Universidad de la República
Facultad de Ingeniería

RattusBot Rioplatensis

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Nicolás Blanco, Mariana del Castillo, Joaquín Quagliotti

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Dr. Ing. Leonardo Barboni ..... Universidad de la República

Tribunal

Dr. Ing. Fernando Silveira ..... Universidad de la República
Mag. Ing. Rafael Canetti ..... Universidad de la República
Dr. Ing. Leonardo Barboni ..... Universidad de la República

Montevideo
miércoles 30 septiembre, 2015

RattusBot Rioplatensis, Nicolás Blanco, Mariana del Castillo, Joaquín Quagliotti.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 150 páginas.
Compilada el miércoles 30 septiembre, 2015.
http://iie.fing.edu.uy/

As far as the laws of mathematics refer to reality, they are not certain, and as far as they are certain, they do not refer to reality.

Albert Einstein

Cuando las leyes de la matemática se refieren a la realidad, no son ciertas; cuando son ciertas, no se refieren a la realidad.

Albert Einstein

Agradecimientos

Dedicamos esta carilla para agradecer a quienes nos ayudaron en el desarrollo del proyecto. Con temor a olvidarnos de nombrar a alguno, agradecemos a:

- José Luis Vila, por el soporte en las actividades relacionadas al diseño e implementación de la mecánica.
- Leonardo Barboni, por la continua disponibilidad e interés.
- Javier Cuneo, por tener paciencia para explicarnos.
- Nuestras familias y amigos, por el apoyo incondicional.

A todos los componentes que dieron su vida por este proyecto, gracias.

Resumen

El presente documento explica el conjunto de actividades realizadas y resultados obtenidos al concluir el proyecto de fin de carrera. El mismo tuvo como objetivo la construcción de una plataforma robótica, para ser usada como herramienta de investigación por los neurocientíficos que estudian los patrones de disparos de redes neuronales, presentes en la corteza entorrinal. Estas redes son responsables de construir mapas cognitivos del ambiente (i.e. representación o codificación del entorno), los cuales se usan como entradas a otras capas neuronales que implementan la acción de navegación de algunos individuos biológicos (e.g. humanos, ratas).

A la fecha, existe gran interés científico por entender los mecanismos por los cuales estas redes implementan, no solo el procesamiento de información de navegación, sino también el aprendizaje de reglas y creación de sistemas de referencias para tomar decisiones de movimiento en ambientes novedosos.

Estos estudios se abordan modelando las redes como sistemas dinámicos acoplados, descritos con ecuaciones diferenciales. De este modo, fue un neurocientífico que está realizado estudios de doctorado en esta área, junto al grupo de microelectrónica del IIE quien proporcionó los modelos matemáticos discretos de las neuronas llamadas *grid cells* y *place cells* (las cuales componen parte de la corteza entorrinal). Estos serán implementados en una plataforma robótica prototipo, la cual debía ser desarrollada y evaluada, para determinar si era adecuada para el estudio de estas redes, motivando de este modo la realizacion del proyecto de fin de carrera.

Para desarrollar del robot prototipo, se estudió cómo implementar los algoritmos propuestos en la plataforma Arduino (e.g. escalado de constantes de las ecuaciones), así como las necesidades de señales de entrada requeridas del ambiente y cómo obtener las salidas de patrones de disparo que los algoritmos podrían generar, los cuales serían los estudiados por los neurocientíficos.

La plataforma robótica se implementó entonces con dos placas de desarrollo Arduino Due que incorporan un microprocesador Atmel SAM3X8E ARM Cortex-M3. Además, la configuración morfológica final del robot prototipo consiste en:

1. Dos ruedas motorizadas con encoders.
2. Una tercera de apoyo, libre.

3. Cuatro optoacopladores.

4. Una cámara con su correspondiente interfaz de comunicación.

5. Cuatro sensores de choque.

6. Piezas mecánicas de ajuste así como otros circuitos electronicos secundarios que se describirán en el documento.

Los modelos matemáticos en tiempo discreto de las redes neuronales implementadas, reciben como estímulo de entrada la posición absoluta (x, y) del robot. Esta posición absoluta se estima mediante una técnica de odometría. La misma consiste en estimar el desplazamiento del robot a partir de información de giro de las ruedas.

