar formato de trama en la Figura 4.37) y ocurre una coincidencia en las direcciones, el bit R/W del registro SSPADD también es puesto a 0. La dirección recibida es cargada en el registro SSPBUF y la línea SDA se mantiene a nivel bajo (ACK). Cuando el byte de condición de overflow esta a 1, el pulso de reconocimiento se envía. Existe una condición de overflow si tanto los bits BF o SSPOV están activados.

  Una interrupción MSSP es generada por cada byte de datos transferido. El bit de flag de interrupción, SSPIF, debe ser borrado en el software. Se utiliza el byte D/A, del registro SSPSTAT, para determinar si el byte recibido se trata de un dato o de una dirección.

  Si SEN está activado, la línea de reloj se mantendrá a nivel bajo a continuación de cada transferencia de datos. El reloj deberá ser liberado activando el bit CKP una vez el esclavo esté listo para recibir más datos.

- **Transmisión**

  En este estado, el dispositivo esclavo enviará información por el bus I2C la información que le requiera el dispositivo maestro. Cuando el bit R/W de la dirección entrante es 1 y ocurre una coincidencia entre las direcciones, el bit R/W del registro SSPSTAT es activado y la dirección recibida es cargada en el registro SSPBUF. Se envía el pulso ACK en el noveno pulso de reloj y a continuación la línea SCL se mantiene a nivel bajo. Mediante el bloqueo del reloj, el dispositivo maestro esta imposibilitado de enviar otro pulso de reloj hasta que el esclavo esté preparado para transmitir datos. Los datos a transmitir deben ser cargados en el registro SSPBUF el cual carga a su vez el SSPSR. Entonces, se activa la señal de reloj SCL al poner el bit CKP a 1. Los ocho bits de datos son desplazados hacia fuera en los flancos de bajada de la entrada SCL, lo cual asegura que la señal SDA sea válida durante los periodos de estado alto en la señal de SCL.

  El pulso de recepción ACK del dispositivo máster es introducido en el flanco de subida del noveno pulso de entrada SCL. Si la línea SDA se encuentra en estado alto (no ACK) significa que la transferencia de datos ha terminado. En este caso, la lógica del dispositivo esclavo es reseteada (resetea SSPSTAT) y comienza a monitorizar en búsqueda de otra

secuencia de Arranque. Si la línea SDA mostró un estado bajo (ACK), los siguientes datos a transmitir deben ser cargados en el registro SSPBUF. Otra vez, el pin SCL debe ser activado poniendo el bit CKP a 1.

Una interrupción MSSP es generada por cada byte de datos transferido. El bit SSPIF debe ser limpiado por software y el registro SSPADD se utiliza para determinar el estado del byte. El bit SSPIF es activado en el flanco de bajada del noveno pulso de reloj.

Con la documentación recopilada, se ha realizado el bloque de código que se encarga del control de la interrupción generada por el módulo MSSP. El código realizado, situado en la función en la que se tratan las interrupciones del microcontrolador, es el siguiente:

```
//Interrrupcion I2C
if (PIR1bits.SSPIF == 1){

        if (SSPSTATbits.S == 1){

                //Estado 1: El maestro quiere que el esclavo transmita
                if (SSPSTATbits.D_A == 0 && SSPSTATbits.R_W == 1){
                        SSPCON1bits.CKP = 0;
                        addr_i2c = SSPBUF;
                        trama_i2c_tx();
                        reg_index = 0;
                        SSPBUF =  i2c_reg_tx[reg_index];
                        SSPCON1bits.CKP = 1;
                }

                //Estado 2: El esclavo continua transmitiendo al maestro
                if (SSPSTATbits.D_A == 1 && SSPSTATbits.R_W == 1){
                        SSPCON1bits.CKP = 0;
                        reg_index++;
                        if (reg_index >= MAX_SIZE_I2C_TX)
                                reg_index = 0;
                        SSPBUF = i2c_reg_tx[reg_index];
                        SSPCON1bits.CKP = 1;
                }

                //Estado 3: El maestro quiere enviar datos al esclavo
                if (SSPSTATbits.D_A == 0 && SSPSTATbits.R_W == 0){
                        SSPCON1bits.CKP = 0;
                        addr_i2c = SSPBUF;
                        reg_index = 0;
                        SSPCON1bits.CKP = 1;
                }

                //Estado 4: Recepción de datos
                if (SSPSTATbits.D_A == 1 && SSPSTATbits.R_W == 0){
                        SSPCON1bits.CKP = 0;
                        i2c_reg_rx[reg_index] = SSPBUF;
                        reg_index++;
                        if (reg_index >= MAX_SIZE_I2C_RX){
                                trama_i2c_rx();
                                reg_index = 0;
                        }
                        SSPCON1bits.CKP = 1;
                }
        }

        PIR1bits.SSPIF = 0;
}
```

En esta sección del código se puede apreciar las acciones que se realizan en el microcontrolador al recibir una petición por parte del dispositivo maestro. Al hacer una petición de lectura o escritura, la ejecución entra dentro de la función gestionada por la interrupción.

Allí, tras comprobar que se ha ejecutado previamente una secuencia de Arranque, extrae los bits pertinentes de la trama de solicitud y actúa en consecuencia al valor de estos bits. Existen cuatro bloques que corresponden a las siguientes acciones:

1. **D/A = 0** y **R/W = 1**: El maestro quiere que el esclavo transmita datos. Primero retira el valor contenido en el registro SSPBUF y así el bit BF se pone a 0 automáticamente. Se construye la trama que va a enviar al maestro (mediante la función trama_i2_tx();) y se inicia la transferencia de la misma.

2. **D/A = 1** y **R/W = 1**: El esclavo sigue transmitiendo datos al maestro. Sigue enviando la trama construida en el bloque anterior. Continúa entrando en este bloque hasta que termina de enviar la trama al maestro.

3. **D/A = 0** y **R/W = 0**: El maestro quiere iniciar transmisión de datos al esclavo. Vacía el contenido almacenado en el registro SSPBUF, poniéndose así el bit BF a 0 para poder comenzar a recibir los datos. Inicializa el contador que se encarga de colocar adecuadamente en el registro i2c_reg_rx[] la trama que se va a recibir.

4. **D/A = 1** y **R/W = 0**: El maestro transmite datos al esclavo. Vuelca el byte recibido en la posición del registro indicada por el contador e incrementa el contador. Si se ha llegado a la última posición del registro, se inicializa el contador y se vuelcan los datos contenidos en la trama a sus respectivas variables.

Como se puede apreciar en el código, la línea de reloj se bloquea mientras se realizan las actuaciones pertinentes y una vez terminan se vuelve a liberar. Esto se hace desactivando durante el proceso el bit CKP. Tras ejecutar cualquiera de esas acciones, y antes de salir de la función, se desactiva el bit de flag de la interrupción para que así pueda volver a activarse en caso de que el maestro vuelva a querer comunicarse.

Combinando toda la información anterior de manera adecuada se ha sido capaz de desarrollar el módulo I2C que se necesita. Tras completar este desarrollo ya se tiene tenemos la posibilidad de transmitir información entre la Nariz Electrónica y el Arduino, actividad que se va a realizar en el siguiente apartado.

5.7.3. Comunicación entre el Arduino y la Nariz Electrónica

Este apartado se realiza con Arduino, y no con SkyMEGA, debido a que era el único de los dos microcontroladores posibles que se tenía en el momento de realizarlo. Sin embargo esto no tiene relevancia ya que las únicas diferencias que presentan las dos placas son la disposición externa de los pines y la forma del PCB, características sin importancia y que no repercuten en el código que vamos a desarrollar para comunicar los dos dispositivos. Se va a proceder a integrar en Arduino todas las funciones de la comunicación que van a ser necesarias cuando se integre la Nariz con el Robot. Esto se realiza así para simplificar el análisis del comportamiento de la comunicación abstrayendo la parte de código perteneciente al Robot.

Para hacer uso de las comunicaciones I2C no se van a tener tantas dificultades como en el PIC (Nariz Electrónica), ya que para la plataforma Arduino si que existe una librería que se encarga de todas las gestiones necesarias [102]. Esta librería se denomina ¡Wire.h¿ y contiene las funciones que permiten establecer la comunicación, recibir y enviar información, y cerrar la transmisión.

Tras varios fracasos al intentar establecer la comunicación se hace evidente que PIC y Arduino tratan las direcciones I2C de forma distinta (véase Figura 5.47). Esto se hace evidente mediante el análisis de las tramas enviadas por cada uno de los dispositivos, conectando su salida SDA a un osciloscopio y observando cada uno de los bits que salen. Resultó ser una tarea difícil pero útil para descubrir el fallo realizado. La dirección I2C tiene una longitud de 7 bits, pero en

PIC se tiene que definir esta dirección incluyendo el bit R/W dando lugar a una combinación de ambos datos. Esta combinación de la dirección junto al bit de Lectura/Escritura tiene su razón de ser debido al formato que presenta la trama de inicio de comunicación en I2C (se puede observar en la Figura 4.37). En cambio, en Arduino no se deben combinar ambos datos si no que se introduce únicamente la dirección del dispositivo esclavo y el propio microcontrolador será quién se encargue de unir estos dos datos para completar la trama.

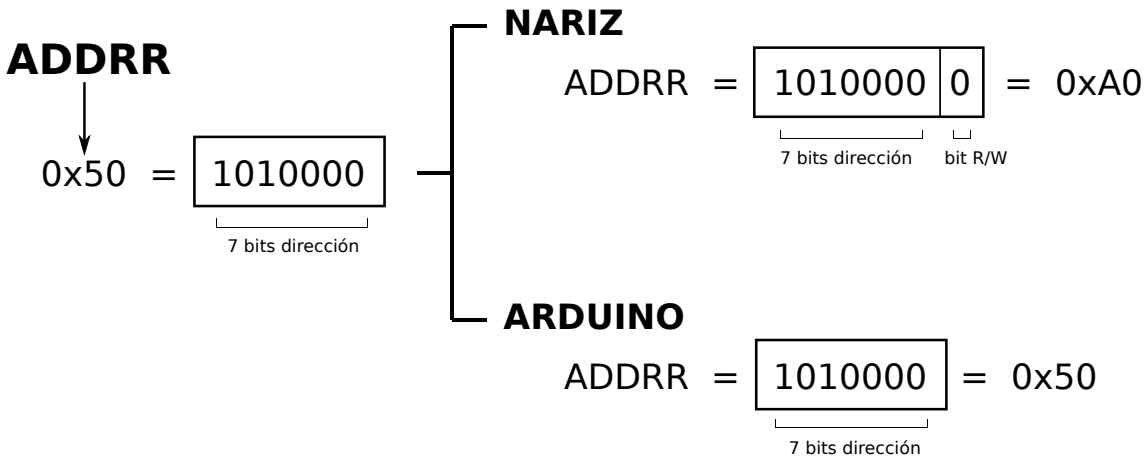


Figura 5.47: Tratamiento de direcciones I2C por parte de Olus2 y Arduino

Se ha utilizado al programar el PIC la dirección 0xA0 para definir la dirección del esclavo, es decir, la perteneciente a la Nariz Artificial. Por tanto, para poder establecer la comunicación entre las dos plataformas, se debe indicar a Arduino que el esclavo tiene la dirección 0x50 y el microcontrolador ya se encargará de gestionarlo.

```
SLAVE_ADDRESS = 0x50
```

Cuadro 5.17: Dirección de esclavo I2C de la Nariz Electrónica

Resuelto el asunto de la dirección de los dispositivos, queda por establecer el formato de la trama de información que se va a comunicar a través del I2C. De todos los datos que pueda contener la Nariz Electrónica, se quiere conocer desde Arduino el valor de cuatro de estas variables (el nombre que se le da a las variables viene del código implementado por David Yáñez):

- **OLORSENS:** Valor obtenido por el sensor químico TGS2600.
- **TEMSENS:** Valor obtenido por el sensor de temperatura.
- **CAUDAL:** Valor actual que presenta el ventilador controlado por Olus2.
- **TEMCAL:** Valor actual que presenta la temperatura de calefacción controlada por Olus2.

Estas variables están definidas por el tipo de dato UNSIGNED INT y su valor en hexadecimal está comprendido entre 000 y 3FF (0 y 1023 en decimal). La transmisión I2C se realiza byte a byte, por lo que se utilizará 3 bytes por cada uno de los datos que se deba enviar. Realmente, al ser su máximo valor 3FF, solo se necesitan 10 bits (3 bytes son 12 bits) ya que el mayor byte como mucho almacenará "0011". Para enviar una trama con las cuatro variables, se podrá combinarlas utilizando estos bits vacíos y así reducir el tamaño de la trama de 12 a 10 bytes. Esta idea es interesante, pero no se va a realizar esta combinación ya que el ahorro en la transmisión de los bytes se traduciría en un mayor número de operaciones tanto en el esclavo como en el maestro por la necesidad de decodificar la trama bit a bit.

La trama de transmisión en el sentido Arduino-Olus2 no tiene el mismo tamaño que la trama que va en el sentido Olus2-Arduino. La trama Arduino-Olus2 tiene una dimensión de 6

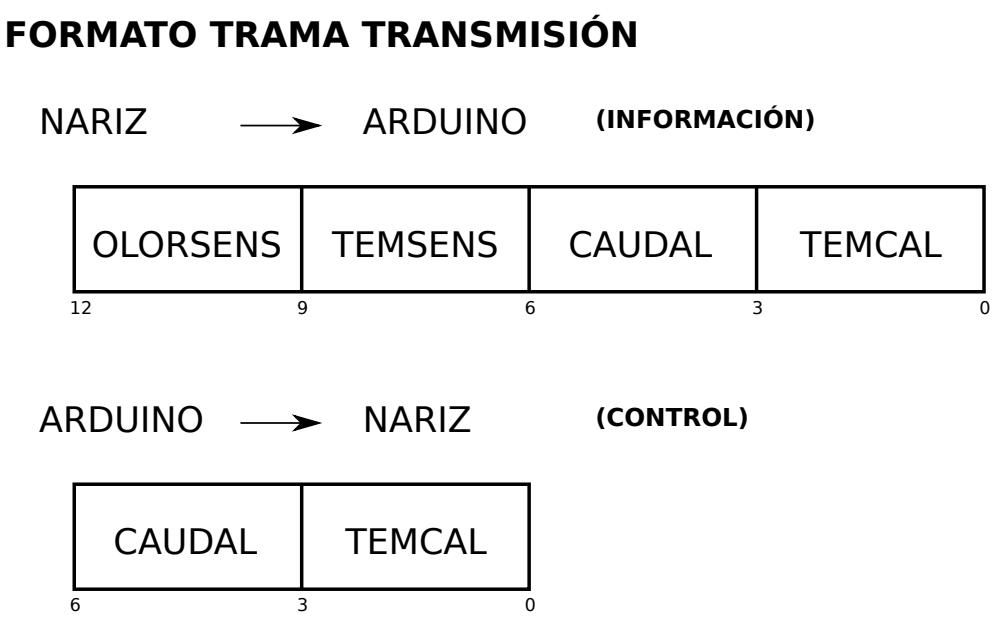


Figura 5.48: Formato de las tramas de Transmisión y Recepción creadas para la comunicación I2C (a la izquierda el bit más significativo)

bytes, debido a que contiene el valor a fijar en las variables CAUDAL y TEMCAL, es decir, la trama tiene como misión controlar estos dos parámetros en la Nariz Electrónica. Por el otro lado, la trama Olus2-Arduino, esta compuesta por 12 bytes ya que contiene a las 4 variables, OLORSENS, TEMSENS, CAUDAL y TEMCAL. Esta trama se utiliza para informar al robot tanto de los valores que está obteniendo la Nariz Electrónica mediante sus sensores como de los valores que presenta en los dispositivos que le permiten mejorar la medida obtenida. El diagrama de la Figura 5.48 muestra el diseño de las tramas. Se trata de una comunicación asimétrica, pero no significa que la comunicación no esté utilizada la mitad del tiempo en el sentido Arduino a Nariz ya que la transacción se realiza únicamente con 6 bits y no 12 bits como en el otro sentido, estando dimensionadas de manera diferente la transacciones en cada sentido.

A la hora de poder enviar y recibir estos datos, se han realizado unas funciones que se encargan de gestionar las tramas para cargar o descargar los datos desde las variables a ellas. Estas funciones se encargan de traducir los valores números en UNSIGNED INT a caracteres ASCII en UNSIGNED CHAR para así poder introducirlos en el array de bytes que se envía a través del bus I2C. Se ha tenido que implementar estas funciones tanto para Olus2 como para el Robot (los códigos están en el Anexo B).

Para observar si estas funciones se ejecutan de la manera deseada se tiene que analizar el funcionamiento combinado de Arduino y la Nariz Electrónica. Se depura el código haciendo uso del terminal serie que incluye Arduino y los leds que incluye Olus2. Se manda información desde Arduino a Olus2 para cambiar los valores de sus variables, y Olus2 modifica el caudal del ventilador o la temperatura de calefacción, y a su vez devuelve el nuevo valor de estas variables (incluyendo también a las últimas mediciones de los sensores) a Arduino.

Además de observar los resultados por la consola serie, se hace uso del programa Arduinoscope [103], herramienta que transforma a Arduino en un analizador lógico y que se encarga de dibujar las señales que se generan en los pines o bien variables internas del microcontrolador (véase Figura 5.49). Gracias a esta herramienta se puede apreciar como oscilan las medidas de los sensores si se aproxima o aleja algún objeto que desprenda mucho olor, o bien como varían estas medidas al cambiar el valor del caudal del ventilador o de la temperatura de calefacción (cada una de las tres gráficas pertenece a una variable y va dibujando la forma de la onda según se van capturando los valores). Tras varias pruebas, y ciertos arreglos en el código, se obtiene una comunicación estable y funcional entre los dos elementos.

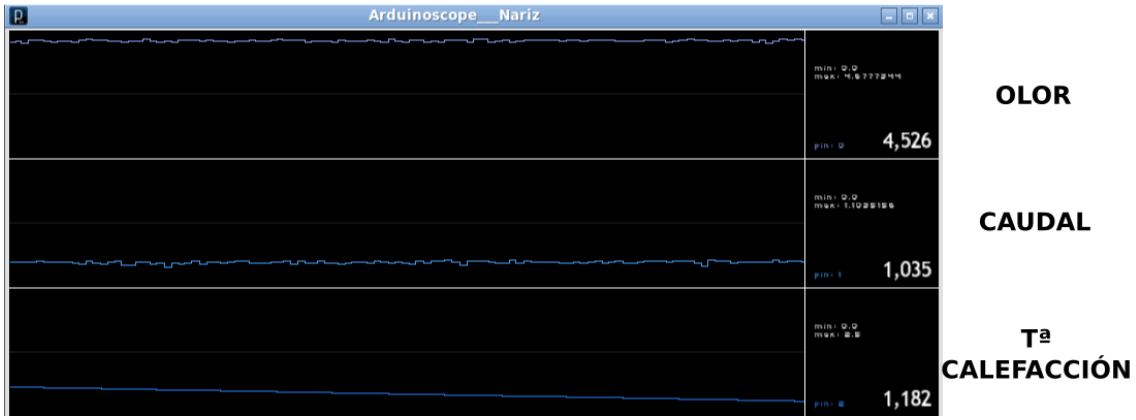


Figura 5.49: Captura de Arduinoscope

Ahora que ya se tiene perfectamente establecida la comunicación I2C entre las dos plataformas, y habiéndose también definido el formato de las tramas, únicamente queda incluir esta funcionalidad en el robot para que así se puedan utilizar los valores recibidos a la hora de decidir la dirección en la que se debe mover.

5.8. Integración de todos los elementos

Una vez se tienen todos los elementos funcionando de manera independiente se va a proceder a integrarlos en un único sistema, donde el funcionamiento de cada uno de ellos dependerá del resto de elementos.

Para poder realizar pruebas uniendo la locomoción del robot con la recepción del sensor de odorante, los dos elementos deben estar acoplados de tal manera que la comunicación entre ellos no se corte por los movimientos del robot. Con el fin de salvaguardar la comunicación y hacer el robot lo más compacto posible, se procede a fijar la Nariz Electrónica al primer módulo de Cube Revolutions. Se pega una extensión de plástico al punto de unión de los módulos #1 y #2, y se une esta prolongación con Olus2 mediante cinta adhesiva de doble cara (permitiendo la separación del sensor al robot de manera sencilla si se requiere para otra aplicación). El resultado se puede apreciar en la siguiente imagen:

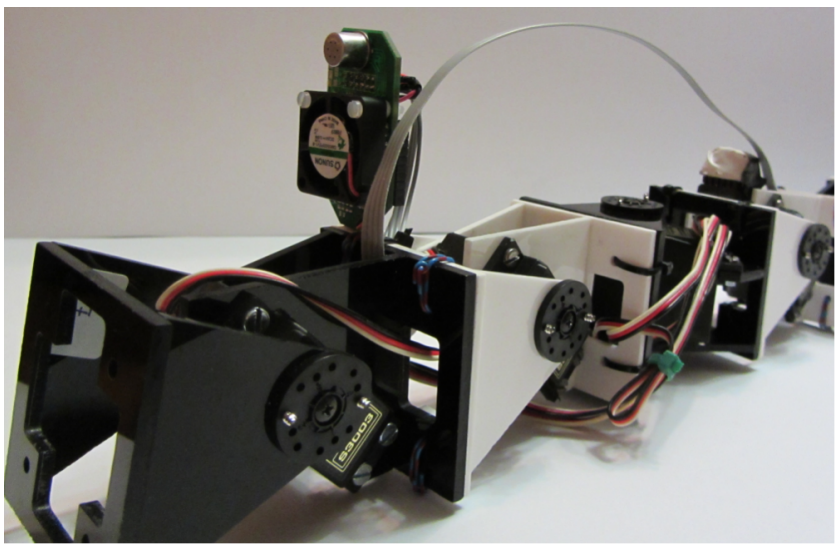


Figura 5.50: Olus2 integrada en la estructura del robot

En la imagen también se puede apreciar el bus utilizado para establecer la comunicación entre Olus2 y SkyMega. Este bus se compone de cuatro cables: Alimentación, masa, datos I2C (SDA) y reloj I2C (SCL). La extensión del bus se ha dimensionado de tal manera que soporte las diferentes posturas que puede ofrecer el robot, experimentando su máxima tensión cuando el robot se enrolla sobre si mismo. Para que no se produzca ninguna desconexión debida a los posibles tirones del cable en los movimientos del robot, se han protegido las conexiones de los dos dispositivos fijando el bus mediante bridas a la estructura del robot. De esta manera no podrán producirse tirones en los extremos del bus sino únicamente en la parte central del mismo.

Una vez se ha establecido la conexión física entre los dos elementos, y tras implementarse en el apartado anterior la comunicación I2C entre ellos, se va a proceder a desarrollar un nuevo módulo en SkyMega que se encargue de, mientras se simula el CPG, gestionar la comunicación I2C con Olus2. Este módulo se debe hacer cargo tanto de la transmisión/recepción de las tramas como del algoritmo que se utilizará para tratar de encontrar la fuente de odorante.

La parte encargada de gestionar la recepción/transmisión de tramas se trata del módulo desarrollado en el apartado anterior (Apartado 5.7.3). Por otro lado, la parte encargada de tratar los datos recibidos de la comunicación y utilizarlos para realizar la búsqueda del odorante se basa en el algoritmo que se va a explicar a continuación.

5.8.1. Establecimiento del patrón de búsqueda

Al iniciarse la ejecución de SkyMega, el microcontrolador establece la comunicación I2C erigiéndose como el Maestro de la misma. Tras iniciar la comunicación, le transmite a la Nariz Electrónica los valores con lo que debe configurar sus actuadores (calefactor y ventilador). El arranque y configuración se realiza mediante las siguientes líneas de código:

```
Wire.begin();

for (i=0; i<NUM_BYTES_RX; i++)
  trama_rx[i] = ' ';

delay(500);
Wire.beginTransmission(DIRECCION_ESCLAVO);
Wire.write("0003FF");
Wire.endTransmission();

cont_algoritmo = 0;
olor_m_actual = 0;
olor_m_previo = 0;
```

Cuadro 5.18: Configuración de la comunicación I2C

Como se puede apreciar en el código, se configura a Olus2 mediante la trama *"0003FF"*. Esto significa, si se observa el formato de trama (Figura 5.48), que se está fijando el valor del caudal a *"000"* (0 en decimal) y el de la temperatura de calefacción a *"3FF"* (1024 en decimal). Es decir, el caudal se apaga, ya que su función es extraer las partículas almacenadas en la caja de Olus2 y en este proyecto la Nariz Electrónica no está contenida en su caja protectora. Por otro lado, la temperatura de calefacción se fija al máximo para maximizar la sensibilidad del sensor de odorante ante las partículas del medio.

Una vez configurado, cada 100 pasos de la simulación se irá reclamando a Olus2 que entregue información de como se encuentra el valor de sus registros. De los cuatro registros que envía (Figura 5.48), SkyMega sólo utilizará el valor de uno de ellos para realizar sus cálculos: OLORSENS (las variables se muestran en la sección 5.7.3). El robot irá acumulando el valor de varias medidas, con la misma distancia temporal entre cada una de las muestras, para que al llegar a tener 10 muestras obtenga el valor medio medido en ese periodo de tiempo. Obtenido

```
i++;
if (i == 100){
  //Ponemos en practica el algoritmo
  olor_m_actual = olor_m_actual + leeI2C();

  if (cont_algoritmo==NUM_VALORES_ALGORITMO-1){
    olor_m_actual = olor_m_actual/10;

  if (olor_m_actual <= (olor_m_previo*97/100))
    cambioSentido();

    cont_algoritmo = 0;
    olor_m_previo = olor_m_actual;
    olor_m_actual = 0;
  }
  else
    cont_algoritmo++;

  i = 0;
}
```

Cuadro 5.19: Algoritmo de búsqueda por umbral

el valor medio actual, lo comparará con el valor medio que se obtuvo anteriormente (valor medio previo) y decidirá que debe hacer en función del resultado de la comparación. Si el nuevo valor es menor que el 97 % del valor de la anterior medida, entonces cambiará el sentido de movimiento del Robot. En cambio, si el nuevo valor se encuentra por encima de este 97 % el Robot continuará desplazándose en el sentido en el que lo hacía en ese instante. El código de este algoritmo se muestra en el Cuadro 5.8.1, aunque se puede ver integrado con el código del programa completo en el Anexo B.

En el algoritmo únicamente se va a utilizar a la variable OLORSENS debido a que los otros parámetros no afectan a los cálculos. La temperatura medida, TEMSENS, no es utilizada para tomar decisiones aunque en futuras versiones del algoritmo de búsqueda si que podría utilizarse. Por otro lado el valor del caudal, CAUDAL, y de la temperatura de calefacción, TEMCAL, se fija al comienzo del programa (se puede ver en la sección de código mostrada en el Cuadro 5.8.1) y no se vuelve a cambiar en ningún momento de la ejecución del algoritmo. En futuras versiones sería interesante hacer que variasen en ciertas situaciones, como por ejemplo para eliminar partículas de las proximidades estén que saturando el sensor (aumentando el valor de CAUDAL), o bien aumentando o disminuyendo la sensibilidad del sensor (cambiando el valor de TEMCAL).

Tras tomar esta decisión, SkyMega volverá a comenzar el almacenaje de muestras de la Nariz Electrónica con las que realizar el cálculo explicado.

Para el cambio de sentido se ha desarrollado un módulo que se encarga de gestionar el intercambio de las conductancias en las sinapsis del modelo (explicado el cambio de sentido en el Apartado 5.5.4). Este módulo también se encarga de proporcionar los valores iniciales con los que debe comenzar la simulación, que se obtienen de una simulación ejecutada previamente (con valores iniciales aleatorios) en el PC y así no se debe esperar hasta que el modelo se encuentre en un estado estable (como ya fue comentado en el Apartado 5.6.2).

Con la ayuda de este sencillo algoritmo el Robot se situará en las cercanías de un punto de alta concentración de odorante, el cual puede tratarse de la fuente en la que se origina la presencia del odorante. Aún así, el Robot no se parará sino que se encontrará oscilando entre un sentido de movimiento y el otro, alejándose y acercándose continuamente del punto en el que siente una mayor presencia de odorante. Aunque el Robot se quede oscilando en torno al punto en que se encuentra la fuente de odorante, se habrá aproximado a ese punto respecto a la posición en la que se encontraba en el instante inicial de la búsqueda. Esto se puede apreciar

en la Figura 5.51.

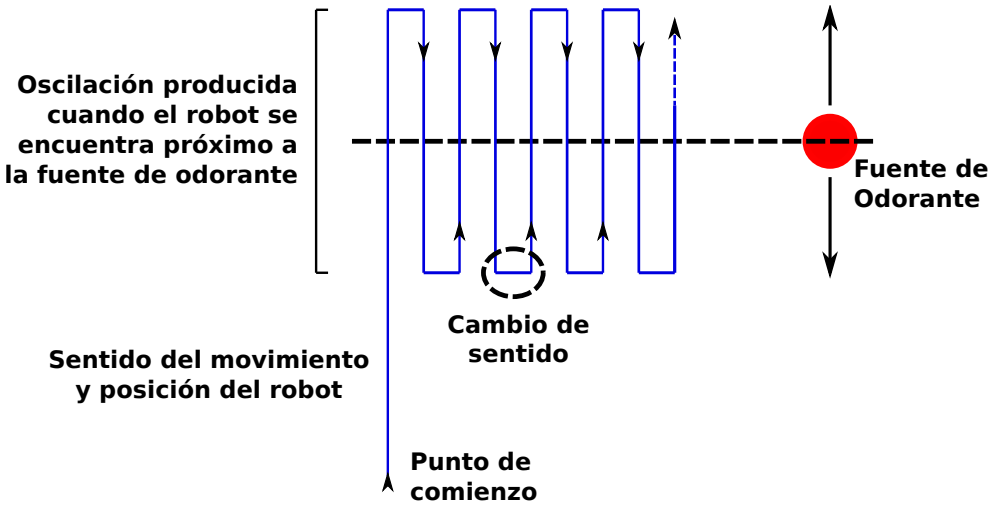


Figura 5.51: Actuación del robot al aplicar el algoritmo de búsqueda

Al probar este algoritmo en una situación real con el robot se observa que no se tuvo en cuenta un factor importante. Cuando el robot detecta que se está alejando de la fuente de odorante, al disminuir el valor de la variable OLORSENS, cambia el sentido de su desplazamiento. En el caso de que el decremento de la variable se alargue en el tiempo, cuando realiza la siguiente medida y al comprobar que sigue disminuyendo, volverá a cambiar el sentido del viaje. El robot continuará tomando esta decisión hasta que el valor medido pare de disminuir y se estabilice. Este comportamiento erróneo no se había tenido en cuenta, por lo que el algoritmo no es válido y se debe procurar realizar la búsqueda de odorante de otra manera.

En la Figura que se muestra a continuación (véase Figura 5.52) se puede observar una prueba realizada utilizando este algoritmo y en la que aparecen los detalles comentados en el anterior párrafo. En ella se muestran los valores medidos por la Nariz Electrónica y las decisiones tomadas por el Robot ante tales valores. Las pruebas se realizaron con un rotulador para pizarras (que contiene alcohol), simulando la proximidad o lejanía de la fuente de odorante alejando o acercando el rotulador al sensor. El cuadrado formado por la línea discontinua señala las decisiones tomadas por el algoritmo y que no eran deseadas, tal y como se acaba de comentar.

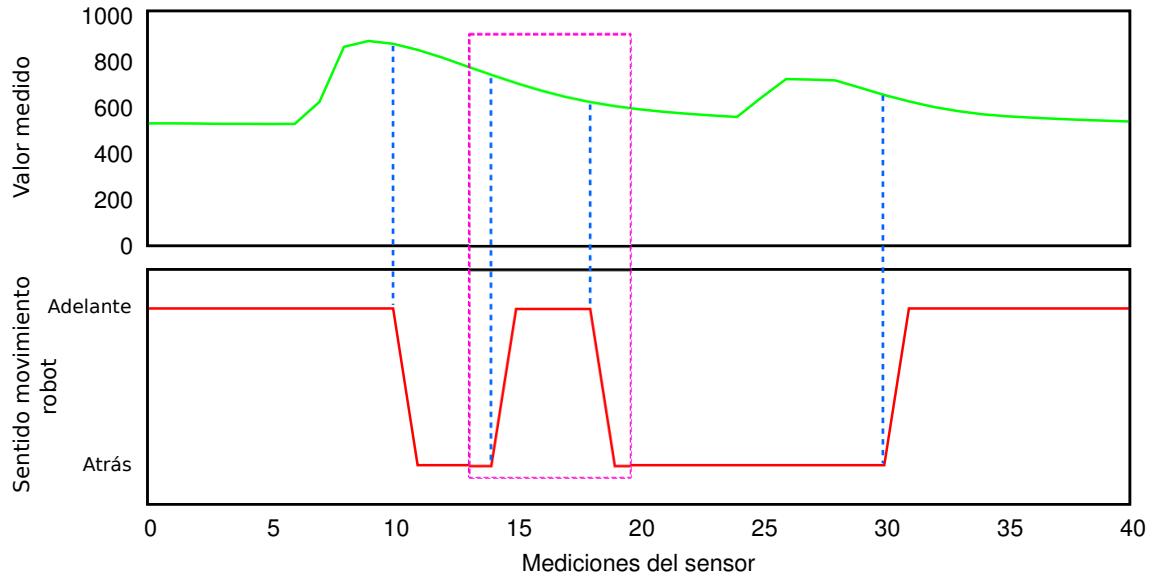


Figura 5.52: Señal de odorante comparada con el sentido de movimiento del robot

5.8.2. Mejora del algoritmo de búsqueda

Tras conocerse los defectos del primer algoritmo implementado, para este segundo se cambia la estrategia a llevar a cabo para aproximarse a la fuente de odorante. Primero, se debe observar el comportamiento mostrado por el sensor. Si se pintase el valor obtenido por la nariz en algún momento, el resultado sería muy parecido al mostrado en la siguiente imagen:

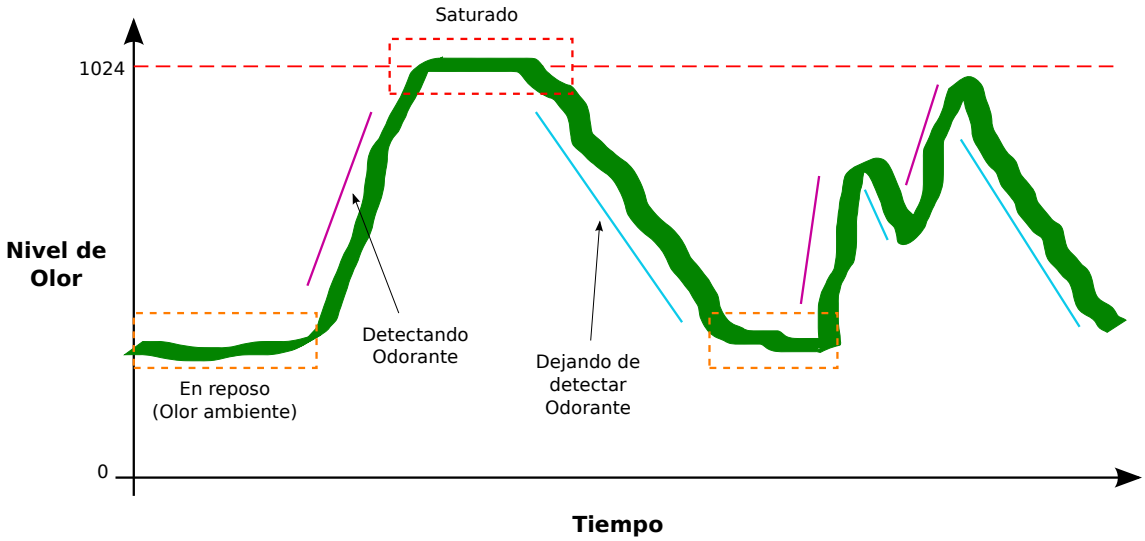


Figura 5.53: Valor obtenido por la Nariz Electrónica

En la imagen se pueden observar los distintos valores que pasa a captar el sensor si se procede a acercar y alejar de él una fuente de odorante. Empieza teniendo una medida influenciada por el olor presente en el medio en el que se encuentra la Nariz Electrónica. Si se aproxima la fuente de odorante, la medida comenzará a aumentar y en caso de que haya mucha presencia de odorante, la medida llegará a su punto máximo, saturándose. Si se aleja lo suficiente la fuente de odorante, la medida comenzará a bajar hasta volver a establecerse alrededor del valor medio presente en el entorno. Si se procede a acercar o alejar la fuente de odorante volveremos a ver situaciones similares, evitando una situación de saturación si no hay mucha presencia de odorante en el ambiente.

Si se traslada esta explicación al ámbito del algoritmo de búsqueda, se desea que el Robot cambie su sentido de desplazamiento cuando comience a disminuir el valor medido y así volver a aproximarse a la fuente de odorante. Esto se hará calculando la pendiente de la variación de las medidas realizadas por el sensor, y haciendo variar el sentido del desplazamiento en el momento en el que la medida realizada comience a disminuir tras haber aumentado previamente (es decir, en el momento en el que la pendiente sea negativa tras haber pasado por un periodo positivo).