De esta manera, para evaluar la utilidad de esta plataforma robótica, a la hora de estudiar la dinámica de las redes neuronales, se buscó reproducir los experimentos que los neurocientíficos realizan. Estos consisten, en hacer recorrer el robot una arena de espacio acotado, como lo hace en la realidad una rata de laboratorio (en nuestro caso un área de 2×2 m con un sistema de coordenadas cartesiano (x, y) a intervalos de 1 cm). El seguimiento de la posición del robot se realiza mediante odometría y se adquieren los patrones de disparo que salen de las redes neuronales implementadas para cada posición (x, y) mediante un bus SPI.

Como resultado de dicha evaluación, se encontraron varios problemas, entre los cuales el dominante es la limitante en el tiempo de ejecución de los algoritmos. La discretización de las ecuaciones diferenciales que modelan las neuronas se resolvió utilizando el método de Euler, con pasos de 1 ms. Por esta razón todas las operaciones que comprenden el cómputo de las *grid cells* y la comunicación hacia afuera de los *spikes*, deben realizarse en ese intervalo de tiempo.

Teniendo en cuenta esta limitación, para cada posición (x, y) se consiguió implementar patrones de disparos de una red con un máximo de 200 neuronas. El cual se envía al exterior por medio del bus SPI, en menos de 1 ms, La cantidad final de neuronas implementadas, fue lograda gracias a un proceso de optimización que buscaba minimizar el tiempo de ejecución de los cálculos correspondientes a la implementación de los algoritmos que modelan las *grid cells*.

Como se mencionó anteriormente, el robot prototipo tiene dos placas modelo Arduino Due. Una se encarga de las funciones de navegación junto a todos los sensores y actuadores incluidos en la plataforma. Es la encargada de implementar las rutinas que mueven al robot en la arena, y a su vez realiza el seguimiento de la posición del mismo mediante una técnica de odometría. Dicha información es luego enviada a la otra placa Arduino Due quien implementa los algoritmos de redes neuronales.

El robot desarrollado implementa los modelos propuestos dentro de las limitaciones del hardware seleccionado, tanto para la odometría, como para el la im-

plementación de los algoritmos de redes neuronales. Se realizaron pruebas de las diferentes partes que componen a la plataforma completa, para verificar su correcto funcionamiento. También se realizaron pruebas de validación sobre los algoritmos implementados durante las distintas etapas de optimización.

Finalmente, se verificó que el disparo de las células correspondiera a las zonas adecuadas del espacio, para algunas de estas *grid cells*, probando la generacion de mapas cognitivos del ambiente en términos de patrones de disparos de las redes.

Como trabajos futuros, se sugiere profundizar en la optimización de tiempos de las operaciones que implementan los algoritmos neuronales. También se proponen ideas para mejorar el sistema de seguimiento espacial. Por ejemplo, se pueden incorporar marcadores externos para corregir errores de estimación de la posicion absoluta. De este modo se lograría aumentar la vida útil del algoritmo de seguimiento y de la corrida experimental (la vida útil se define en términos de cotas máximas aceptables, para el crecimiento de error en la estimación de la posicion por acumulación de errores durante los movimientos). También se pueden añadir dispositivos que integren encoders en cuadratura u otra clase de motores y engranajes de reducción de velocidad.

Tabla de contenidos

Capítulo 1

Introducción

Este Capítulo consiste en una introducción general al contenido de este proyecto, así como también su desarrollo. Se incluye, una descripción del proyecto, los objetivos generales, el alcance del proyecto, y las restricciones que lo rigen junto a una serie de criterios de éxito para evaluar su resultado.

El presente documento describe los detalles del proyecto RattusBot Riopalatensis, incluyendo los criterios y definiciones tomadas durante el desarrollo del mismo así como las soluciones de hardware y software implementadas. Antes de comenzar con los detalles más técnicos se desarrolla una breve introducción comentando los fundamentos y el marco en que se especificó y realizó dicho proyecto.