El código que se ha desarrollado y que hace posible la implementación de este algoritmo es el siguiente:

```
i++;
if (i == 100){
  olor_m_actual = olor_m_actual + leeI2C();

  if (cont_algoritmo==NUM_VALORES_ALGORITMO-1){
    olor_m_actual = olor_m_actual/10;
    diferencia_actual = olor_m_actual - olor_m_previo;

   if ((diferencia_actual < 0) && (olor_m_actual <= (olor_m_previo*97/100)) && (
       puede_cambiar_sentido == 1)){
      sentido = !sentido;
      estableceSentido(sentido);
      puede_cambiar_sentido = 0;
    }
```

```
    if ((diferencia_actual > 0) && (olor_m_actual >= (olor_m_previo*97/100)) && (
        puede_cambiar_sentido == 0)){
      puede_cambiar_sentido = 1;
    }

    cont_algoritmo = 0;
    olor_m_previo = olor_m_actual;
    olor_m_actual = 0;
    diferencia_previa = diferencia_actual;

  }
  else
    cont_algoritmo++;

  i = 0;
}
```

Como en el primer algoritmo, se continúa haciendo una medida cada 100 pasos de simulación y realizando tras 10 medidas el cálculo con el que se decide el sentido de movimiento. En cambio ahora, además de prestar atención al umbral en el que se encuentra el valor medido, se prestará atención a la pendiente obtenida entre los dos últimos puntos medidos. Se observará si esta pendiente es negativa, y únicamente cambiará de sentido si previamente detectó la presencia de odorante. De esta manera se evitará que el Robot esté continuamente cambiando el sentido de movimiento mientras el valor de las medidas continúa disminuyendo. Además, se sigue comprobando el umbral del 97 % de la media anterior para así evitar cambiar el sentido del movimiento ante pequeñas variaciones en las medidas obtenidas.

Tras hacerse pruebas con este segundo algoritmo, se observa una buena respuesta por parte del robot (véase Figura 5.54). Ahora es capaz de cambiar el sentido de giro cuando se aleja de la fuente de odorante, y sólo vuelve a girar en el caso de que previamente su valor medido aumentase antes de volver a disminuir. El robot no es capaz de detectar la fuente de odorante, pero si es capaz de moverse alrededor de ella (en el caso en el que el odorante se encuentre situado en una zona muy concreta) como ya se explicó anteriormente (véase Figura 5.51). No será capaz de encontrar la fuente pero si de situarse próximo a ella.

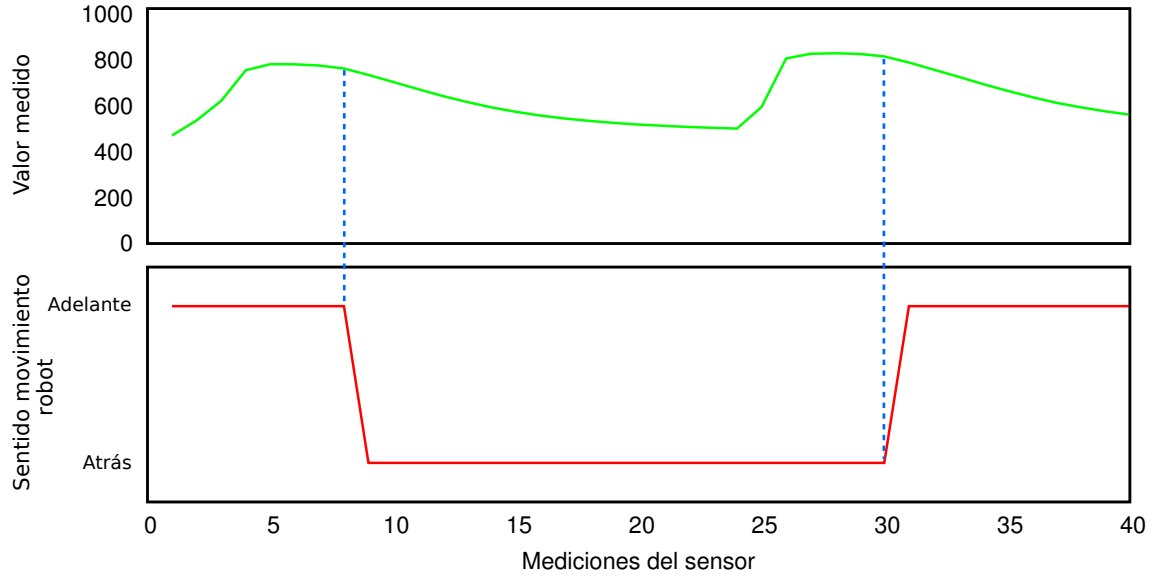


Figura 5.54: Señal de odorante comparada con el sentido de movimiento del robot (segundo algoritmo)

El funcionamiento del robot con este algoritmo se puede ver en un vídeo grabado con el fin de poder ser mostrado al público [100]. El vídeo se puede ver tanto en la web incluida en la

referencia cómo en los archivos que se encuentran adjuntos a este documento.

Tras observarse cómo con este algoritmo se han conseguido los objetivos buscados, se puede dar por concluida la búsqueda de la fuente de odorante en este proyecto. Se podría realizar una búsqueda más exhaustiva, pero el diseño de tal algoritmo podría ser el único propósito de un nuevo proyecto. La búsqueda precisa de la fuente de la que proviene el odorante no es el objetivo primario de este proyecto por lo que el segundo algoritmo propuesto se considera válido en referencia a los objetivos previamente establecidos.

6

Resumen de objetivos

A continuación se muestra la consecución de objetivos que se ha llegado a alcanzar. La lista de objetivos marcados, y la situación que se ha llegado a alcanzar en cada uno de ellos, es la siguiente:

- **Modelo de sinapsis eléctricas:**

  En la búsqueda de un modelo sencillo que no acarrease un alto coste computacional, se marcó como objetivo el utilizar sinapsis eléctricas en vez de las sinapsis químicas que ya se habían utilizado en trabajos anteriores de implementación en robots de este departamento. Se ha desarrollado un modelo con este tipo de sinapsis que, aunque ofrece una estabilidad algo menor a la de un modelo con sinapsis químicas (se aprecia en las gráficas, donde el modelo generado para este PFC muestra un valor menos plano en la representación de los desfases entre módulos), es completamente válido para la obtención de fases estables (como se ha mostrado en el Apartado 5.5.3).

- **Diseño del CPG para la generación de movimiento con el nuevo modelo sináptico e implementación de movimientos básicos:**

  Utilizando sinapsis químicas ya se había logrado obtener la generación de distintos tipos de movimiento. En este proyecto se busca hacer lo mismo pero con el modelo de sinapsis eléctricas desarrollado. Mediante el nuevo modelo desarrollado, primero se ha conseguido el moviente del robot en línea recta tanto hacia adelante como hacia atrás (desarrollado en el Apartado 5.5.4). A continuación se ha cambiado la configuración de los módulos del robot y se ha modificado el diseño del CPG, manteniendo el nuevo modelo desarrollado, para que Cube Revolutions pueda moverse tanto en línea recta como en una trayectoria circular (en el Apartado 5.6.3). Con el rediseño del CPG el robot tiene la posibilidad de desplazarse por todas las direcciones del plano horizontal.

- **Diseño de un algoritmo con el que calcular el desfase entre módulos:**

  Se diseña y desarrolla un algoritmo que permite calcular el desfase entre los módulos que conforman el robot. Se trata de un algoritmo propio y realmente sencillo que es capaz de detectar la diferencia de fase en cada instante entre módulos vecinos. También se puede configurar para calcular el desfase entre otros módulos (no vecinos directamente) pero para este proyecto el cálculo que interesa es el que se realiza entre los módulos vecinos y así poder ver si el desfase es estable o no y si se mantiene en los valores deseados. Se implementa en las simulaciones del CPG para así poder realizar un barrido paramétrico y filtrado automático posterior de los que son válidos para generar movimiento en el robot.

- **Control total por parte del microcontrolador:**

  Para la simulación de las neuronas, en anteriores trabajos se había recurrido a la simulación en un PC para posteriormente ir introduciendo los resultados en la placa según el microcontrolador fuese requiriendo los datos para controlar a los distintos servos. En este proyecto se busca realizar la simulación en tiempo real y desconectado del PC, es decir, ejecutando la simulación en el propio microcontrolador. En un primer intento se descubre que la placa utilizada en anteriores proyectos, SkyPIC, no sirve en este proyecto al no disponer de memoria suficiente para contener las variables necesarias a la hora de ejecutar la simulación (en el Apartado 5.6.1). Se consigue simular el modelo neuronal desarrollado ejecutándolo en una placa Arduino (como se muestra en el Apartado 5.6.2). Tras observar que puede ejecutar la simulación neuronal, se procede a conectar la placa al robot y así se comprueba que el CPG diseñado es válido al obtenerse desplazamiento con los movimientos del robot (en el Apartado 5.6.3).

- **Establecimiento de comunicación I2C entre los dispositivos:**

  A la hora de conectar los dispositivos se decide realizar la comunicación a través del protocolo I2C. La Nariz Electrónica no posee ningún módulo programado encargado de gestionar las comunicaciones I2C así que se desarrolla un módulo (a bajo nivel debido a la falta de bibliotecas) para que se encargue de realizar esa tarea (en detalle en el Apartado 5.7.2). Debido a que no existía una librería funcional ni tampoco se encuentra una documentación detallada en internet, se puede decir que se ha desarrollado algo original y novedoso. También se desarrolla un módulo similar en Arduino, haciendo uso de las bibliotecas disponibles y posicionando a la placa controladora del robot como Maestro en la comunicación I2C (en el Apartado 5.7.3). La comunicación se establece de la manera Maestro(robot)-Esclavo(nariz), aunque el protocolo admite varios modos de funcionamiento (comentados en el Apartado 3.11). Se establece un formato de trama para el envío de información entre los dispositivos, y así poder configurar los parámetros necesarios o comunicar la información obtenida por los sensores (en el Apartado 5.7.3).

- **Integración de los elementos en el robot:**

  En pruebas realizadas en los trabajos precedentes a este proyecto se encontraron con la limitación que ofrecía el robot a la hora de ejecutar los movimientos. Al estar la SkyPIC conectada entre el PC y el robot, y además situarse fuera de la estructura de Cube Revolutions, se producían numerosas desconexiones con los movimientos del robot. Al utilizar Arduino se elimina la conexión PC-placa y su problemática, pero se pierde robustez en la conexión robot-Arduino al no disponer del conector adecuado (comentado en el Apartado 5.6.2). Para solucionarlo se utiliza SkyMEGA, microcontrolador que se acomoda perfectamente a la estructura del robot atornillándose a ella (como se puede ver en el Apartado 5.6.3). Este microcontrolador es compatible con la plataforma Arduino y además dispone de un conector similar al de SkyPIC, con lo que mejora la conectividad con el robot respecto a Arduino. Se realiza una adaptación del conector de la fuente de alimentación para evitar desconexiones provocadas por los movimientos del robot. También se fija la nariz al módulo delantero del robot para así ofrecer medidas que proceden realmente de el lugar donde se encuentra Cube Revolutions (como se puede ver en el Apartado 5.8).

- **Variación del movimiento ante estímulos olfativos:**

  Se diseña un algoritmo que, utilizando los valores medidos por la nariz electrónica y transferidos a Arduino mediante la comunicación I2C, es capaz de cambiar el sentido del movimiento que presenta el robot (desarrollado en el Apartado 5.8.1). Se modifica el algoritmo para evitar que cambie continuamente el sentido del movimiento al continuar disminuyendo el valor medido (en el Apartado 5.8.2). Estos algoritmos se desarrollan como prueba de concepto para presentar la interacción entre el Robot y la Nariz Electrónica.

7
Conclusiones

Los objetivos de este proyecto han quedado explicados en el capítulo anterior, donde se ha visto que todos o casi todos los hitos han sido cumplidos.

Se ha visto como al utilizar un modelo de sinapsis eléctrico se ha podido desarrollar un CPG que permite generar señales locomotoras aptas para conseguir movimiento por parte del robot. Gracias a estas sinapsis se han simplificado en gran medida los cálculos en el CPG respecto al modelo que se presentaba en trabajos anteriores del departamento. Esto ha permitido ejecutar la simulación del Sistema Neuronal en un microcontrolador en tiempo real, es decir, el CPG se ejecuta dentro del robot. La generación de la locomoción en tiempo real a su vez permite la modificación de los parámetros del CPG en el momento que se quiera realizar, como por ejemplo al detectar menor concentración de odorante.

No hay que olvidar los puntos que han hecho utilizar un CPG en vez de señales sinusoidales para la obtención de los movimientos del robot, pues bien es cierto que los recursos computaciones empleados son mucho mayores, pero a cambio se ha obtenido una mayor robustez ante posibles errores aportando una recuperación segura sea cual sea el estado en el que se encuentre. Esta recuperación se puede apreciar al realizar el cambio de parámetros en las conductancias del CPG, situación en la que el modelo es capaz de estabilizarse pasado un tiempo de reajuste (como se puede ver en el Apartado 5.5.4).

Se puede pensar que quizás tal complejidad computacional no compense tanto ya que, al unir los cálculos que se tienen que realizar para la simulación neuronal y las operaciones que se necesitan para la gestión de la comunicación I2C, la placa SkyMEGA trabaja al límite de sus posibilidades (se tuvieron que realizar ajustes en el CPG para simplificar los cálculos y para que los incrementos entre pasos de la simulación fueran mayores, cambios detallados en el Apartado 5.6.2). Si se añadiese, por ejemplo, un feedback que devolviese la posición real de los servos (y que así pudiese auto-ajustarse el CPG los parámetros para mostrar un funcionamiento óptimo) cuestionaría la viabilidad de esta placa para el proyecto, al estar posiblemente funcionando al límite de sus capacidades. En cambio, los recursos aquí empleados no supondrían ningún problema en los sistemas embebidos de bajo coste que están apareciendo actualmente en el mercado [104, 105], los cuales pueden dar lugar a robots eficientes y muy baratos.

Por otro lado, se ha conseguido establecer una comunicación sencilla y fiable entre la Nariz Electrónica y el robot mediante el uso del protocolo I2C. Esto permite utilizar a Olus como guía para el robot, cambiando el desplazamiento que esté ofreciendo debido a los odorantes recibidos por la Nariz Electrónica. Cube Revolutions consulta a Olus el valor que ha medido cada cierto

periodo de tiempo, y en función de la variación de este valor toma unas decisiones u otras.

Como prueba de concepto de la comunicación desarrollada, se ha diseñado un algoritmo de interacción con odorantes. Gracias a este algoritmo, el robot es capaz de situarse próximo a la fuente de odorante. Se sitúa próximo a esta fuente y se queda oscilando alejándose y acercándose continuamente a su alrededor, no detectando su localización de manera exacta pero si delimitándola en un círculo de acción (allí donde la concentración de odorante es mayor). De ahí en adelante hay mucho trabajo por hacer, tanto que da para hacer un proyecto con ese como único objetivo, ya que se trata de un problema realmente complejo. Hay numerosas estrategias posibles, en alguna de las cuales ya se está trabajando activamente en el GNB, como es la detección de la fuente mediante el uso de robótica colaborativa.

Cabe destacar que la electrónica que ha sido utilizada en este proyecto es muy sencilla y barata, pudiéndose así desarrollar robots sencillos pero con una gran capacidad para poder abordar proyectos realmente interesantes. El campo de los sensores olfativos no está del todo desarrollado aún y es un buen momento para hacer grandes avances, y para ello utilizar a Olus como mediador para captar las partículas que definen los olores.

Si en vez del microcontrolador SkyMega se utilizase un sistema embebido (como los existentes en la actualidad que son capaces de ejecutar GNU/Linux), además de permitir hacer más complejo el CPG, facilitaría la búsqueda de la fuente de odorante o bien la distinción entre diferentes odorantes mediante el uso de algoritmos con fuerte carga computacional. Debido a esas dos razones parece acertado apostar por los sistemas embebidos [104, 105] y portar a esta tecnología el desarrollo realizado para futuros proyectos.

Por último, comentar que se ha mejorado la movilidad respecto a los trabajos realizados anteriormente en el GNB con este robot. Al implementar la placa controladora en la estructura del robot y al haber conseguido ejecutar el CPG íntegramente en el interior de la placa, se ha conseguido desarrollar un robot autónomo en el que las decisiones se toman en tiempo real. Al no disponer de conexiones externas hacia fuera de la estructura (sólo está conectado a la fuente de alimentación, tratándose de una conexión robusta) se ha conseguido una operatividad mayor por parte del robot, permitiendo al robot realizar movimientos sin temor a su desconexión.

8

Trabajo futuro

El trabajo presentado en este Proyecto Final de Carrera tiene muchas semillas para desarrollar diferentes ramas en un trabajo futuro. Algunos de estos posibles puntos a seguir en futuros proyectos son:

- Desarrollar más movimientos para el robot, tal y como hicieron Damián Zamorano y Juan González en sus respectivos trabajos. Juan utilizó para ello generadores sinusoidales, pero Damián desarrolló varios CPGs en función del movimiento a realizar. Sería interesante replicar esos CPGs utilizando para ello sinapsis eléctricas, haciéndolo aún más interesante si se consiguiese juntar todos los CPGs en uno sólo y en el que únicamente se tuviese que cambiar las conductancias para cambiar el movimiento.
- Hacer un simulador completo del CPG en el que se pudiese cambiar aspectos tales como el número de neuronas, las conexiones entre ellas, el valor de la conductancia de esas conexiones e incluso el modelo de simulación de cada neurona (utilizar otro modelo distinto al de Rulkov si así se prefiere) o el modelo de la sinapsis (eléctrica o química), etc.
- Hacer que el robot sea totalmente autónomo suministrándole la alimentación mediante baterías situadas en su estructura.
- Aportar al CPG un feedback sensorial como es el que conozca el ángulo real en el que se encuentran los servos y así pueda auto-ajustar su ritmo a lo largo de la simulación.
- Diseñar un algoritmo de búsqueda para encontrar la fuente de odorante. El trabajo realizado en este proyecto referente a ese aspecto es poco respecto a lo que se puede hacer. Convendría estudiar los distintos algoritmos de búsqueda que existen e implementarlos en este sistema para así poder ver mejores resultados.
- Por último, y como ya se ha comentado en el capítulo anterior, sería muy interesante trasladar el trabajo realizado al área de los sistemas embebidos ya que hay alternativas muy baratas y que podrían ofrecer un alto nivel de procesamiento y facilidad de programación a la hora de implementar los algoritmos (estas máquinas pueden funcionar bajo GNU/Linux). Existen muchas opciones en el mercado, entre las que se encuentran proyectos muy interesantes y prometedores como Raspberry Pi [104] o Cubieboard [105].

Glosario de acrónimos

- **CPG**: Generador Central de Patrones
- **EN**: Nariz Electrónica
- **GNB**: Grupo de Neurocomputación Biológica
- **I2C**: Inter-Circuitos Integrados
- **IDE**: Entorno de Desarrollo Integrado
- **PCB**: Printed Circuit Board

Bibliografía

[1] WikiRobotics: Página web de Juan González con documentación de sus programas y robots. `http://www.iearobotics.com/wiki/index.php?title=P%C3%A1gina_Principal`. Accedido: 15/03/2013.

[2] F. Herrero-Carrón. *Study and application of central patter generators to the control of a modular robot*. PhD thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2007.

[3] D. Zamorano. PFC: Control locomotor multidireccional de un robot modular mediante Circuitos Generadores Centrales de Patrones Bioinspirados, 2011.

[4] D.J. Yáñez. Trabajo fin de master: Estrategias bioinspiradas para la adquisición de olores en narices artificiales, 2009.

[5] Deutecno: empresa de david yáñez. `http://www.deutecno.com/`. Accedido: 15/03/2013.

[6] D.J. Yáñez, A. Toledano, E. Serrano, A.M. Martín de Rosales, F. B. Rodríguez, and P. Varona. Characterization of a clinical olfactory test with an artificial nose. *Frontiers in Neuroengineering*, 5:1, 2012.

[7] E. Marder and D. Bucher. Central pattern generators and the control of rhythmic movements. *Current Biol*, 11:R986–R996, 2001.

[8] A.I. Selverston, M.I. Rabinovich, H.D.I. Abarbanel, R. Elson, A. Szucs, R.D. Pinto, R. Huerta, and P. Varona. Reliable circuits from irregular neurons: a dynamical approach to unterstanding central pattern generators. *J. Physiol. (Paris)*, 94:357–374, 2000.

[9] A. Selverston. *Cellular and Molecular Mechanisms Underlying Higher Neural Functions*. John Wiley Sons, 1994.

[10] P. Varona R. Latorre, F.B. Rodríguez. Characterization of triphasic rhythms in central pattern generators (i): Interspike interval analysis. *Lect. Notes Comput. Sc.*, 2415:160–166, 2002.

[11] P. Varona R. Latorre, F.B. Rodríguez. Characterization of triphasic rhythms in central pattern generators (ii): Burst information analysis. *Lect. Notes Comput. Sc.*, 2415:167–173, 2002.

[12] P. Varona, J.J. Torres, R. Huerta, and H.D.I. Abarbanel. Regularization mechanisms of spiking-bursting neurons. *Neural Networks*, 14:865–875, 2001.

[13] F.B. Rodríguez, P. Varona, R. Huerta, M.I. Rabinovich, and H.D.I. Abarbanel. Richer network dynamics of intrinsically non-regular neurons measured through mutual information. *Lect. Notes Comput. Sc.*, 2084:490–497, 2001.

[14] T.G. Brown. On the nature of the fundamental activity of the nervous centres; together with an analysis of the conditioning of rhythmic activity in progression, and a theory of the evolution of function in the nervous system. *J Physiol*, 48:18–46, 1914.

[15] R. Huerta, P. Varona, M.I. Rabinovich, and H.D.I. Abarbanel. Topology selection by chaotic neurons of a pyloric central pattern generator. *Biological Cybernetics*, 84 (1):L1–L8, 2001.

[16] G.R. Stiesberg, M.B. Reyes, P. Varona, R.D. Pinto, and R. Huerta. Connection topology selection in central pattern generators by maximizing the gain of information. *Neural Computation*, 19 (4):974–993, 2007.

[17] R. Latorre, F.B. Rodríguez, and P. Varona. Signature neural networks: Definition and application to multidimensional sorting problems. *IEEE Transactions on Neural Networks*, 22(1):8–23, 2011.

[18] R. Latorre, F.B. Rodríguez, and P. Varona. Neural signatures: Multiple coding in spiking bursting cells. *Biological Cybernetics*, 95(2):169–183, 2006.

[19] M.I. Rabinovich, P. Varona, A.I. Selverston, and H.D.I. Abarbanel. Dynamical principles in neuroscience. *Reviews of Modern Physics*, 78(4):1213–1265, 2006.

[20] S. Grillner. Biological pattern generation: The cellular and computational logic of networks in motion. *Neuron*, 52:751–766, 2006.

[21] E. Marder and D. Bucher. Understanding circuit dynamics using the stomatogastric nervous system of lobsters and crabs. *Annu Rev Physiol*, 69:291–316, 2007.

[22] M. Reyes, R. Huerta, M. Rabinovich, and A. Selverston. Artificial synaptic modification reveals a dynamical invariant in the pyloric cpg. *Eur. J. Appl. Physiol*, 102:667–675, 2008.

[23] C. Di Natale, F.A.M. Davide, A. D'Amico, P. Nelli, S. Groppelli, and G. Sberveglieri. An electronic nose for the recognition of the vineyard of a red wine. *Sensors and Actuators B: Chemical*, 33:83–88, 1996.

[24] K. Brudzewski, S. Osowski, and A. Dwulit. Recognition of coffee using differential electronic nose. *Instrumentation and Measurement, IEEE Transactions on*, 61(6):1803–1810, 2012.

[25] A. Loutfi and S. Coradeschi. Relying on an electronic nose for odor localization. In *Virtual and Intelligent Measurement Systems, 2002. VIMS '02. 2002 IEEE International Symposium on*, 2012.

[26] A.T. Hayes, Alcherio Martinoli, and R.M. Goodman. Swarm robotic odor localization: Off-line optimization and validation with real robots. *Robotica*, 21(4):427–441, 2003.

[27] W. Li and M.M. Elgassier. 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*, 2006.

[28] A. Kuzu, S. Bogosyan, and M. Gokasan. Survey: detection, recognition and source localization of odor. *WTOS*, 7(6):611–621, 2008.

[29] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata. Distributed selfreconfiguration of m-tran iii modular robotic system. *Int. J. Robot. Res*, 27:373–386, 2008.

[30] M. Yim, D.G. Duff, and K.D. Roufas. Polybot: a modular reconfigurable robot. *Proc. IEEE Int. Conf. Robotics and Automation*, 1:514–520, 2000.

[31] J. Ayers and J. Witting. Biomimetic approaches to the control of underwater walking machines. *Phil. Trans. R. Soc. A*, 365:273–295, 2007.

[32] P. Arena, L. Fortuna, M. Frasca, and L. Patane. A cnn-based chip for robot locomotion control. *IEEE Trans. Circuits Syst*, 52:1862–1871, 2005.

[33] S-J. Chung and M. Dorothy. Neurobiologically inspired control of engineered flapping flight. *J. Guid. Control Dyn*, 33:440–453, 2009.

[34] K. Seo, S-J. Chung, and J.J. Slotine. Cpg-based control of a turtle-like underwater vehicle. *Auton. Robots*, 28:247–469, 2010.

[35] P. Manoonpong, T. Geng, T. Kulvicius, and B. Porr. Adaptive, fast walking in a biped robot under neuronal control and learning plos. *Comput. Biol*, 3:1305–1320, 2007.

[36] S. Aoi and K. Tsuchiya. Stability analysis of a simple walking model driven by an oscillator with a phase reset using sensory feedback. *IEEE Trans. Robot*, 22:391–407, 2006.

[37] A.J. Ijspeert and J. Kodjabachian. Evolution and development of a central pattern generator for the swimming of a lamprey. *Artif. Life*, 5:247–269, 1999.

[38] F. Herrero-Carrón, F.B. Rodríguez, and P. Varona. Flexible entrainment in a bio-inspired modular oscillator for modular robot locomotion. *Lect Notes Comput. Sc*, 6692:532–539, 2011.

[39] F. Herrero-Carrón, F.B. Rodríguez, and P. Varona. Dynamical invariants for cpg control in autonomous robots. *Proceedings of ICINCO, 7th International Conference on Informatics in Control, Automation and Robotics*, 7:441–445, 2010.

[40] F. Herrero-Carrón, F.B. Rodríguez, and P. Varona. Novel modular cpg topologies for modular robotics. *Proceedings of the IEEE 2010 International Conference on Robotics and Automation workshop*, pages 82–93, 2010.

[41] F. Herrero-Carrón, F.B. Rodríguez, and P. Varona. Studying robustness against noise in oscillators for robot control. *Proceedings of the ARCS Conference, Orlando*, pages 58–64, 2010.

[42] I. Urziceanu, F. Herrero-Carrón, J. González-Gómez, M. Nitulescu, F.B. Rodríguez, and P. Varona. Central pattern generator control of a differential wheeled robot. *System Theory, Control and Computing (ICSTCC)*, 15:1–6, 2011.

[43] D.P. Pulsifer and A. Lakhatia. Background and survey of bioreplication techniques. *Bioinspiration & Biomimetics*, 6(3):031001, 2011.

[44] N.F. Rulkov. Modeling of spiking-bursting neural behavior using two-dimensional map. *American Physical Society*, 65:041922, 2002.

[45] A. Destexhe, Z.F. Mainen, and T.J. Sejnowski. An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. *Neural Computation*, 6:14–18, 1994.

[46] E.M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. MIT Press, 2007.

[47] E.R. Kandel, J.H. Schwartz, and T.M. Jessell. *Principles of Neural Science*. Elsevier, 1981.

[48] J.A. Siguenza. *Neurocomputacin: Cómo funciona el cerebro*. Eudema Biología, 1993.

[49] G.M. Shepherd. *The Synaptic Organization of the Brain*. Oxford University Press, 2004.

[50] K. Matsuoka. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological Cybernetics*, 56:345–353, 1987.

[51] T. Matsuo and K. Ishii. A cpg control system for a modular type mobile robot. *International Congress Series*, 1301:206–209, 2007.

[52] G. Liu, M. Habib, K. Watanabe, and K. Izumi. Central pattern generators based on matsuoka oscillators for the locomotion of biped robots. *Artificial Life and Robotics*, 12:264–269, 2008.

[53] U. Zimmerman and G.A. Neil. *Electromanipulation of Cells*. CRC Press Inc, 1996.

[54] T.P. Trappenberg. *Fundamentals of Computational Neuroscience*. Oxford University Press, 2010.

[55] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133, 1943.

[56] D.A. Patterson and R.C. Corretger J.L. Hennessy. *Estructura y diseño de computadores: Interfaz circuitería - programación*. Reverte, 2000.

[57] M. Minsky and S. Papert. *Perceptrones*. MIT Press, 1969.

[58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 6:386408, 1958.

[59] J. González-Gómez. *Diseño de robots ápodos*. PhD thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2003.

[60] M. Yim. *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Stanfofd University, 1995.

[61] M. Yim, Y. Zhang, and D. Duff. Modular robots. *IEEE Spectrum*, 39(2)::30–34, 2002.

[62] M. Yim, D. Duff, and K. Roufas. Modular reconfigurable robots, an aproach to urban search and rescue. *In Proc. of 1st Intl. Workshop on Human-friendly welfare Robotic Systems*, 1:69–76, 2000.

[63] J.W. Suh, S.B. Homans, and M. Yim. Telecubes: mechanical design of a module for selfreconfigurable robotics. *In Proceeings of the IEEE Int. Conf. on Robotics and Automation*, 4:4095–4101, 2002.

[64] G.J. Taylor, R.C.F. Lentz, S.J. Lawrence, L.M. Martel, W-M. Shen, P.M. Will, M.H. Sims, S. Colombano, D. Kortenkamp, B. Damer, and W. Chun. Superbots on the lunar surface: Mini-mobile investigation system (mini-mis). *Space Resources Roundtable*, 7:86, 2005.

[65] Cube Revolutions: Página web de Juan González con documentación de cube revolutions. `http://www.iearobotics.com/wiki/index.php?title=Cube_Revolutions`. Accedido: 15/03/2013.

[66] A.J. Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21:642–653, 2008.

[67] C. Stefanini, G. Orlandi, A. Menciassi, Y. Ravier, G.L. Spina, and S. Grillner. A mechanism for biomimetic actuation in lamprey-like robots. *Biomedical Robotics and Biomechatronics*, 1:579–584, 2006.

[68] B. Klaassen, R. Linnemann, D. Spenneberg, and F. Kirchner. Biomimetic walking robot scorpion: control and modeling. *Autonomous Robots*, 41:69–76, 2002.

[69] J. Conradt and P. Varshavskaya. Distributed central pattern generator control for a serpentine robot. In *In Proceedings of the International Conference on Artificial Neural Networks and Neural Information Processing*, 2003.

[70] H. Kimura, S. Akiyama, and K. Sakurama. Realization of dynamic walking and running of the quadruped using neural oscillators. *Autonomous Robots*, 7(3:247–258, 1999.

[71] A.J. Ijspeert, A. Crespi, D. Ryczko, and J.M. Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315(5817):1416–1420, 2007.

[72] G. Taga, Y. Yamaguchi, and H. Shimizu. Selforganized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, 65:147–159, 1991.

[73] D. Fernández. PFC: Diseño de un interfaz hombre hombre-máquina con un observador dinámico en tiempo real, 2011.

[74] T.C. Pearce, S.S. Schiffman, H.T. Nagle, and J.W. Gardner. *Handbook of Machine Olfaction - Electronic Nose Technology*. John Wiley & Sons, 2003.

[75] L. Marques, N. Almeida, and A.T. Almeida. Olfactory sensory system for odour-plume tracking and localization. *Sensors*, 1:418–423, 2003.

[76] Microchip: Página web de Microchip con la documentación de sus microcontroladores. `http://www.microchip.com/`. Accedido: 15/03/2013.

[77] Atmel: Página web de Atmel con la documentación de sus microcontroladores. `http://www.atmel.com/`. Accedido: 15/03/2013.

[78] GNU: Página web oficial de GNU con información sobre software libre. `http://www.gnu.org/home.en.html`. Accedido: 15/03/2013.

[79] Arduino: Página web de la plataforma de hardware libre Arduino. `http://www.arduino.cc/`. Accedido: 15/03/2013.

[80] Wiring: Página web de Wiring, el lenguaje de programación para Arduino. `http://wiring.org.co/`. Accedido: 15/03/2013.

[81] SkyMega: Página web de Juan González con documentación de SkyMega. `http://www.iearobotics.com/wiki/index.php?title=SkyMega`. Accedido: 15/03/2013.

[82] I2C: Página web con documentación del protocolo de comunicación I2C. `http://www.i2c-bus.org/i2c-bus/`. Accedido: 15/03/2013.

[83] Módulos Y1: Página web de Juan González con documentación de los módulos y1. `http://www.iearobotics.com/wiki/index.php?title=M%C3%B3dulos_Y1`. Accedido: 15/03/2013.

[84] R. Latorre, F.B. Rodríguez, and P. Varona. Reaction to neural signatures through excitatory synapses in central pattern generator models. *Neurocomputing*, 70(10-12):1797–1801, 2007.

[85] J.B. Thuma, L.G. Morris, A.L. Weaver, and S.L. Hooper. Lobster (panulirus interruptus) pyloric muscles express the motor patterns of three neural networks, only one of which innervates the muscles. *Journal of Neuroscience*, 23:8911–8920, 2003.

[86] TGS2600: Página web del sensor de la casa figaro. `http://www.figarosensor.com/products/2600pdf.pdf`. Accedido: 15/03/2013.

[87] Tutorial ODE: Tutorial de Juan González sobre ODE. `http://www.iearobotics.com/wiki/index.php?title=Tutorial:_ODE_y_robots_modulares`. Accedido: 15/03/2013.

[88] BULLET: Página web oficial del simulador de físicas BULLET. `http://bulletphysics.org/wordpress/`. Accedido: 15/03/2013.

[89] OGRE3D: Página web oficial del motor gráfico OGRE3D. `http://www.ogre3d.org/`. Accedido: 15/03/2013.

[90] MKFIFO: Página web de la utilidad de GNU/Linux MKFIFO. `http://unixhelp.ed.ac.uk/CGI/man-cgi?mkfifo`. Accedido: 15/03/2013.

[91] SkyPic: Página web de Juan González con documentación de SkyPic. `http://www.iearobotics.com/wiki/index.php?title=Skypic`. Accedido: 15/03/2013.

[92] star-servos8: Página web de Juan González con documentación del programa star-servos8. `http://www.iearobotics.com/proyectos/stargate/servidores/sg-servos8/sg-servos8.html`. Accedido: 15/03/2013.

[93] Libstargate: Página web de la librería libstargate de Juan González. `http://www.iearobotics.com/proyectos/stargate/clientes/libstargate/libstargate.html`. Accedido: 15/03/2013.

[94] Libiris: Página web de la librería libiris de Juan González. `http://www.iearobotics.com/wiki/index.php?title=LibIris`. Accedido: 15/03/2013.

[95] SDCC: Página web del compilador de lenguaje C para microcontroladores SDCC. `http://sdcc.sourceforge.net/`. Accedido: 15/03/2013.

[96] Pydownloader: Página web del cargador de programas para la Skypic. `http://www.iearobotics.com/wiki/index.php?title=Pydownloader-wx`. Accedido: 15/03/2013.

[97] Arduino UNO: Página web de Arduino UNO. `http://arduino.cc/en/Main/ArduinoBoardUno`. Accedido: 15/03/2013.

[98] Biblioteca Servo Arduino: Página web de la biblioteca para control de servos en Arduino. `http://arduino.cc/en/Reference/Servo`. Accedido: 15/03/2013.

[99] Biblioteca Serial Arduino: Página web de la biblioteca para comunicación serie en Arduino. `http://arduino.cc/en/Reference/Serial`. Accedido: 15/03/2013.

[100] Video comportamiento robot: Muestra el comportamiento del robot cube revolutions ante los estímulos olfativos recibidos. `http://youtu.be/3UnMFSic4RI`. Accedido: 15/03/2013.

[101] Pickit 2: Página web del programador de PIC pickit2. `http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023805`. Accedido: 15/03/2013.

[102] Biblioteca I2C Arduino: Página web de la biblioteca para comunicación I2C en Arduino. `http://arduino.cc/en/Reference/Wire`. Accedido: 15/03/2013.

[103] Arduinoscope: Página web del programa Arduinoscope. `http://code.google.com/p/arduinoscope/`. Accedido: 15/03/2013.

[104] Raspberry Pi: Página web del sistema embebido de bajo coste. `http://www.raspberrypi.org/`. Accedido: 15/03/2013.

[105] Cubieboard: Página web del sistema embebido de bajo coste. `http://www.cubieboard.org/`. Accedido: 15/03/2013.

A

Documentación técnica de la comunicación

Para poder establecer comunicación entre el Robot y la Nariz Electrónica se ha utilizado el protocolo I2C. En este apartado se trata de explicar todos aquellos detalles que no hayan quedado claros a lo largo del documento o simplemente poner todos estos detalles juntos para poder tener una visión global de la comunicación más fácilmente.

En el protocolo I2C hay un elemento maestro y el resto son elementos esclavo. En la comunicación implementada la SkyMega es el dispositivo maestro mientras que Olus2 es el dispositivo esclavo (véase Figura A.1).