1.1. Descripción del Proyecto

El proyecto encuentra su origen en la forma en que los animales determinan su posición espacial relativa. Ciertas regiones del cerebro, contienen diversas neuronas que participan en los intrincados sistemas que permiten, a un individuo, navegar en el espacio conociendo su posición. Algunas de estas células tienen la particularidad de volverse activas únicamente cuando el animal se encuentra en regiones específicas del espacio. Es este comportamiento fuertemente vinculado con el entorno físico, el que últimamente codifica y determina la comprensión de la propia posición, y de la ubicación relativa con respecto a otros lugares. Se identifican dos tipos de células con estas características, las *place cells* y las *grid cells*.

Las *place cells* son neuronas que se encuentran en el hipocampo y su principal característica es que muestran actividad cuando el animal se encuentra en un área particular del espacio. Cada célula de este tipo tiene asociada una única región en el ambiente que se denomina *place field*. Estas zonas se caracterizan por una distribucion de tipo gaussiana que rige la frecuencia de disparo de la neurona en torno al centro del *place field*

Por otro lado, las células sobre las que se centró particularmente la atención de este proyecto se encuentran en la corteza entorrinal medial y se conocen como *grid cells*. Estas neuronas reciben su nombre debido a que muestran actividad formando en el espacio un patrón regular con forma de red triangular, en el que cada nodo de la malla tiene una distribución similar a la de las *place cells*. Cada neurona genera un patrón caracterizado por ciertos parámetros que varían célula en célula dependiendo de diversos factores que serán descritos en profundidad en el Capítulo 4.

Los neurocientíficos estiman que ambos sistemas de neuronas interactúan entre sí, siendo los principales responsables de formar el mapa cognitivo que permite a un animal orientarse en el espacio.

En este contexto, el IUEMHI (Instituto Universitario Escuela de Medicina del

Hospital Italiano – Bs. As.) realiza investigaciones desarrollando modelos de estos sistemas neuronales, a los que inclusive han incorporado conceptos de neurogénesis (i.e. creación de neuronas no entrenadas y por lo tanto de gran plasticidad, que se incorporan a la red para colaborar en situaciones novedosas).

Se planteó entonces desarrollar un robot móvil , que contenga modelos biológicos de *grid cells*, simulados en tiempo real. El objetivo de la plataforma es que pueda ser usada como herramienta para estudiar el comportamiento de estas redes.

1.2. Interesados

Se identificaron durante la especificación del proyecto los siguientes interesados en el proyecto, vinculados con el contexto en que este surge.

- Ing. Biomédico Javier Cuneo (Argentino, IUEMHI , estudiante de Doctorado del Grupo de Microelectrónica , DA, Co-DT Fernando Silveira).
- Dr. Pablo Francisco Argibay (Argentino, IUEMHI , co-tutor de tesis de Doctorado de Javier Cuneo).
- Grupo de Microelectrónica del IIE (FING-UdelaR).

1.3. Planificación

Durante la planificación se fijó el alcance del proyecto, en función de suposiciones equivocadas. Inicialmente se entendió que las neuronas actuarían como sistema de navegación, entregando una salida que permitiese comandar los motores, y tomando como entrada la información de los sensores de la plataforma. Sin embargo esta no era la solución requerida por los interesados del proyecto, la solicitud fue, como se expresa en este documento, que el robot implemente modelos de *grid cells* para utilizar como herramienta de investigación. Se redefinieron entonces el alcance y los objetivos como se presenta a continuación, y son estos, los que se toman como válidos durante el desarrollo del proyecto. La versión original de la planificación y los documentos entregados en los hitos se encuentran adjuntos en los Apéndices B, C y D.

1.3.1. Objetivos

A continuación se describen los objetivos fijados para el proyecto luego de resueltos los problemas en la especificación.

Objetivo General

Desarrollar un robot demostrador que incorpore modelos de sistemas neuronales propuestos por IUEMHI para que pueda someterse a operación en un ambiente real.