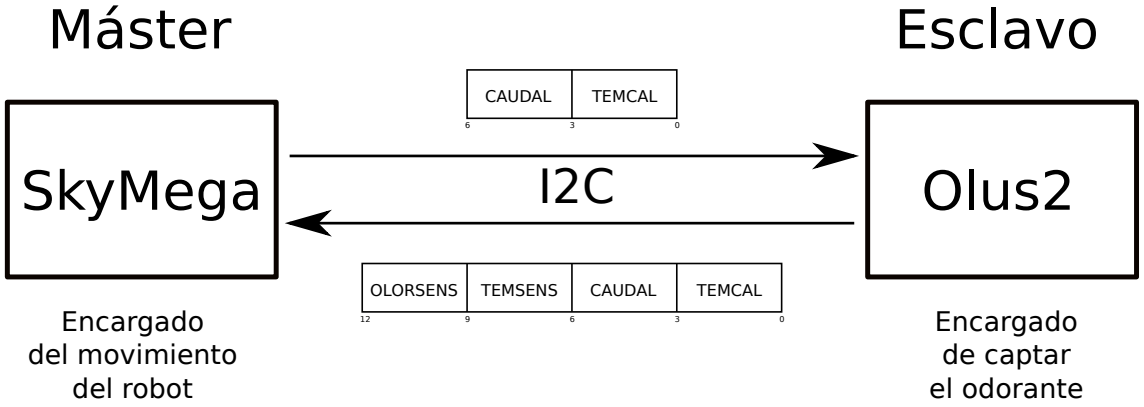


Figura A.1: Arquitectura comunicaciones entre Robot y Nariz Electrónica

La comunicación I2C se realiza a través de un bus que incluye 4 señales. SDA y SCL, señales propias de la comunicación, y VCC y GND, que se encargan de la alimentación de Olus2. Las conexiones entre los diferentes pines de las placas de control se encuentran detalladas en la Figura A.2.

A este bus se le pueden añadir más dispositivos en caso de que se necesitasen (tales como sensores de proximidad, giroscopios o una memoria de datos). Sólo habría que utilizar una dirección para Olus2 diferente a la que tengan por defecto estos dispositivos (como se comentó en el Apartado 4.7 los dispositivos comerciales disponen de una dirección I2C estándar preasignada según la función que desempeñen).

Por otro lado, se puede hacer que Olus2 sea el maestro de la comunicación y SkyMega el esclavo, liberando así de carga a el microcontrolador encargado de controlar el movimiento del robot. Se decidió no hacerlo en este proyecto para así mantener a la Nariz Electrónica lo más

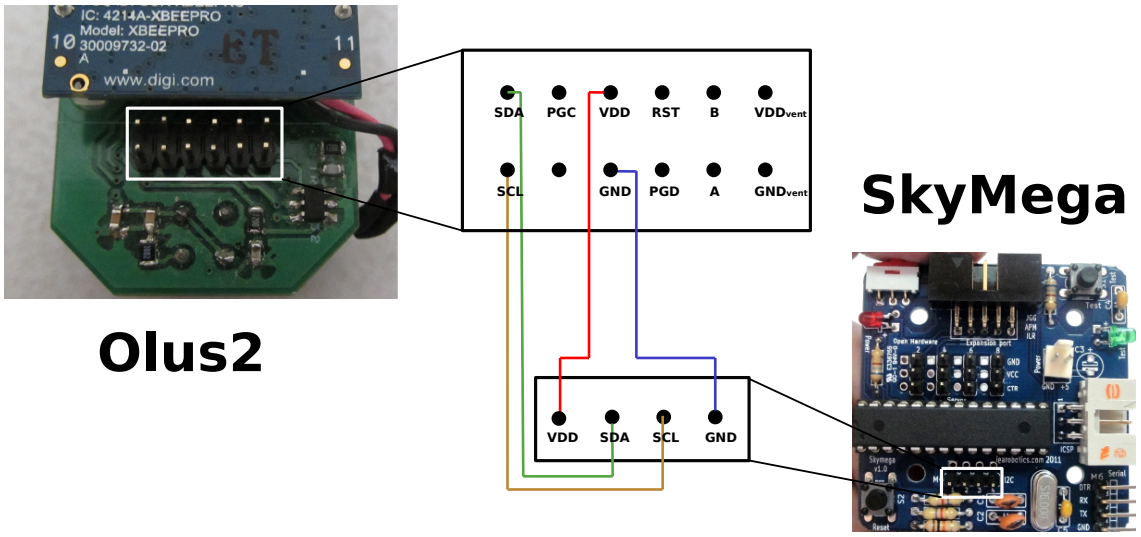


Figura A.2: Diagrama de interconexión de SkyMega y Olus2

sencilla posible y que sea el elemento encargado de controlarla el que se ocupe de decidir el momento en el que se deben obtener medidas y el cambio de los parámetros de Olus2. De esta manera Olus2 es un modelo caja cerrada en la que únicamente se configuran sus parámetros mediante una trama de envío (máster-¿esclavo) y devuelve las medidas obtenidas mediante una trama de recepción (esclavo-¿máster), tramas que tienen un formato único definido (explicado en el Apartado 5.7.3).

El protocolo tiene tres mensajes posibles si se observa la comunicación desde una perspectiva de alto nivel (los mensajes en bajo nivel fueron introducidos en el Apartado 4.7). Estos tres mensajes son:

- **Arranque:** Desde SkyMega se indica la dirección I2C de Olus2, la velocidad de transmisión que se desea, el modo de funcionamiento de I2C que se va a utilizar. Por último se envía una condición de inicio (condición de bajo nivel introducida en la Figura 4.34) a Olus2 para a partir de entonces esté a la escucha de los mensajes con datos que envíe el Maestro. Los parámetros utilizados para la configuración inicial son:
  - Dirección Maestro (SkyMega): No posee dirección.
  - Formato dirección: 7 bits (Se puede utilizar un formato de 10 bits en caso de que se deba conectar más de 128 dispositivos al bus).
  - Dirección Esclavo (Olus2): 0xA0. Debido al tratamiento que hace de los bits de la dirección, en SkyMega se debe configurar la dirección como 0x50 (se obtiene retirando el bit de menor peso de la dirección de Olus2). Este detalle se encuentra explicado en el Apartado **??**.
  - Velocidad de transmisión: Se configura para realizar la comunicación a 100 kHz.

- **Transmisión:** Una vez iniciada la comunicación se comienza a realizar la transmisión de datos entre los dos dispositivos. La SkyMega se encarga de solicitar información a Olus2 cada 0,1 segundos y Olus2 le responde transmitiendo los valores que obtenido por medio de sus sensores. La disposición de las tramas se explicó en el Apartado 5.7.3 y también se puede observar de manera esquemática en la Figura A.1. Una breve descripción de las mismas es:
  - Petición (SkyMega Olus2): Trama de 6 bytes que se utiliza para solicitar los datos medidos a Olus2. Además se aprovecha esta trama para enviar dos parámetros de configuración de la Nariz Electrónica. La trama incluye los valores que debe mostrar Olus2 en el ventilador y en el calefactor.

- Recepción (Olus2 SkyMega): Trama de 12 bytes con la que responde Olus2 a la petición de SkyMega. Incluye el valor de 4 variables: los valores que tiene fijados en el ventilador y en el calefactor (que deben corresponder a los valores enviados por SkyMega en la trama anterior si la comunicación se está realizando correctamente) y los valores que ha medido mediante el sensor de temperatura y el sensor químico.

Cada uno de los valores enviados se trata de un valor de 3 bytes cuyo formato, tanto de recepción como de envío, es de tipo carácter UNSIGNED CHAR. Para poder operar con estos valores, nada mas recibirlos se envían a una función del código cuya única tarea se rata de convertirlos al formato numérico UNSIGNED INT.

- **Finalización:** Una vez se termina la necesidad de enviar datos entre los dispositivos se debe cortar la comunicación. En este proyecto el robot se encuentra moviéndose en todo momento y no tiene un final de ejecución definido, sino que cuando se desea que finalice se le retira la alimentación. La lectura del sensor se realiza a lo largo de toda la simulación y con repeticiones lo suficientemente cercanas como para que no sea útil el apagado y reinicio continuado de la comunicación. Por tanto no se hará uso del mensaje de finalización de la comunicación a lo largo del programa que controla el robot. En cambio, si que se utiliza la condición de parada a bajo nivel (introducido en la Figura 4.36) para parar las transmisiones de las tramas de petición y recepción indicadas en el punto anterior.

B

Manual del programador

En este apartado se incluye el código utilizado en las funciones clave del proyecto.

B.1. Código utilizado para desarrollar el CPG

La carpeta con el código del proyecto está dividida en numerosas subcarpetas. Cada una de estas subcarpetas incluye distintas simulaciones que han sido utilizadas a lo largo del desarrollo del CPG ya que la realización del mismo se ha ejecutado de manera incremental (las carpetas tienen una numeración, siendo la carpeta número 1 la primera que se realizó en la ejecución del proyecto). En todas estas carpetas hay un lanzador (un script bash) que se encarga de ir lanzando los diversos ejecutables o llamadas a programas de manera secuencial. En el siguiente diagrama se representa uno de los lanzadores (en este caso es el utilizado para calcular el desfase mostrado en el CPG entre los 8 módulos) que se han utilizado:

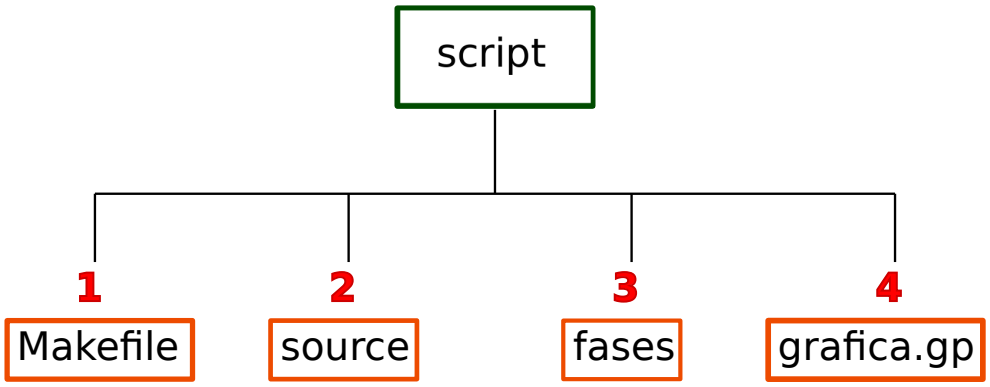


Figura B.1: Estructura de `script`, mostrando los pasos que ejecuta

Los pasos que realiza por tanto el Script en este caso (siendo muy parecido en el resto de subcarpetas) es:

1. **Makefile** Hace una llamada a la utilidad MAKE a través del archivo Makefile para limpiar el directorio de posibles simulaciones anteriores y posteriormente compilar el código C de los archivos que definen al CPG.
2. **source** Llama al ejecutable que realiza la simulación del CPG.

3. **fases** Llama al ejecutable que calcula el desfase mostrado entre las motoneuronas existentes en el modelo.
4. **grafica.gp** Llama al programa Gnuplot con el script grafica.gp para generar una gráfica en la que se muestra el valor de los desfases entre las motoneuronas de los módulos del CPG.

Se puede observar como se realiza este procedimiento en el código del archivo `script`:

```
#! /bin/bash

clear

echo "Script que realiza la simulación de ocho pares de neuronas."
echo "1 - Obtenemos los resultados de la simulación en ficheros."
make cleanall
make source
./source

echo "2 - Obtenemos la diferencia de fase entre las 8 motoneuronas."
make fases
./fases

echo "3 - Representamos los resultados."
gnuplot ./grafica.gp

make clean
rm simulacion.txt simulacion_motoneurona.txt
echo "Fin del script"
```

Para poder borrar archivos creados en simulaciones anteriores se utiliza el archivo `Makefile`. Además, con este archivo también se compila el código que forma el CPG y genera el ejecutable. El texto incluido en el archivo `Makefile` es el siguiente:

```
CC = gcc
CFLAGS = -I.
DEPS = source.h
PROGS = source fases

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

source: source.o rulkov.o motoneurona.o
	$(CC) -o source source.c rulkov.c motoneurona.c $(CFLAGS)

fases: fases.o
	$(CC) -o fases fases.c $(CFLAGS)

clean:
	rm -f *.o
cleanall:
	rm -f *.o (PROGS) *.png *.txt *.eps *~
```

El ejecutable que realiza la simulación del CPG, `source`, se detalla tras la presentación del resto de las llamadas que realiza el script ya que es el elemento más complejo y que necesita ser explicado más en detalle. El código fuente que lo define en su práctica totalidad es:

```
#include "source.h"

int main (void)
{
	NEURON neurona[NUM];
	int i=0, j, k;
	FILE *fichero1, *fichero2;
	float g, g1, g2, g_aux;
```

```
float x_untouched [NUM];
float m[NUM/2];

srand(time(NULL));

//Constantes comunes
g = -0.029;
g_aux = g;
g1 = -0.011;
g2 = 0.024;

for (j=0; j<NUM/2; j++)
        m[j] = 0;

printf ("\n");
//Damos valor a las constantes de las neuronas:
for (j=0; j<NUM; j++){
        printf ("Neurona %d:   ", j+1);
        neurona[j] = arrancaNeurona();
        printf ("%f\t%f\n", neurona[j].x, neurona[j].y);
}

//Neurona 0:
neurona[0].num_entradas = 2;
neurona[0].entrada[0] = 1;
neurona[0].entrada[1] = 2;
//Neurona 1:
neurona[1].num_entradas = 2;
neurona[1].entrada[0] = 0;
neurona[1].entrada[1] = 3;
//Neurona 2:
neurona[2].num_entradas = 3;
neurona[2].entrada[0] = 0;
neurona[2].entrada[1] = 3;
neurona[2].entrada[2] = 4;
//Neurona 3:
neurona[3].num_entradas = 3;
neurona[3].entrada[0] = 1;
neurona[3].entrada[1] = 2;
neurona[3].entrada[2] = 5;
//Neurona 4:
neurona[4].num_entradas = 3;
neurona[4].entrada[0] = 2;
neurona[4].entrada[1] = 5;
neurona[4].entrada[2] = 6;
//Neurona 5:
neurona[5].num_entradas = 3;
neurona[5].entrada[0] = 3;
neurona[5].entrada[1] = 4;
neurona[5].entrada[2] = 7;
//Neurona 6:
neurona[6].num_entradas = 3;
neurona[6].entrada[0] = 4;
neurona[6].entrada[1] = 7;
neurona[6].entrada[2] = 8;
//Neurona 7:
neurona[7].num_entradas = 3;
neurona[7].entrada[0] = 5;
neurona[7].entrada[1] = 6;
neurona[7].entrada[2] = 9;
//Neurona 8:
neurona[8].num_entradas = 3;
neurona[8].entrada[0] = 6;
neurona[8].entrada[1] = 9;
neurona[8].entrada[2] = 10;
//Neurona 9:
```

```
neurona[9].num_entradas = 3;
neurona[9].entrada[0] = 7;
neurona[9].entrada[1] = 8;
neurona[9].entrada[2] = 11;
//Neurona 10:
neurona[10].num_entradas = 3;
neurona[10].entrada[0] = 8;
neurona[10].entrada[1] = 11;
neurona[10].entrada[2] = 12;
//Neurona 11:
neurona[11].num_entradas = 3;
neurona[11].entrada[0] = 9;
neurona[11].entrada[1] = 10;
neurona[11].entrada[2] = 13;
//Neurona 12:
neurona[12].num_entradas = 3;
neurona[12].entrada[0] = 10;
neurona[12].entrada[1] = 13;
neurona[12].entrada[2] = 14;
//Neurona 13:
neurona[13].num_entradas = 3;
neurona[13].entrada[0] = 11;
neurona[13].entrada[1] = 12;
neurona[13].entrada[2] = 15;
//Neurona 14:
neurona[14].num_entradas = 2;
neurona[14].entrada[0] = 12;
neurona[14].entrada[1] = 15;
//Neurona 15:
neurona[15].num_entradas = 2;
neurona[15].entrada[0] = 13;
neurona[15].entrada[1] = 14;

//Abrimos el archivo donde vamos a poner los datos de la simulacin
fichero1 = fopen("simulacion.txt", "w");
if (!fichero1){
        printf ("Error en la apertura del archivo.\n");
        return -1;
}

fichero2 = fopen("simulacion_motoneurona.txt", "w");
if (!fichero2){
        printf ("Error en la apertura del archivo.\n");
        return -1;
}

fprintf (fichero1, "%d", i);
fprintf (fichero2, "%d", i);
for (j=0; j<NUM; j++){
        fprintf (fichero1, "\t%f\t%f", neurona[j].x, neurona[j].y);
        if (j<NUM/2)
                fprintf (fichero2, "\t%f", m[j]);
}
i++;

for (i; i<TAM; i++){
        fprintf (fichero1, "\n%d", i);
        fprintf (fichero2, "\n%d", i);
        for (j=0; j<NUM; j++)
                x_untouched[j] = neurona[j].x;

        for (j=0; j<NUM; j++){

                //Calculamos In
                neurona[j].In = 0.17*neurona[j].In;
                for (k=0; k<neurona[j].num_entradas; k++){
```

```
                                if ((j==0 && neurona[j].entrada[k]==2) || (j==1
                                    && neurona[j].entrada[k]==3) || (j==2 &&
                                    neurona[j].entrada[k]==4) || (j==3 && neurona[
                                    j].entrada[k]==5) || (j==4 && neurona[j].
                                    entrada[k]==6) || (j==5 && neurona[j].entrada[
                                    k]==7) || (j==6 && neurona[j].entrada[k]==8)
                                    || (j==7 && neurona[j].entrada[k]==9) || (j==8
                                     && neurona[j].entrada[k]==10) || (j==9 &&
                                    neurona[j].entrada[k]==11) || (j==10 &&
                                    neurona[j].entrada[k]==12) || (j==11 &&
                                    neurona[j].entrada[k]==13) || (j==12 &&
                                    neurona[j].entrada[k]==14) || (j==13 &&
                                    neurona[j].entrada[k]==15))
                                        g = g2;
                                else if ((j==2 && neurona[j].entrada[k]==0) || (j
                                    ==3 && neurona[j].entrada[k]==1) || (j==4 &&
                                    neurona[j].entrada[k]==2) || (j==5 && neurona[
                                    j].entrada[k]==3) || (j==6 && neurona[j].
                                    entrada[k]==4) || (j==7 && neurona[j].entrada[
                                    k]==5) || (j==8 && neurona[j].entrada[k]==6)
                                    || (j==9 && neurona[j].entrada[k]==7) || (j
                                    ==10 && neurona[j].entrada[k]==8) || (j==11 &&
                                     neurona[j].entrada[k]==9) || (j==12 &&
                                    neurona[j].entrada[k]==10) || (j==13 &&
                                    neurona[j].entrada[k]==11) || (j==14 &&
                                    neurona[j].entrada[k]==12) || (j==15 &&
                                    neurona[j].entrada[k]==13))
                                        g = g1;
                                else
                                        g = g_aux;

                                neurona[j].In = neurona[j].In + g*(-neurona[j].x
                                    + x_untouched[neurona[j].entrada[k]]);
                        }

                        //Calculamos el potencial de la membrana (x) y la
                            variable de dinámica lenta (y)
                        calculo_xy (&neurona[j].x, &neurona[j].y, neurona[j].In);
                }

                for (j=0, k=0; j<NUM/2; j++){
                        m[j] = f_m(neurona[k].x, neurona[k+1].x, m[j]);
                        k = k + 2;
                        fprintf (fichero2, "\t%f", m[j]);
                }
        }

        fclose(fichero1);
        fclose(fichero2);
        return 0;
}
```

Tras la llamada a `source` se pasa a calcular el desfase entre las motoneuronas del CPG utilizando el algoritmo diseñado mediante el ejecutable `fases`. Este cálculo lo realiza obteniendo los datos de un fichero generado por el ejecutable anterior (introducido en la Figura X-DIAGRAMA SOURCE). El código C fuente que define a este ejecutable es el siguiente:

```
#include <stdio.h>

#define NUM_MOTONEURONAS        8
#define CAD                             100

int main()
{
        FILE *f[NUM_MOTONEURONAS];
        int contador, j;
```

```
char archivos[NUM_MOTONEURONAS][CAD] = {"simulacion_motoneurona.txt", "
    fases12.txt", "fases23.txt", "fases34.txt", "fases45.txt", "fases56.
    txt", "fases67.txt", "fases78.txt"};
int i[NUM_MOTONEURONAS], inicio[NUM_MOTONEURONAS], contador_pre[
    NUM_MOTONEURONAS], longitud[NUM_MOTONEURONAS];
float dato[NUM_MOTONEURONAS], dato_pre[NUM_MOTONEURONAS];
int a1_x[NUM_MOTONEURONAS-1];
float diferencia1_x[NUM_MOTONEURONAS-1], desfasaje1_x[NUM_MOTONEURONAS
    -1], desfase;

for (j=0; j<NUM_MOTONEURONAS; j++){
        i[j] = 0;
        inicio[j] = 1;
}

for (j=0; j<NUM_MOTONEURONAS-1; j++)
        a1_x[j] = 0;

//Abrimos los archivos
for (j=0; j<NUM_MOTONEURONAS; j++){
        if (j==0)
                f[j] = fopen(archivos[j], "r");
        else
                f[j] = fopen(archivos[j], "w");

        if (!f[j]){
                printf ("Error en la apertura del archivo f%d.\n", j);
                return -1;
        }
}

fscanf (f[0], "%d ", &contador);
for (j=0; j<NUM_MOTONEURONAS; j++)
        fscanf (f[0], "\t%f", &dato[j]);

for (j=0; j<NUM_MOTONEURONAS; j++)
        dato_pre[j] = dato[j];

while (!feof(f[0])){

        if (dato_pre[0]<=0 && dato[0]>0){
                i[0]++;
                if (inicio[0] == 0){
                        longitud[0] = contador - contador_pre[0];

                        for (j=1; j<NUM_MOTONEURONAS; j++){
                                if (inicio[j] == 0){
                                        diferencia1_x[j-1] = contador_pre
                                            [j] - contador_pre[0];
                                        desfasaje1_x[j-1] = 2*(
                                            diferencia1_x[j-1]/longitud
                                            [0]);
                                        if (j==1){
                                                desfase = desfasaje1_x[j
                                                    -1];
                                                if (desfase > 1)
                                                        desfase = desfase
                                                            - 2;
                                                if (desfase < -1)
                                                        desfase = desfase
                                                            + 2;
                                                else
                                                        desfase = desfase
                                                            ;

                                                fprintf (f[j], "%d %f\n",
```

```
                                                        a1_x[j-1]++, desfase)
                                                    ;
                                            }
                                            else{
                                                    desfase = desfasaje1_x[j
                                                        -1]-desfasaje1_x[j-2];
                                                    if (desfase > 1)
                                                            desfase = desfase
                                                                - 2;
                                                    if (desfase < -1)
                                                            desfase = desfase
                                                                + 2;
                                                    else
                                                            desfase = desfase
                                                                ;

                                                    fprintf (f[j],"%d %f\n",
                                                        a1_x[j-1]++,desfase);
                                            }
                                    }
                            }
                    }
                    else
                            inicio[0] = 0;

                    contador_pre[0] = contador;
            }

            for (j=1; j<NUM_MOTONEURONAS; j++){
                    if (dato_pre[j]<=0 && dato[j]>0){
                            i[j]++;
                            if (inicio[j] == 0)
                                    longitud[j] = contador - contador_pre[j];
                            else
                                    inicio[j] = 0;

                            contador_pre[j] = contador;
                    }
            }

            for (j=0; j<NUM_MOTONEURONAS; j++)
                    dato_pre[j] = dato[j];

            fscanf (f[0], "%d ", &contador);
            for (j=0; j<NUM_MOTONEURONAS; j++)
                    fscanf (f[0], "\t%f", &dato[j]);
    }

    for (j=0; j<NUM_MOTONEURONAS; j++)
            printf ("Numero de pasos por cero de la motoneurona %d: %d\n", j,
                i[j]);

    //Cerramos los archivos
    for (j=0; j<NUM_MOTONEURONAS; j++)
            fclose (f[j]);

    return 0;
}
```

Por último, tras haber realizado todos los cálculos, se pasa a mostrar el resultado de los mismos. Esto se realiza llamando a Gnuplot y pasándole el script `graficas.gp`. El contenido de este script de Gnuplot es el siguiente:

```
#!/usr/bin/gnuplot

#Para incluir acentos
```

```
set encoding iso_8859_1
#Acentuadas: a=\341 e=\351 i=\355 o=\363 u=\372


# Para salida a un archivo tipo portable network graphics
set term postscript eps enhanced color lw 1.5
set output "Simulacion_fases.eps"

unset key
set border linewidth 1
set style line 1 linecolor rgb '#FF0000' linetype 1 linewidth 1.7 # blue
set style line 2 linecolor rgb '#00FF00' linetype 1 linewidth 1.7 # red
set grid

plot [:] [-1.2:1.2] "fases12.txt" using 1:2 title "Diferencia motoneuronas 1-2"
    with lines linewidth 1.5 , \
                        "fases23.txt" using 1:2 title "Diferencia motoneuronas
                            2-3" with lines linewidth 1.5, \
                        "fases34.txt" using 1:2 title "Diferencia motoneuronas
                            3-4" with lines linewidth 1.5, \
                        "fases45.txt" using 1:2 title "Diferencia motoneuronas
                            4-5" with lines linewidth 1.5, \
                        "fases56.txt" using 1:2 title "Diferencia motoneuronas
                            5-6" with lines linewidth 1.5, \
                        "fases67.txt" using 1:2 title "Diferencia motoneuronas
                            6-7" with lines linewidth 1.5, \
                        "fases78.txt" using 1:2 title "Diferencia motoneuronas
                            7-8" with lines linewidth 1.5

# Cierra el archivo de salida
set output
```

Una vez se han listado todos los elementos se pasa a detallar el funcionamiento de la simulación del CPG. Esta simulación está definida por el ejecutable `source` cuyo diagrama de funcionamiento es el siguiente:

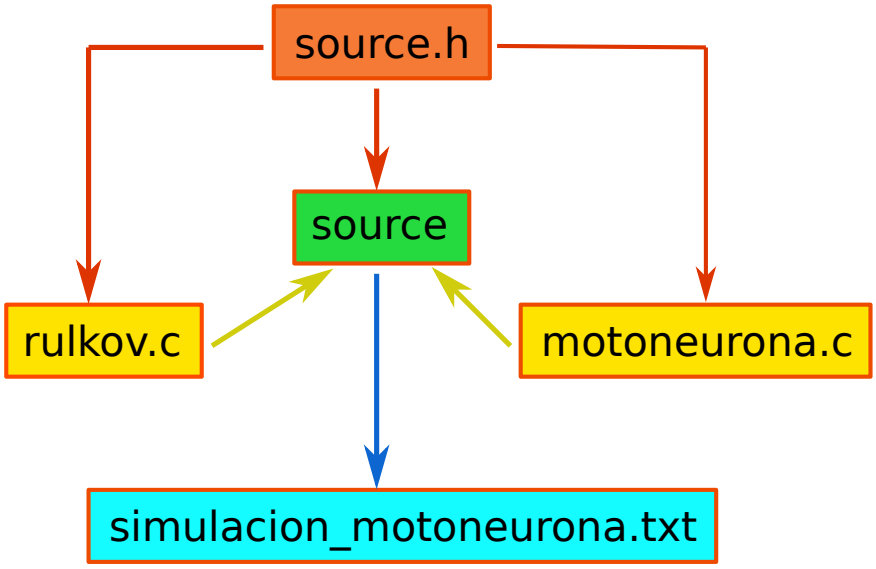


Figura B.2: Estructura de `source`, programa principal de la simulación

Para configurar el CPG se debe editar el fichero `source.h`. De este fichero se nutren los tres ejecutables que se ven implicados en la simulación del CPG: `source`, `rulkov` y `motoneurona`. La estructura del fichero cabecera que define a la simulación es:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define TAM 100000
#define NUM 16          //Nmero de neuronas en la simulacin

//Parmetros neuronas de Rulkov
#define MU              0.001
#define ALFA    15
#define TETA    0.33
#define B_E             1
#define TETA_E  1

//Para la motoneurona
#define ALFA_M          30
#define UMBRAL_M        -1.5
#define TAO_M           1.5
#define CHI                     30

typedef struct{
        float x, y, In;
        int entrada[NUM-1], num_entradas;
}NEURON;

NEURON arrancaNeurona();
void calculo_xy (float *x, float*y, float In);
float f (float x, float y);

float f_m(float Rx, float Px, float m);
int f_s (float x);
```

El ejecutable `source` ya fue descrito unas páginas atrás mediante su código fuente. El código que define en esta simulación al modelo neuronal desarrollado por Rulkov, el fichero `rulkov.c`, es el siguiente:

```
#include "source.h"

NEURON arrancaNeurona()
{
        NEURON neurona_temp;

        neurona_temp.x = ((float)(-2000 + rand() %(4001)))/1000;
        neurona_temp.y = ((float)(-2000 + rand() %(4001)))/1000;
        neurona_temp.In = 0;

        return neurona_temp;
}

void calculo_xy (float *x, float *y, float In)
{
        float xn, yn;

        xn = *x;
        yn = *y;

        *x = f(xn, yn+B_E*In);
        *y = yn - MU*(xn+1) + MU*TETA + MU*TETA_E*In;

        return;
}

float f (float x, float y)
{
        if (x <= 0)
                return (ALFA/(1-x)) + y;
```

```
        else if (0<=x && x<(ALFA+y))
                return (ALFA + y);

        else
                return -1;
}
```

También se ve implicado en la simulación el fichero `motoneurona.c`, que es el fichero donde se encuentra la definición de la función que controla a las motoneuronas de cada uno de los módulos. Si se desea cambiar el modelo de motoneurona en la simulación, solo se debería cambiar a la función incluida en este fichero. El código fuente que contiene es el siguiente:

```
#include "source.h"

float f_m(float Rx, float Px, float m)
{
        float m_nuevo, c;
        int i;

        c = (f_s(Px) - f_s(Rx));

        m_nuevo = m + (ALFA_M*MU/TAO_M)*(-m + CHI*c);

        return m_nuevo;
}

int f_s (float x)
{
        if (x > UMBRAL_M)
                return 1;
        else
                return 0;
}
```

B.2. Código utilizado para integrar el CPG en el simulador RoboSim

Para probar el CPG se utiliza el simulador desarrollado por Fernando Herrero, RoboSim. El código utilizado en este simulador es el siguiente:

El fichero `Codigo_ROBOSIM_seno.c` que muestra el funcionamiento del robot en el simulador al utilizar señales sinusoidales para generar el movimiento.

```
#include "CPG.h"
#include <cmath>
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>

#define FIFONAME "/home/tomas/fifo_CPG"

//Funcion que construye el objeto CPG
CPG::CPG()
{
  _t = 0;
}

//Funcion que destruye el objeto CPG
```

```
CPG::~CPG()
{

}

//Funcion que calcula el siguiente paso del objeto CPG
void CPG::step(InputVector::iterator begin, InputVector::iterator end)
{
  InputVector::iterator module;
  const double phase_difference = M_PI / 2; //Desfase entre modulos
  int counter = 0;

  for(module = begin; module != end; module++){
      *module = 3 * sin(2 * M_PI * _t + counter * phase_difference);
      counter++;
    }

  _t += 0.01;
}
```

El fichero `Codigo_ROBOSIM_fichero.c` que muestra el funcionamiento del robot en el simulador al obtener de un fichero las posiciones de los servos que han sido el resultado del CPG que se ha simulado de manera previa a este programa.

```
#include "CPG.h"
#include <cmath>
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>

//Funcion que construye el objeto CPG
CPG::CPG()
{
        _t = 0;
        _archivo = fopen("/home/XX/RoboSim-0.1.1-Source/sim.txt", "r");
        if (!_archivo)
                printf ("Error en apertura archivo.\n");
}

//Funcion que destruye el objeto CPG
CPG::~CPG()
{
        fclose(_archivo);
}

//Funcion que calcula el siguiente paso del objeto CPG
void CPG::step(InputVector::iterator begin, InputVector::iterator end)
{
  InputVector::iterator module;
  float datos[8], contador;
  int counter = 0, i, n;

        n = read(fdx, buf, sizeof(buf));
        sscanf(buf, "%f %f %f %f %f %f %f %f", &datos[0], &datos[1], &datos[2], &
            datos[3], &datos[4], &datos[5], &datos[6], &datos[7]);
        for(module = begin; module != end; module++){
                *module = datos[counter]/10;
                counter++;
        }
        _t += 0.01;

        for(i=0;i<12;i++)
                n = read(fdx, buf, sizeof(buf));
}
```

El fichero `Codigo_ROBOSIM_tuberia.c` que muestra el funcionamiento del robot en el simulador al obtener de una tubería los valores que tiene que asignar a los servos y que estan siendo simulados en el CPG al mismo tiempo en el que se esta visualizando el resultado en RoboSim.

```
#include "CPG.h"
#include <cmath>
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>

#define FIFONAME "/home/tomas/fifo_CPG"

//Funcion que construye el objeto CPG
CPG::CPG()
{
        _t = 0;

        //Quitamos cualquier tuberia anterior
        unlink(FIFONAME);

        //Creamos la nueva tuberia
        if (mkfifo(FIFONAME, 0666) < 0){
                perror("mkfifo");
                exit(1);
        }

        //Abrimos la tubería para escribir en ella
        fdx = open(FIFONAME, O_RDONLY);
        if (fdx < 0){
                perror("open");
                exit(1);
        }
}

//Funcion que destruye el objeto CPG
CPG::~CPG()
{
        close(fdx);
}

//Funcion que calcula el siguiente paso del objeto CPG
void CPG::step(InputVector::iterator begin, InputVector::iterator end)
{
        InputVector::iterator module;
        float datos[8], contador;
        int counter = 0, i, n;

        n = read(fdx, datos, sizeof(datos)+1);
        for(module = begin; module != end; module++){
                *module = datos[counter]/10;
                counter++;
        }
        _t += 0.01;

        for(i=0;i<12;i++)
                n = read(fdx, datos, sizeof(datos)+1);
}
```

B.3. Código utilizado para desarrollar la comunicación I2C con la que comunicar Robot y Nariz

Tras introducir la forma en la que se simula el CPG, se va a incluir el funcionamiento del robot y la nariz mediante los programas que se han desarrollado para que realicen las funciones deseadas. El diagrama que define este sistema es el siguiente:

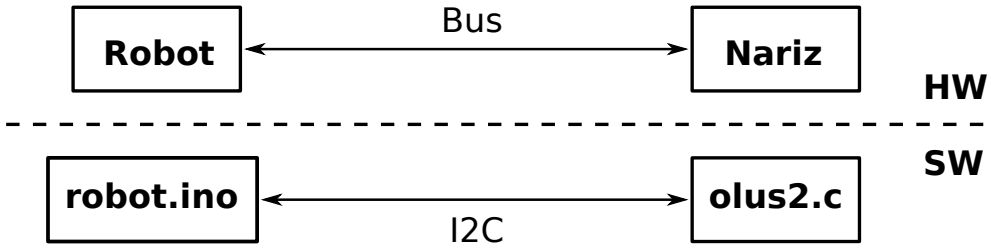


Figura B.3: Relación entre HW y SW al comunicar Robot y Nariz Electrónica

Este diagrama se compone de dos bloques de código que son los que definen la totalidad del proyecto realizado, ya que se tratan de los ficheros con los que se programa tanto a SkyMega (el Robot) como a Olus2 (la Nariz Electrónica). El fichero `robot.ino` está escrito en el lenguaje Wiring y se compila e inserta en la placa SkyMega a través del IDE Arduino. Su sintaxis es similar a la del lenguaje C e incluye ciertas bibliotecas creadas expresamente para la plataforma Arduino. El código que contiene este fichero es el siguiente:

```
#include <Servo.h>
#include <Wire.h>

//Parametros simulacion
#define TAM 20000
#define NUM 12  //Nmero de neuronas en la simulacin
#define NUM_MOTONEURONAS 8
#define CAD 100
#define CENTRO 90

//Parametros neuronas Rulkov
#define MU 0.002
#define ALFA 15
#define TETA 0.33
#define B_E 1
#define TETA_E 1

//Parametros motoneurona
#define ALFA_M 30
#define UMBRAL_M -1.5
#define TAO_M 1.6
#define CHI 30

//Parametros comunicacion I2C
#define NUM_BYTES_RX 12
#define NUM_BYTES_DATO 3
#define NUM_VALORES_ALGORITMO 10
#define DIRECCION_ESCLAVO 0x50

//Variables para controlar el ROBOT
Servo myservo[NUM_MOTONEURONAS];  // create servo object to control a servo
unsigned char pos_servo[NUM_MOTONEURONAS];  // variable to store the servo
    position
unsigned char centro_servo[NUM_MOTONEURONAS];

//Variables para controlar el MODELO NEURONAL
float g;
```

```
float g1;
float g2;
float g_aux;
//float g_swap;
float x[NUM];
float xn;
float x_untouched[NUM];
float y[NUM];
float yn;
float yx;
float In[NUM];
unsigned int num_entradas[NUM];
unsigned int entrada[NUM][3];
float m[NUM_MOTONEURONAS];
float mn;
int c;
boolean sentido;

//Variables para controlar la COMUNICACIÓN I2C
char trama_rx[NUM_BYTES_RX];
unsigned int num_olor, num_temp, caudal, temcal, puede_cambiar_sentido,
    cont_algoritmo;
float olor_m_previo, olor_m_actual;
float diferencia_previa, diferencia_actual;

//Variables del programa
unsigned int i, j, k;

//FUNCIONES COMPLEMENTARIAS

unsigned int ascii2int(char num[NUM_BYTES_DATO])
{
  unsigned int i, j, valor, resultado, exponente;

  valor = 0;
  resultado = 0;

  for (i=0; i<NUM_BYTES_DATO; i++){
        exponente = 1;
        if (num[i]>=0x30 && num[i]<=0x39)  //Si estamos de '0' a '9'
                valor = num[i] - 0x30;
        else if (num[i]>=0x41 && num[i]<=0x46)  //Si estamos de 'A' a 'F'
                valor = num[i] - 0x41 + 10;

        for (j=NUM_BYTES_DATO-1; j>i; j--)
                exponente *= 16;
        resultado += exponente*valor;
  }

  return resultado;
}

unsigned int leeI2C()
{
  char temp[NUM_BYTES_DATO];