Objetivos específicos

- Armar mecánicamente el kit robot y lograr funcionalidades básicas.
- Selección de la plataforma Hardware-Software e implementación de los modelos neuronales en la plataforma elegida.
- Modelar la mayor cantidad posible de neuronas.
- Implementar un sistema de navegación basado en odometría.
- Implementar todo el sistema logrando que interactúen correctamente todas las partes que lo integran.
- Armado de la arena.
- Integración, prueba y depuración del sistema completo.
- Documentación del proyecto.

1.3.2. Alcance

Se definieron como parte del alcance del proyecto los siguientes ítems:

- El robot demostrador se construirá tomando como base el kit "All-Ready Arduino Robot Chassis Kit".
- En esta plataforma se programará:
  - Comando de los motores.
  - Control de sensores y procesado de las señales..
  - Modelado de las *grid cells*: es propuesto por los interesados y los algoritmos son dados.
  - Se construirá un ambiente o arena de 2×2 m con paredes que limiten el área de acción del robot y este podrá tener dibujado patrones de imágenes.

1.3.3. Restricciones

Se plantearon también las siguientes restricciones.

- El proyecto debe finalizar a mas tardar el 30 de Abril del 2015. (Se pidió prórroga de un mes)
- El grupo de proyecto tiene una disponibilidad de 30 hs. semanales (10 hs. de dedicación semanal continua por estudiante durante un año).
- La placa microcontroladora seleccionada deberá ser un Arduino Due (por la capacidad de cálculo requerida estimada para desarrollar la aplicación). En caso de necesidad se podrá incorporar una placa extra como la Arduino Mega (para solucionar problemas de compatibilidad de sensores).
- El robot será alimentado por una fuente de tensión externa y se moverá en una arena con piso liso de $2 \times 2\ m$.

1.3.4. Criterios de éxito

Los indicadores de éxito que se definieron durante la planificación del proyecto son los siguientes.

- Lograr implementar los modelos de redes neuronales propuestos en el robot.
- Satisfacer las necesidades de los investigadores en neurociencias, también actores en este proyecto implementando al menos un modelo de redes neuronales propuestos en el robot.

1.4. Descripción de los siguientes Capítulos

A continuación se presenta en el **Capítulo 2**, una descripción general de la plataforma desarrollada en sus aspectos de hardware y software, así como un panorama general de los modelos biológicos de las neuronas. Luego en el **Capítulo 3** se profundiza la descripción del sistema de navegación y todos sus componentes, detallando cada subsistema en hardware y software. **El Capítulo 4** describe con sumo detalle los aspectos matemáticos de los modelos utilizados para las *grid cells*, su implementación en software y brevemente los aspectos biológicos involucrados. En el **Capítulo 5**, se presentan las distintas interfaces de comunicación utilizadas, tanto para el traspaso de información entre las placas, así como para la comunicación de las mismas con el exterior. Transversalmente el **Capítulo 6** describe las pruebas realizadas para la validación de cada componente del RattusBot, así como del sistema integrado. Se concluirá en el **Capítulo 7** con una evaluación general, algunas conclusiones e ideas para trabajos a futuro ya que la plataforma tiene aún infinidad de aspectos para mejorar.

Capítulo 2

Descripción General

2.1. Aspectos generales

Este Capítulo describe en rasgos generales los componentes y sistemas que integran al RattusBot, incluyendo hardware utilizado y el software implementado. Se exponen los dos bloques principales: el sistema de navegación y el modelado de neuronas. Se comentan también algunas consideraciones generales realizadas durante la planificación.