  Wire.requestFrom(DIRECCION_ESCLAVO, NUM_BYTES_RX);  // request 12 bytes from
      slave device 0x50

  i=0;
  while(Wire.available())  // slave may send less than requested
  {
    trama_rx[i] = Wire.read();  // receive a byte as character
    delay(5);
    i++;
  }
```

```
  for (i=0; i<4; i++){
    for (j=0; j<NUM_BYTES_DATO; j++)
      temp[j] = trama_rx[i*3+j];

    switch(i){
      case 0:
        num_olor = ascii2int(temp);
        break;
      case 1:
        num_temp = ascii2int(temp);
        break;
      case 2:
        caudal = ascii2int(temp);
        break;
      case 3:
        temcal = ascii2int(temp);
        break;
      default:
        break;
    }
  }

  return num_olor;
}

int f_s (float x)
{
  if (x > UMBRAL_M)
    return 1;
  else
    return 0;
}

void estableceSentido(boolean opcion)
{
  if (opcion == true){

    g1 = 0.024;
    g2 = -0.011;

    //Parametros para movimiento hacia delante
    x[0] = 8.112353;   //  g1 = 0.024  g2 = -0.011;
    y[0] = -7.121844;
    x[1] = -5.887208;
    y[1] = -7.784068;
    x[2] = 0.119136;
    y[2] = -7.683637;
    x[3] = -5.140301;
    y[3] = -7.145892;
    x[4] = -6.461991;
    y[4] = -8.400140;
    x[5] = 1.027232;
    y[5] = -6.450457;
    x[6] = -5.467537;
    y[6] = -7.317634;
    x[7] = -1.000000;
    y[7] = -7.595460;
    x[8] = -4.004359;
    y[8] = -6.508699;
    x[9] = 7.147984;
    y[9] = -8.419033;
    x[10] = 7.173554;
    y[10] = -7.913090;
    x[11] = -4.160799;
    y[11] = -7.060231;
```

```
  }
  else if (opcion == false){

    g1 = -0.011;
    g2 = 0.024;

    //Parametros para movimiento hacia atras
    x[0] = -1.000000;  //  g1 = -0.011  g2 = 0.024;
    y[0] = -7.973985;
    x[1] = -4.403632;
    y[1] = -7.045348;
    x[2] = -3.611740;
    y[2] = -6.465964;
    x[3] = -1.000000;
    y[3] = -8.391280;
    x[4] = -5.234147;
    y[4] = -7.278301;
    x[5] = 0.052328;
    y[5] = -7.598947;
    x[6] = -6.523196;
    y[6] = -8.512330;
    x[7] = 8.654764;
    y[7] = -6.450986;
    x[8] = 7.708195;
    y[8] = -7.903526;
    x[9] = -5.043962;
    y[9] = -7.085382;
    x[10] = -1.000000;
    y[10] = -7.369161;
    x[11] = -5.991342;
    y[11] = -7.661357;
  }
  return;
}

//Funcion a utilizar si se desea hacer girar al gusano (no utilizada en este
    proyecto)
void giraGusano (int angulo)
{
        return;
}

void setup()
{
  //Constantes comunes
  g = -0.029;
  g1 = 0.024;
  g2 = -0.011;
  g_aux = g;

  for (j=0; j<NUM_MOTONEURONAS; j++)
    m[j] = 0;


  //Conectamos a las neuronas entre si
  //Neurona 0:
  num_entradas[0] = 2;
  entrada[0][0] = 1;
  entrada[0][1] = 2;
  //Neurona 1:
  num_entradas[1] = 2;
  entrada[1][0] = 0;
  entrada[1][1] = 3;
  //Neurona 2:
  num_entradas[2] = 3;
  entrada[2][0] = 0;
```

```
entrada[2][1] = 3;
entrada[2][2] = 4;
//Neurona 3:
num_entradas[3] = 3;
entrada[3][0] = 1;
entrada[3][1] = 2;
entrada[3][2] = 5;
//Neurona 4:
num_entradas[4] = 3;
entrada[4][0] = 2;
entrada[4][1] = 5;
entrada[4][2] = 6;
//Neurona 5:
num_entradas[5] = 3;
entrada[5][0] = 3;
entrada[5][1] = 4;
entrada[5][2] = 7;
//Neurona 6:
num_entradas[6] = 3;
entrada[6][0] = 4;
entrada[6][1] = 7;
entrada[6][2] = 8;
//Neurona 7:
num_entradas[7] = 3;
entrada[7][0] = 5;
entrada[7][1] = 6;
entrada[7][2] = 9;
//Neurona 8:
num_entradas[8] = 3;
entrada[8][0] = 6;
entrada[8][1] = 9;
entrada[8][2] = 10;
//Neurona 9:
num_entradas[9] = 3;
entrada[9][0] = 7;
entrada[9][1] = 8;
entrada[9][2] = 11;
//Neurona 10:
num_entradas[10] = 2;
entrada[10][0] = 8;
entrada[10][1] = 11;
//Neurona 11:
num_entradas[11] = 2;
entrada[11][0] = 9;
entrada[11][1] = 10;

sentido = true;  //Establecemos el movimiento hacia delante al iniciar el
    programa
estableceSentido(sentido);

/***Arrancamos todo lo referente a el ROBOT***/
centro_servo[0] = CENTRO -22;
centro_servo[1] = CENTRO -30;
centro_servo[2] = CENTRO -25;
centro_servo[3] = CENTRO -29;
centro_servo[4] = CENTRO -25;
centro_servo[5] = CENTRO -29;
centro_servo[6] = CENTRO -23;
centro_servo[7] = CENTRO -23;

for (i=0; i<NUM_MOTONEURONAS; i++)
  pos_servo[i] = centro_servo[i];

myservo[0].attach(A0);
myservo[1].attach(6);
myservo[2].attach(2);
```

```
  myservo[3].attach(3);
  myservo[4].attach(7);
  myservo[5].attach(A1);
  myservo[6].attach(4);
  myservo[7].attach(5);

  for (i=0; i<NUM_MOTONEURONAS; i++){
    myservo[i].write(pos_servo[i]);
  }

  /***Arrancamos todo lo referente a la comunicacion I2C***/
  Wire.begin();  // Nos unimos al bus I2C

  for (i=0; i<NUM_BYTES_RX; i++)
    trama_rx[i] = ' ';

  delay(500);
  Wire.beginTransmission(DIRECCION_ESCLAVO);
  Wire.write("0003FF");
  Wire.endTransmission();

  puede_cambiar_sentido = 0;
  olor_m_actual = 0;
  olor_m_previo = 0;
  cont_algoritmo = 0;

  //Inicializamos el contador de programa
  i = 0;
}

//Bucle de funcionamiento del programa
void loop()
{
    for (j=0; j<NUM; j++)
      x_untouched[j] = x[j];

    for (j=0; j<NUM; j++){

      In[j] = In[j]*0.18;
      for (k=0; k<num_entradas[j]; k++){

        if ((j==0 && entrada[j][k]==2) || (j==1 && entrada[j][k]==3) || (j==2 &&
            entrada[j][k]==4) || (j==3 && entrada[j][k]==5) || (j==4 && entrada[j
            ][k]==6) || (j==5 && entrada[j][k]==7) || (j==6 && entrada[j][k]==8)
            || (j==7 && entrada[j][k]==9) || (j==8 && entrada[j][k]==10) || (j==9
           && entrada[j][k]==11))
          g = g2;
        else if ((j==2 && entrada[j][k]==0) || (j==3 && entrada[j][k]==1)  || (j
            ==4 && entrada[j][k]==2) || (j==5 && entrada[j][k]==3) || (j==6 &&
            entrada[j][k]==4) || (j==7 && entrada[j][k]==5) || (j==8 && entrada[j
            ][k]==6) || (j==9 && entrada[j][k]==7) || (j==10 && entrada[j][k]==8)
            || (j==11 && entrada[j][k]==9))
          g = g1;
        else
          g = g_aux;

        In[j] = In[j] + g*(-x[j] + x_untouched[entrada[j][k]]);
      }

      //Calculamos el potencial de la membrana (x) y la variable de dinmica lenta
          (y)
      xn = x[j];
      yn = y[j];
      yx = yn+B_E*In[j];

      if (xn <= 0)  //Calculamos X
```

```
        x[j] = (ALFA/(1-xn)) + yx;
      else if (0<=xn && xn<(ALFA+yx))
        x[j] = (ALFA + yx);
      else
        x[j] = -1;

      y[j] = yn - MU*(xn+1) + MU*TETA + MU*TETA_E*In[j];   //Calculamos Y
    }

    //Calculamos el valor de las motoneuronas
    for (j=0, k=0; j<NUM/2; j++){
      c = (f_s(x[k+1]) - f_s(x[k])); //c = (f_s(Px) - f_s(Rx));
      mn = m[j];
      m[j] = mn + (ALFA_M*MU/TAO_M)*(-mn + CHI*c);
      k = k + 2;
    }

    for (j=0; j<NUM_MOTONEURONAS; j++){
      pos_servo[j] = centro_servo[j] + 1.2*m[j];
      myservo[j].write(pos_servo[j]);
    }

    i++;
    if (i == 90){
      //Ponemos en practica el algoritmo
      olor_m_actual = olor_m_actual + leeI2C();

      if (cont_algoritmo==NUM_VALORES_ALGORITMO-1){
        olor_m_actual = olor_m_actual/10;
        diferencia_actual = olor_m_actual - olor_m_previo;

        Serial.print(olor_m_actual);
        Serial.print("   -   ");
        Serial.print(olor_m_previo);
        Serial.print("   -   ");
        Serial.print(diferencia_actual);
        Serial.print("\n");

        if ((diferencia_actual < 0) && (olor_m_actual <= (olor_m_previo*97/100))
           && (puede_cambiar_sentido == 1)){
          Serial.print ("\nCambio de sentido!!\n");
          sentido = !sentido;
          estableceSentido(sentido);
          puede_cambiar_sentido = 0;
        }

        if ((diferencia_actual > 0) && (olor_m_actual >= (olor_m_previo*97/100))
           && (puede_cambiar_sentido == 0)){
          puede_cambiar_sentido = 1;
        }

        cont_algoritmo = 0;
        olor_m_previo = olor_m_actual;
        olor_m_actual = 0;
        diferencia_previa = diferencia_actual;

      }
      else
        cont_algoritmo++;

      i = 0;
    }
}
```

Por otro lado, el archivo con el que se programa el comportamiento de la Nariz Electrónica es `olus2.c`. Este archivo está escrito en lenguaje C y se crea el fichero en hexadecimal que

introducir en Olus2 mediante el compilador MCC18. El código que se encuentra en el fichero forma parte tanto del trabajo previo que hizo David Yáñez al diseñar la nariz como el realizado en este proyecto para implementar la comunicación I2C. El código que lo forma es el siguiente:

```
#include <p18f4550.h>
#include <timers.h>
#include <pwm.h>
#include <adc.h>
#include <delays.h>
#include <usart.h>
#include <eep.h>
#include <stdlib.h>
#include <i2c.h>

#define OLOR PORTAbits.RA0  //Entrada analógica señal sensor.
#define TEMP PORTAbits.RA1  //Entrada analógica temperatura.
#define LED0 PORTBbits.RB5  //Led 0.
#define LED1 PORTBbits.RB4  //Led 1.
#define LED2 PORTBbits.RB3  //Led 2.
#define LED3 PORTBbits.RB2  //Led 3.
#define RX_TX PORTCbits.RC0  //Rx/Tx=1 Envía Rx/Tx=0 Recibe.
#define TAMBUFF 9  //Tamaño máximo buffer trama RX, TX sin contar con el bloque
    de datos.
#define TAMDATOS 14  //Tamaño máximo buffer de bloque de datos
#define LONFILTRO 15  //Tamaño de muestra para su filtrado.
#define VENTMIN 0x250  //Caudal mínimo ventilador.
#define CALEMAX 0x3FF  //Caudal mínimo ventilador.
#define DIR_I2C 0xA0  //-- Definiciones para I2C
#define MAX_SIZE_I2C_TX 12
#define MAX_SIZE_I2C_RX 6
#define MAX_SIZE_DATO 3

#pragma config PLLDIV = 12  //48MHz/12=4MHz->96MHz.
#pragma config CPUDIV = OSC2_PLL3  //96MHz/3=32MHz.
#pragma config USBDIV = 2  //Fuente de reloj para USB procedente de los 96MHz PLL
     div. por 2.
#pragma config FOSC = HSPLL_HS  //HS oscillator, PLL enabled, HS used b USB.
#pragma config FCMEN = OFF  //Fail-Safe Clock ;onitor disabled.
#pragma config IESO = OFF  //Oscillator Switchover mode disabled.
#pragma config PWRT = OFF  //Power up disabled.
#pragma config BOR = OFF  //Brown-out Reset activado en hardware, SBOREN is
    disabled.
#pragma config BORV = 3  //Brown-out Voltage bits.
#pragma config VREGEN = OFF  //USB Voltage Regulator Enable bit-> Disabled.
#pragma config WDT = OFF  //HW disabled- SW Controlled.
#pragma config WDTPS = 32768  //Watchdog Timer Postscale Select bits. 1:32768
#pragma config MCLRE = OFF  //RE3 input pin disabled; MCLR enabled.
#pragma config CCP2MX = ON  //CCP2 input/output multiplexado por RC1.
#pragma config LPT1OSC = OFF  //Timer 1 configured as digital I/O on Reset.
#pragma config PBADEN = OFF  //PORT<4:0> pins are configured as digital I/O on
    Reset.
#pragma config STVREN = ON  //Stack full/underflow will cause Reset.
#pragma config LVP = OFF  //Single-Supply ICSP enabled.
#pragma config XINST = OFF  //Instruction set extension and indexed addressing
    mode disabled.
#pragma config DEBUG = OFF  //Background debugger disabled, RB6 and RB7
    configured as general purpose I/O pins.
#pragma config CP0 = OFF  //not code-protected.
#pragma config CP1 = OFF  //not code-protected.
#pragma config CP2 = OFF  //not code-protected.
#pragma config CPB = OFF  //not code-protected.
#pragma config CPD = OFF  //not code-protected.
#pragma config WRT0 = OFF  //not write-protected.
#pragma config WRT1 = OFF  //not write-protected.
#pragma config WRT2 = OFF  //not write-protected.
#pragma config WRTB = OFF  //not write-protected.
#pragma config WRTC = OFF  //not write-protected.
```

```
#pragma config WRTD = OFF  //not write-protected.
#pragma config EBTR0 = OFF  //not protected from table reads executed on other
    blocks.
#pragma config EBTR1 = OFF  //not protected from table reads executed on other
    blocks.
#pragma config EBTR2 = OFF  //not protected from table reads executed on other
    blocks.
#pragma config EBTRB = OFF  //not protected from table reads executed on other
    blocks.

unsigned char b_recib=0;  //Bytes recibidos para cuando vamos llenando bufferTX.
unsigned char ok=1;  //Flag: 0/1 Trama RX ha/no ha comenzado ':'.
unsigned char direc=0;  //Direccin del equipo en RAM.
unsigned int caudal=VENTMIN;  //Caudal de aire.
unsigned int temcal=CALEMAX;  //Temperatura de calefacción.
unsigned char bufferRX[TAMBUFF+TAMDATOS];  //Buffer de recepción.
unsigned char bufferTX[TAMBUFF+TAMDATOS];  //Buffer de envío.
//Variables I2C
unsigned char i2c_reg_tx[MAX_SIZE_I2C_TX];
char i2c_reg_rx[MAX_SIZE_I2C_RX];
unsigned char reg_index;  // puntero para recorrer la tabla
unsigned char addr_i2c;


//RAFAGA: Parpadeo LEDs
void rafaga(void)
{
        static int paso=0;
        int velo;

        velo=1030-caudal;
        velo=5+(velo/5);

        Delay10KTCYx(velo);
        if (paso==0)     {LED0=1;LED1=0;LED2=0;LED3=0;}  //Frame1 + Tiempo de
            espera.
        else if (paso==1) {LED0=1;LED1=1;LED2=0;LED3=0;}  //Frame2 + Tiempo de
            espera.
        else if (paso==2) {LED0=0;LED1=1;LED2=0;LED3=0;}  //Frame3 + Tiempo de
            espera.
        else if (paso==3) {LED0=0;LED1=1;LED2=1;LED3=0;}  //Frame4 + Tiempo de
            espera.
        else if (paso==4) {LED0=0;LED1=0;LED2=1;LED3=0;}  //Frame5 + Tiempo de
            espera.
        else if (paso==5) {LED0=0;LED1=0;LED2=1;LED3=1;}  //Frame6 + Tiempo de
            espera.
        else if (paso==6) {LED0=0;LED1=0;LED2=0;LED3=1;}  //Frame7 + Tiempo de
            espera.
        else if (paso==7) {LED0=0;LED1=0;LED2=1;LED3=1;}  //Frame8 + Tiempo de
            espera.
        else if (paso==8) {LED0=0;LED1=0;LED2=1;LED3=0;}  //Frame2 + Tiempo de
            espera.
        else if (paso==9) {LED0=0;LED1=1;LED2=1;LED3=0;}  //Frame3 + Tiempo de
            espera.
        else if (paso==10) {LED0=0;LED1=1;LED2=0;LED3=0;}  //Frame4 + Tiempo de
            espera.
        else if (paso==11) {LED0=1;LED1=1;LED2=0;LED3=0;}  //Frame5 + Tiempo de
            espera.

        paso++;
        if(paso==12)
                paso=0;
}

//ACTUAL(): Actualiza valores en RAM para realizar medida.
int actual(void)
```

```
{
        direc=256*Read_b_eep(0x0000)+Read_b_eep(0x0001);  //Obtiene dirección de
            coms.
        caudal=256*Read_b_eep(0x0002)+Read_b_eep(0x0003);  //Obtiene caudal del
            ventilador.
        temcal=256*Read_b_eep(0x0004)+Read_b_eep(0x0005);  //Obtiene temperatura
            de calefacción.
}

//SENSORES(): Lectura de la señal del olor y la temperatura.
int sensores(char canal)
{
        if(canal==0)  //Canal 0 usado entrada por ADC.
                SetChanADC(ADC_CH0);  //Seal del olor.
        else  //Canal 1 usado entrada por ADC.
                SetChanADC(ADC_CH1);  //Seal del olor.