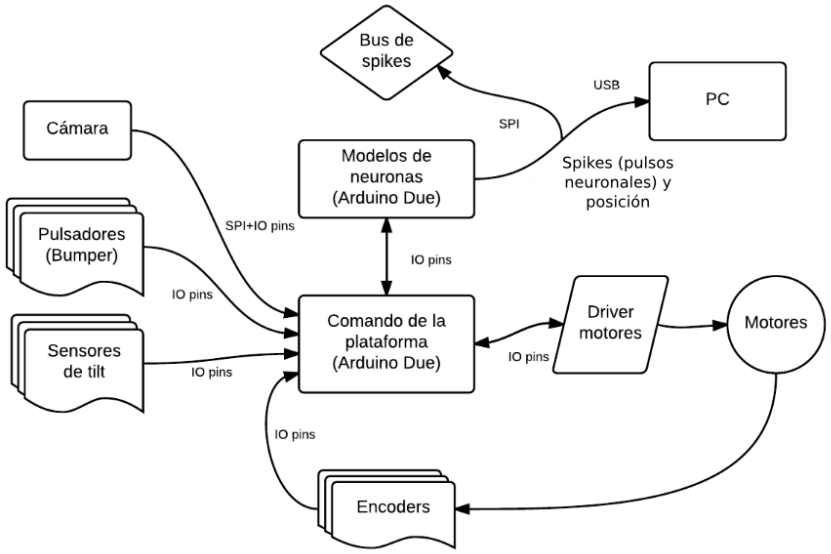


Figura 2.1: Diagrama de bloques del Sistema completo

Previo al desarrollo e implementación de hardware y software se realizó un profundo proceso de estudio y planificación. Debido a que la especificación del proyecto fue bastante abierta en muchos aspectos, fue necesario definir una gran multiplicidad de factores que se dejaron a definir por el equipo.

Desde el punto de vista del hardware se definieron sensores, actuadores y controladores del robot estudiando diferentes opciones en busca de la solución óptima para las funciones deseadas. Particularmente se evaluaron las diferentes alternativas de microcontroladores a utilizar en función de sus prestaciones y de los requisitos del sistema a implementar. El punto crítico del problema es la simulación de los modelos biológicos de las *grid cells*. Esto es debido a que, como se presentará más adelante, el cómputo de los estados de las células debe realizarse en pasos de tiempo muy cortos, lo que implica la necesidad de contar con un procesador potente que pueda realizar los cálculos que implica procesar una neurona en el menor tiempo posible.

Desde el punto de vista del software, se identificaron algunas de las tareas que debe realizar el robot y se evaluaron diferentes arquitecturas para resolver de for-

ma óptima los procesos necesarios para el funcionamiento del robot.

Se implementó entonces el sistema según el diagrama de bloques presentado en la Figura 2.1. El robot cuenta con dos microcontroladores interconectados, uno encargado de determinar el comportamiento del robot y otro que implementa los modelos de *grid cells*. Posee también diversos tipos de sensores incluyendo una cámara que le permite identificar el ambiente, sensores de tilt y bumper que utiliza para navegar por él y encoders que le permiten rastrear su propio movimiento. La plataforma es propulsada por dos motores DC controlados, utilizando un driver tipo puente H. Finalmente, todas las partes del robot se interconectan con múltiples medios de comunicación, contando ademas con la posibilidad de reportar hacia afuera (por ejemplo a una PC) los comportamientos de las neuronas modeladas y la posición de la plataforma.

2.1.1. Arduino

Para los microcontroladores se definió utilizar placas de desarrollo de la familia Arduino debido a que provee plataformas en una amplia gama, con gran portabilidad entre estas. Inicialmente se planificó utilizar una placa Arduino Mega para el comando de la plataforma y una Arduino Due para el modelado de neuronas.

La Arduino Mega cuenta con un procesador de 16 MHz basado en la misma arquitectura de 16 *bits* que la mayoría de las placas de la familia Arduino. Se utilizó esta placa durante toda la etapa inicial de estudio de sensores, algoritmos básicos de funcionamiento y comando de motores. Sin embargo, a la hora de estudiar el funcionamiento de la cámara (incluida entre los sensores enumerados anteriormente), se identificaron múltiples problemas de compatibilidad. Un ejemplo es que la revisión de la placa Arduino Mega con la que se contaba, no incluye el pin $I_{o_{ref}}$. El mismo es utilizado por ambos shiels de Arducam para auto-controlar el nivel lógico de las salidas, puesto que estan previstos para trabajar tanto con sistemas de 5 V (Familia Arduino estándar), como con 3.3 V que es el voltaje de trabajo del procesador de la Arduino Due. Otro ejemplo es la incompatibilidad de niveles lógicos entre la Mega y la DUE que complica los vínculos de comunicación entre ambas placas. Por estas razones se definió finalmente utilizar dos microcontroladores Arduino Due.

La placa de desarrollo Arduino Due, está basada en el procesador de Atmel, SAM3X8E ARM Cortex-M3 [5]. Este procesador, utiliza una arquitectura de 32 *bits*, y un reloj 84 MHz. Es la placa más potente de la familia Arduino y se consideró por esto la mejor opción para el modelado neuronal. Si bien las librerías incluidas en el IDE de Arduino presentan una gran compatibilidad entre estas dos plataformas, pues manejan las interacciones de hardware por medio de *macros* condicionales, para algunas funcionalidades específicas se requirieron modificaciones a los código previamente implementados.

Para la implementación del software se eligió utilizar el IDE de Arduino, por su simpleza, disponibilidad de librerías, así como también su capacidad de compatibilidad entre las distintas plataformas Arduino. El IDE utiliza un lenguaje propio basado en C y C++ mezclado con *macros* del fabricante que facilitan en algunos aspectos la programación. A su vez Arduino provee una librería básica que facilita la interacción con hardware.

Dado que ninguno de los integrantes del equipo estaba familiarizado con los sistemas Arduino, fue necesario un estudio completo de las herramientas provistas, así como de los detalles de los procesadores utilizados. Fue necesario también profundizar los conocimientos sobre el lenguaje C, ya que hasta el momento se contaba únicamente con conocimientos a un nivel rudimentario.

A continuación se describen los elementos y subsistemas que comprenden al RattusBot. Los bloques principales como el sistema de navegación y los modelos neuronales se describen en forma esquemática ya que se profundiza sobre ellos en los Capítulos siguientes.

2.2. Sistema de navegación

El sistema de navegación es el bloque que interpreta la información de los sensores y en función de la información adquirida define el comportamiento subsiguiente del robot. Se considera parte de este subsistema el software del microcontrolador que controla a la plataforma así como el siguiente hardware:

- Una placa Arduino Due.
- Dos acrílicos perforados para el soporte mecánico de las partes.
- Dos motores DC, con sus correspondientes ruedas.
- Una rueda libre.
- Dos discos ranuradas para la odometría.
- Dos sensores de tilt.
- Dos microswitches.
- Un parachoque de acrilico.
- Cuatro optoacopladores para la odometría [10].
- Un driver de motores basado en el integrado L298.
- Un conversor de nivel de cuatro canales, bilateral.

- Un conversor de nivel de dos canales tipo UART.
- Dos PCB para leer el estado de los tilts.
- Cuatro PCB para adecuar los encoders.
- Un PCB para facilitar el conexionado de las alimentaciones correspondientes.

Muchos de los componentes que integran el hardware fueron fabricados de manera "artesanal" durante el transcurso del proyecto con los materiales que fue posible adquirir. Se construyó con la ayuda de José Vila (I.E.M.) la plataforma de acrílico de dos niveles, de modo de aprovechar el espacio para colocar todos los componentes. Se realizó también la fabricación, montaje y ajuste del un bumper frontal con detectores de choque, así como los soportes para los sensores de tilt. Se montaron cuatro leds para mejorar la imagen obtenida a través de la cámara.

Se realizaron los PCB correspondientes a las adaptaciones de los sensores, así como de una barra de alimentación de 5 V, 3.3 V y GND, para facilitar el conexionado de todos los componentes.

La Figura 2.2 muestra al RattusBot en su ensamble final, indicando los elementos principales que lo integran.

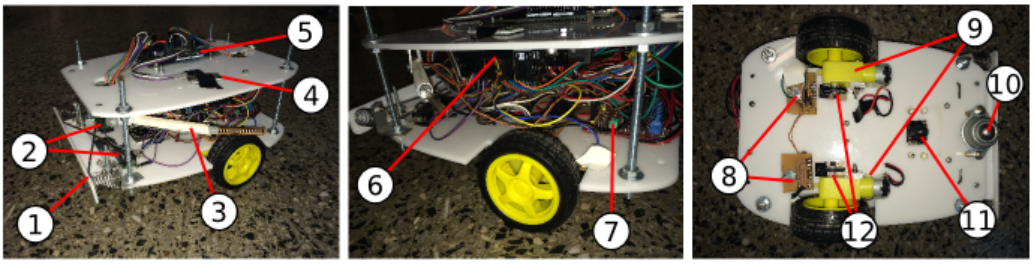


Figura 2.2: 1: Parachoque, 2: Microswitches, 3: Sensor de tilt, 4: Módulo Bluetooth, 5: Arduino Due (Alta), 6: Arduino Due (Baja), 7: Driver Motores, 8: PCB + encoder, 9: Ruedas motorizadas, 10: Rueda "libre", 11: Cámara, 12: Rueda ranurada para odometría