        ConvertADC();  //Activa conversión.

        while(BusyADC()){}  //Espera actualización.

        return ReadADC();  //Lee y envía valor señal analógica.
}

//INT2ASCII(numero, pos): Devuelve el número ascii de la posición 'pos' de un
    entero pasado a hexadecimal 0,1,2...,D,E,F. pos=0,1,2
unsigned char int2ascii(unsigned int num, unsigned int pos)
{
        unsigned int cont;
        unsigned int resto;
        unsigned int dividendo;
        dividendo=num;
        for(cont=0;cont<=pos;cont++){  //Recorre posiciones
                resto=dividendo%16;
                dividendo=dividendo/16;
        }

        if(resto<10)  //Si estamos de '0' a '9'
                resto+=48;  //0=48, 1=49, 2=50, 3=51...
        else  //Si estamos de 'A' a 'F'
                resto+=55;  //A=65, B=66, C=67, D=68...

        return (unsigned char)resto;
}

//ASCII2INT(array 4 digs): Devuelve número entero con el valor numérico de array
    ascii de cuatro dígitos.
unsigned int ascii2int(unsigned char num[4])
{
        unsigned int result=0, expon=1, cont;

        for(cont=0;cont<4;cont++){  //Recorre los 4 dígitos.
                if(num[cont]<0x3A)  //Si estamos de '0' a '9'
                        result+=expon*(num[cont]-0x30);
                else if(num[cont]>0x40) //Si estamos de 'A' a 'F'
                        result+=expon*(num[cont]-0x37);

                expon*=16;  //incrementa 2^4
        }

        return result;  //Manda valor.
}

unsigned int ascii2int_i2c(char num[MAX_SIZE_DATO])
{
  unsigned int i, j, valor, resultado, exponente;
```

```
  valor = 0;
  resultado = 0;

  for (i=0; i<MAX_SIZE_DATO; i++){
        exponente = 1;
        if (num[i]>=0x30 && num[i]<=0x39)  //Si estamos de '0' a '9'
                valor = num[i] - 0x30;
        else if (num[i]>=0x41 && num[i]<=0x46)  //Si estamos de 'A' a 'F'
                valor = num[i] - 0x41 + 10;

        for (j=MAX_SIZE_DATO-1; j>i; j--)
                exponente *= 16;
        resultado += exponente*valor;
  }

  return resultado;
}

//LRC: Calcula LRC de una trama para ser enviado. Trama sobre la que calcular LRC
    , longitud de ésta.
unsigned int montaLRC(unsigned char *trama, int lon)
{
        int cont;
        unsigned char bytelrc=0;  //Byte de redundancia.

        for(cont=1;cont<lon-4;cont+=2){  //Todos menos ':' y LRCs+CRLFs
                if(trama[cont]<58 && trama[cont+1]<58)  //Convertimos ascii->
                    enteros.
                        bytelrc+=(16*(trama[cont]-48)+(trama[cont+1]-48));
                else if(trama[cont]<58 && trama[cont+1]>64)
                        bytelrc+=(16*(trama[cont]-48)+(trama[cont+1]-55));
                else if(trama[cont]>64 && trama[cont+1]<58)
                        bytelrc+=(16*(trama[cont]-55)+(trama[cont+1]-48));
                else if(trama[cont]>64 && trama[cont+1]>64)
                        bytelrc+=(16*(trama[cont]-55)+(trama[cont+1]-55));
        }

        bytelrc=~bytelrc;  //Complemento a uno.
        bytelrc+=1;  //Completa complemento a 2.

        return bytelrc;
}

//FILTRO: Filtro de la lectura de longitud lon. Promediado de "lon" valores 1<lon
    <64, 0/1 real/normalizado. Canal 0/1 Alta Resolu/Rango.
unsigned int filtro(char lon, char canal)
{
        unsigned char cont;
        unsigned int dato, suma=0;
        float salida;

        for(cont=0;cont<lon;cont++)
                suma+=(unsigned int)sensores(canal);  //Suma valor normalizada
                    para media.

        dato=(suma/lon);  //Calcula media.

        return dato;  //Manda dato.
}

//TRAMATX: Monta trama de envío según función y luego la manda
void tramatx(char fun, unsigned int dato)
{
        unsigned char tamTX=0;  //Tamao de la trama enviada.
        unsigned char lrctx[2], bytelrc;  //Buffer para bloque de dirección.
```

```
        unsigned int cont;
        unsigned int a=35, b=18;

        //LED0=1;LED1=1;LED2=1;LED3=1;  //LED TX

        if(fun=='6'){  //Tenemos peticin de escritura.
                //Monta trama confirmacin de escritura.
                tamTX=17;  //Tamao trama respuesta a escritura.
                for(cont=0;cont<13;cont++)
                        bufferTX[cont]=bufferRX[cont];  //Copia primer tramo de
                            trama.

                bytelrc=montaLRC(bufferTX, 17);  //Construye LRC(Trama y long de
                    trama).
                bufferTX[13]=int2ascii(bytelrc,1);  //LRC0 Redundancia
                    longitudinal 0.
                bufferTX[14]=int2ascii(bytelrc,0);  //LRC1 Redundancia
                    longitudinal 1.
                bufferTX[15]=0x0D;
                bufferTX[16]=0x0A;
        }       //Trama de 17 bytes.

        else if(fun=='3'){  //Tenemos peticin de lectura.
                //Monta trama confirmacin de escritura.
                tamTX=15;
                for(cont=0;cont<5;cont++)
                        bufferTX[cont]=bufferRX[cont];  //Copia primer tramo de
                            trama.

                bufferTX[5]=0x30;  // '0'
                bufferTX[6]=0x32;  // '2'
                for(cont=0;cont<4;cont++)
                        bufferTX[cont+7]=int2ascii(dato,4-cont-1);  //Datos.

                bytelrc=montaLRC(bufferTX, 15);  //Construye LRC(Trama y long de
                    trama).
                bufferTX[11]=int2ascii(bytelrc,1);  //LRC0 Redundancia
                    longitudinal 0.
                bufferTX[12]=int2ascii(bytelrc,0);  //LRC1 Redundancia
                    longitudinal 1.
                bufferTX[13]=0x0D;
                bufferTX[14]=0x0A;
        }  //Trama de 15 bytes.


        RX_TX=1;  //Desactivamos lectura.
        //Delay100TCYx(2);  //Esperamos estabilización.
        while (BusyUSART());  //Bus libre?
        for(cont=0;cont<tamTX; cont++){  //Recorremos buffer TX.
                while (TXSTAbits.TRMT==0);  //Sigue enviando?
                TXREG=bufferTX[cont];  //Envía trama.
        }
        while (TXSTAbits.TRMT==0);  //Sigue enviando?

        //LED0=0;LED1=0;LED2=0;LED3=0;  //LED TX

        RX_TX=0;  //Desactivamos lectura.
}

//OPERA: Realiza operaciones demandadas por el maestro.
void opera(void)
{
        unsigned char dat[4], error=0;  //Dato nivel alto, bajo. No error.
        unsigned int cont, dir, dato;

        if(bufferRX[4]=='3'){  //Si funcin = '3' (lectura)
```

```
            ADCON1 = 0B00001101; //<(* ??? *)>\\  //CH0, CH1, CH2 y CH3.

            for(cont=0;cont<4;cont++)  //Recorre trama para obtener dato.
                    dat[3-cont]=bufferRX[cont+5];
            dir=ascii2int(dat);  //Obtiene dirección.

            switch(dir){     //En func. de dirección != respuesta
                    case 0x0000:dato = direc;  //Dirección de red.
                            break;
                    case 0x0001:dato = caudal;  //Caudal de aire.
                            break;
                    case 0x0002:dato = temcal;  //Temperatura de calefacción.
                            break;
                    case 0x0003:dato = (unsigned int)filtro(LONFILTRO, 0);
                        //Adquiere señal de olor.
                            break;
                    case 0x0004:dato = (unsigned int)filtro(LONFILTRO, 1);
                        //Adquiere señal de temperatura.
                            break;
                    default:        error=1;  //Petición no válida. Error.
                            break;
            }
    }

    else if(bufferRX[4]=='6'){  //Si función = '6' (escritura)
            for(cont=0;cont<4;cont++)  //Recorre trama para obtener dato.
                    dat[3-cont]=bufferRX[cont+15];
            dato=ascii2int(dat);     //Obtiene dato.

            for(cont=0;cont<4;cont++)  //Recorre trama para obtener direcc.
                    dat[3-cont]=bufferRX[cont+5];
            dir=ascii2int(dat);  //Obtiene dirección.

            switch(dir){  //En func. de dirección != respuesta
                    case 0x0000:  //Cambia dirección de comunicaciones.
                            Write_b_eep (2*dir, dato/256);
                            Busy_eep ();  //Guarda dato en EEPROM en
                                direcciones 2*dir, 2*dir+1.
                            Write_b_eep (2*dir+1, dato);
                            Busy_eep ();
                            direc=dato;
                            break;
                    case 0x0001:  //Cambiar caudal de aire.
                            Write_b_eep (2*dir, dato/256);
                            Busy_eep ();  //Guarda dato en EEPROM en
                                direcciones 2*dir, 2*dir+1.
                            Write_b_eep (2*dir+1, dato);
                            Busy_eep ();
                            caudal=dato;
                            break;
                    case 0x0002:  //Cambia temperatura de calefacción.
                            Write_b_eep (2*dir, dato/256);
                            Busy_eep ();  //Guarda dato en EEPROM en
                                direcciones 2*dir, 2*dir+1.
                            Write_b_eep (2*dir+1, dato);
                            Busy_eep ();
                            temcal=dato;
                            break;
                    default:
                            error=1;  //Petición no válida. Error.
                            break;
            }
    }

    if(error==0)  //Si no hay error.
            tramatx(bufferRX[4], dato);  //Genera y manda trama.
```

```
}

//TRAMA_I2C_TX: Construye la trama que se envía por I2C con los valores de olor,
    temp_sensor, caudal y temp_cal
void trama_i2c_tx(void)
{
        unsigned int i, olor, temsens, prueba1, prueba2;

        olor = (unsigned int)filtro(LONFILTRO, 0);
        temsens = (unsigned int)filtro(LONFILTRO, 1);

        for (i=0; i<MAX_SIZE_DATO; i++){
                i2c_reg_tx[i]   = int2ascii(olor, MAX_SIZE_DATO -i -1);
                i2c_reg_tx[i+3]= int2ascii(temsens, MAX_SIZE_DATO -i -1);
                i2c_reg_tx[i+6]= int2ascii(caudal, MAX_SIZE_DATO -i -1);
                i2c_reg_tx[i+9]= int2ascii(temcal, MAX_SIZE_DATO -i -1);
        }
}

//TRAMA_I2C_RX:
void trama_i2c_rx(void)
{
        unsigned int i, temp=0;
        char caudal_rx[MAX_SIZE_DATO], temcal_rx[MAX_SIZE_DATO];

        for (i=0; i<MAX_SIZE_DATO; i++){
                caudal_rx[i] = i2c_reg_rx[i];
                temcal_rx[i] = i2c_reg_rx[i + MAX_SIZE_DATO];
        }

        temp = ascii2int_i2c(caudal_rx);
        if (temp>=0 && temp<=1023)
                caudal = temp;
        temp=0;
        temp = ascii2int_i2c(temcal_rx);
        if (temp>=0 && temp<=1023)
                temcal = temp;

        return;
}

//INTERRUPCIONES: Baja prioridad.
void low_isr(void);

#pragma code low_vector=0x008  //Configura vector interrupción.
void low_interrupt(void)
{
        _asm
                goto low_isr
        _endasm
}
#pragma code

#pragma interruptlow low_isr  //Rutina Interrup. RX USART
void low_isr(void)
{
        unsigned char leo;

        //Interrupcion USART
        if(PIR1bits.RCIF){
                while (BusyUSART());  //Espera bus liberado.
                leo=RCREG;  //Carga nuevo dato.
                bufferRX[b_recib]=leo;  //Carga dato.
                PIR1bits.RCIF=0;  //Apaga flag de interrup.

                if(leo==':'){  //Si es carácter de comienzo
```

```
                ok=0;  //La trama RX va bien.
                b_recib=0;  //Resetea el conteo de bytes.
                bufferRX[2]=0;  //No sabemos direc. equipo.
                bufferRX[0]=':';
        }

        //Si trama comienza con ':' && es su direccin o dir=0 && tamaño
            trama corresponde -> TRAMA OK
        else if(ok==0 && (direc==(16*(bufferRX[1]-0x30)+(bufferRX[2]-0x30
            )) || (16*(bufferRX[1]-0x30)+(bufferRX[2]-0x30))==0) && ((
            b_recib==(TAMBUFF+TAMDATOS-6)-1 && bufferRX[4]=='3') || (
            b_recib==(TAMBUFF+TAMDATOS)-1 && bufferRX[4]=='6'))){
                LED0=1;
                LED1=1;
                LED2=1;
                LED3=1;

                opera();  //Efectúa tratamiento trama.
                ok=1;  //Nueva trama por def. no ok.
                b_recib=0;  //Vacía buffer RX.
        }

        else if(b_recib==(TAMBUFF+TAMDATOS)){  //Buffer lleno y trama no
            ok.
                ok=1;  //Nueva trama por def. no ok.
                b_recib=0;  //Conteo de bytes a 0.
        }

        b_recib++;  //Mueve índice de buffer.
        LED0=0;  //LED RX
}

//Interrrupcion I2C
if (PIR1bits.SSPIF == 1){
        if (SSPSTATbits.S == 1){
                //Estado 1: El maestro quiere que el esclavo le transmita
                    sus registros
                if (SSPSTATbits.D_A == 0 && SSPSTATbits.R_W == 1){
                        SSPCON1bits.CKP = 0;
                        addr_i2c = SSPBUF;

                        trama_i2c_tx();

                        reg_index = 0;
                        SSPBUF =  i2c_reg_tx[reg_index];
                        SSPCON1bits.CKP = 1;

                        //LED0 = 1;
                }

                //Estado 2: El esclavo sigue transmitiendo sus registros
                    al maestro
                if (SSPSTATbits.D_A == 1 && SSPSTATbits.R_W == 1){
                        SSPCON1bits.CKP = 0;

                        reg_index++;
                        if (reg_index >= MAX_SIZE_I2C_TX)
                                reg_index = 0;
                        SSPBUF = i2c_reg_tx[reg_index];
                        SSPCON1bits.CKP = 1;

                        LED1 = 1;
                }

                //Estado 3: El maestro quiere enviar datos al esclavo
                if (SSPSTATbits.D_A == 0 && SSPSTATbits.R_W == 0){
```

```
                                SSPCON1bits.CKP = 0;
                                addr_i2c = SSPBUF;

                                reg_index = 0;
                                SSPCON1bits.CKP = 1;

                                LED2 = 1;
                        }

                        //Estado 4: Recepción de datos
                        if (SSPSTATbits.D_A == 1 && SSPSTATbits.R_W == 0){
                                SSPCON1bits.CKP = 0;
                                i2c_reg_rx[reg_index] = SSPBUF;
                                reg_index++;
                                if (reg_index >= MAX_SIZE_I2C_RX){
                                        trama_i2c_rx();
                                        reg_index = 0;
                                }

                                SSPCON1bits.CKP = 1;

                                LED0 = 1;
                        }
                }
                PIR1bits.SSPIF = 0;
        }
}

//CONFIGURA: Configura PIC
void configura(void)
{
        unsigned int cont;

        TRISA = 0b11111111;  //RA0/AN0 RA1/AN1 entradas (Olor/Temperatura)
        TRISB = 0b00000011;  //Puerto B salida (LED). Como entradas RB0 y RB1 (
            I2C)
        TRISC = 0b10111000;  //Entradas/salidas puerto C.

        //ADC
        OpenADC(ADC_FOSC_4 & ADC_RIGHT_JUST & ADC_0_TAD, ADC_CH0 & ADC_INT_OFF &
            ADC_REF_VDD_VSS, ADC_1ANA);//Abra ADC- Analog canal 0 y 1

        //PWM
        OpenTimer2(T2_PS_1_1);  //Config. timer2, para las PWM
        //Calefaccin
        OpenPWM1(0xFF);  //PWM1 a frecuencia máxima
        SetDCPWM1(CALEMAX);  //Inicialmente Calefaccin PWM1 nula
        //Ventilador
        OpenPWM2(0xFF);  //PWM2 a frecuencia máxima
        SetDCPWM2(VENTMIN);  //Inicialmente Ventilador PWM2 mínimo

        //INTERRUPCIONES
        RCONbits.IPEN = 0;  //Config. int. sin prioridades.
        INTCON = 0b11000000;  //Habilitamos PEIE y GIE (1)
        INTCON3 = 0b00000000;  //Deshabilitamos todas las
        PIE1 = 0b00101000;  //interrupciones menos la RX
        PIE2 = 0b00000000;  //originada por la USART y también por el I2C
        //IPR1bits.RCIP=0;  //Prioridad de RX USART baja.

        //Configuramos el I2C
        OpenI2C (SLAVE_7, SLEW_OFF);
        SSPADD = DIR_I2C;
        SSPCON2bits.SEN  = 1;

        reg_index = 0;
        for (cont=0; cont<MAX_SIZE_I2C_TX; cont++)
```

```
                i2c_reg_tx[cont] = '0';
}

void main ()
{
        unsigned int dato;

        //Configuramos el PIC
        configura();  //Configura periféricos
        actual();  //Toma valores de comienzo de EEPROM

        //Valores iniciales
        caudal = 512;
        temcal = 512;

        while (1){
                //rafaga();

                SetDCPWM1(temcal);  //Calefacción: Inicialmente nula
                SetDCPWM2(caudal);  //Ventilador:  Inicialmente mínimo
        }

        CloseUSART();  //Cerramos USART (Formalismo)
        ClosePWM1();  //Cerramos PWM1 (Formalismo)
        ClosePWM2();  //Cerramos PWM2 (Formalismo)
        CloseI2C();  //Cerramos I2C (Formalismo)
}
```

En este Anexo únicamente se incluye una pequeña parte del total del código generado para la realización de este proyecto. Para una mayor comprensión del funcionamiento de los programas realizados se deben buscar los fuentes en las carpeta de trabajo incluida con el PFC. La estructura del resto de ficheros es similar a la explicada en este Anexo con lo que con los conocimientos adquiridos a lo largo de este capítulo se debe ser capaz de entender el funcionamiento del resto de programas.

C

Presupuesto

1)	**Ejecución Material**	
	▪ Compra de ordenador personal	1.000 €
	▪ Componentes del robot	170 €
	▪ Material de oficina	50 €
	▪ Total de ejecución material	1.220 €
2)	**Gastos generales**	
	▪ 16 % sobre Ejecución Material	196 €
3)	**Beneficio Industrial**	
	▪ 6 % sobre Ejecución Material	74 €
4)	**Honorarios Proyecto**	
	▪ 1000 horas a 15 €/ hora	15000 €
5)	**Material fungible**	
	▪ Gastos de impresión	200 €
	▪ Encuadernación	10 €
6)	**Subtotal del presupuesto**	
	▪ Subtotal Presupuesto	16700 €
7)	**I.V.A. aplicable**	
	▪ 21 % sobre Subtotal Presupuesto	3507 €
8)	**Total presupuesto**	
	▪ Total Presupuesto	20207 €

Madrid, Abril 2013

El Ingeniero Jefe de Proyecto

Fdo.: Tomás Vázquez Rubio

Ingeniero Superior de Telecomunicación

D

Pliego de condiciones

Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un *Integración de una nariz electrónica ultra-portátil en un robot modular para el control de su movimiento a través de los odorantes recibidos*. 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 corresponderá 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 garanta 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 Contratаz 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.