Para el microcontrolador asociado a este sistema se implementó una arquitectura de Round-Robin con interrupciones, con el fin de manejar cómodamente la multiplicidad de sensores que comprenden a la plataforma. En este se implementaron cuatro subsistemas que serán descritos brevemente a continuación.

2.2.1. Odometría

Se entiende por odometría las técnicas que utilizan la información de sensores para determinar la posición relativa de un robot durante su desplazamiento.

Para el RattusBot se colocaron discos ranurados en los ejes de las ruedas que permiten rastrear la rotación de las mismas utilizando optoacopladores (el conjunto disco-optoacoplador se denomina encoder). A partir de esta información se puede determinar el desplazamiento de la plataforma y así su posición relativa al punto de partida.

Durante la especificación del proyecto se planteó que el RattusBot debe navegar en una arena de 2×2 m con paredes que delimitan su frontera. La utilización de odometría para especificar la posicion del robot implica entonces que su punto de partida en cada experimento realizado sea siempre el mismo. Esto es debido a que el método solo mide desplazamientos relativos por lo que se debe fijar un punto de referencia para que la posicion sea absoluta (en este caso el punto de partida). Cambiar el punto inicial de partida implica una reprogramación de la placa Arduino.

En la Sección 3.1 se dedica exclusivamente a profundizar sobre este tema, describiendo los algoritmos utilizados y el hardware asociado a este sistema.

2.2.2. Sensores

Otro componente importante del robot son los múltiples sensores que se instalaron en la plataforma. Estos le permiten interactuar con su entorno, detectando por ejemplo los límites de la arena en la que se desplaza.

Cámara

La cámara cumple la función de identificar el ambiente en que el robot se desplaza. La necesidad de identificar diferentes ambientes surge del fundamento del proyecto, que es simular el comportamiento de una rata en un ambiente confinado. Diferentes ambientes implican diferentes mapa cognitivos asociados lo que se traduce en diferentes comportamientos para las *grid cells*. Para comprender como los ambientes afectan el comportamiento de las neuronas es necesario profundizar en los modelos que se presentarán en el Capítulo 4. Se utiliza entonces la cámara para identificar distintos ambiente" para los que se simulan comportamientos diferentes de las neuronas. La identificación se realiza por medio de tarjetas con diferentes figuras, que se colocan debajo del robot en el punto de partida. Estas tarjetas son fotografiadas por la cámara y procesadas por un algoritmo que las identifica, y asocia a ellas un determinado ambiente. En la Sección 3.2.1 se profundiza sobre los detalles de este algoritmo. Se simularon cuatro posibles entornos para que el robot se desplace. La única influencia que tienen sobre el RattusBot es en el modelo de las *grid cells* y no en el comportamiento de la plataforma.

Bumper

Se instaló en el frente del robot un parachoques de acrílico, que cumple la función de amortiguar el efecto de las colisiones frontales del robot, utilizando dos resortes de acero. Además se instalaron 2 microswitches, que son accionados por palancas de alambre al mover el bumper, de modo que se puede detectar si la colisión es totalmente frontal, frontal izquierda ó frontal derecha.

Sensores de tilt

Los sensores de tilt son resistencias variables por flexión, montadas sobre un soporte de acrílico que se colocan como "bigotes" (emulando a la musa del proyecto, la rata), y se utilizan para detectar la proximidad de las paredes de la arena.

2.2.3. Motores

Como se menciona anteriormente el RattusBot se desplaza por el entorno gracias a dos motores instalados a los lados de la plataforma. Los motores son de corriente continua, con imanes permanentes y son controlados utilizando un driver tipo "Dual Full-Bridge" (L298). Este recibe las señales de control enviadas por el microcontrolador y alimenta a los motores de acuerdo a estas, para así obtener el comportamiento deseado.

La elección de utilizar motores de continua se realizó fundamentada por la gran disponibilidad que se tuvo de ellos al inicio de proyecto. Al momento de planteada la especificación inicial, no parecía ser necesario utilizar otro tipo de motores y dado el bajo costo y pensando en el bajo presupuesto que se acordó, fueron en ese momento una buena elección. Sin embargo, las características de este tipo de motores los convirtió en una pésima elección, causante de más de un dolor de cabeza.

El Capitulo 3 está dedicado exclusivamente al sistema de navegación y detalla la arquitectura y la lógica de control del driver

2.2.4. Comportamiento

El movimiento del robot se especificó como aleatorio, por lo que se implementó una rutina que comanda al RattusBot para que realice una trayectoria errática al azar. Sin embargo no todo el movimiento es caótico, ya que a la hora de atender situaciones, como por ejemplo, choques o la actuación de los "bigotes", se debe tener un criterio de modo que el desplazamiento resuelva el incidente. Por ejemplo ante un choque el robot, este se mueve de modo que el obstáculo no quede nuevamente frente a la plataforma una vez maniobrado.

2.3. Modelos neuronales

Otro subsistema fundamental del robot es el modelado de las *grid cells*. Este no afecta directamente al comportamiento del robot, pero si utiliza información adquirida por el sistema de navegación para funcionar. Concretamente los modelos requieren dos elementos;

1. Identificar el ambiente en el cual se encuentra el robot.
2. Conocer la posición del robot en tiempo real.

El Elemento 1 se requiere, ya que como se mencionó en la Sección 2.2.2, para diferentes entornos las neuronas se comportan de modo diferente. Por otro lado el Elemento 2 es necesario, ya que como se menciona en el Capítulo 1, las *grid cells* son neuronas cuyo comportamiento está asociado a la posición espacial.

2.3.1. Simulación de neuronas

En el Capítulo 4, se explican en profundidad los modelos matemáticos que se utilizan para simular el comportamiento eléctrico de las células, así como el vínculo entre la posición y la actividad de las mismas. A modo de introducción y para comprender el comportamiento eléctrico de una neurona, se hace referencia a uno de los modelos biológico más sencillos llamado *integrate and fire*. Como se puede apreciar en la Figura 2.3, se modela a la célula como un condensador de capacidad C_m que se carga con una corriente de excitación (I_{ex}). A medida que se carga el condensador la tensión en bornes aumenta, hasta que al alcanzar un determinado nivel que se denomina voltaje umbral (V_{th}), luego es descargado generando un pulso tipo delta en la salida, que se denomina *spike*. Es facil ver que ante una mayor corriente de excitación, se obtiene una mayor frecuencia de *spikes* y viceversa. Además, si la excitación no es suficiente, el capacitor se descarga por la resistencia y la tensión cae alejándose del umbral.

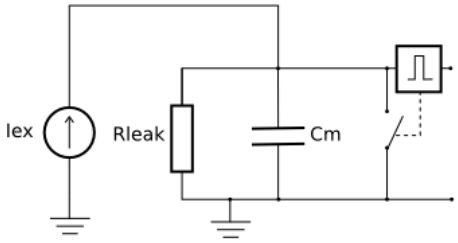


Figura 2.3: Modelo de *Integrate and fire*

El modelo eléctrico implementado para el RattusBot es más complejo, y propone una mayor precisión desde el punto de vista biológico. Se continúa profundizando sobre estos modelos en el Capítulo 4. Hasta aquí se pretende únicamente

que se llegue a comprender el comportamiento de una neurona, para lo que el *integrate and fire* es más que suficiente.

Yendo al caso específico de las *grid cells*, como se presentó en el Capítulo 1, estas tienen actividad en puntos específicos del espacio formando, una malla triangular en la que las zonas con mayor cantidad de *spikes* coinciden con los vértices de los triángulos. Si por ejemplo, se considera el modelo *integrate and fire* para la neurona, para igualar el comportamiento de una *grid cell*, se debe inyectar mayor corriente de excitación en los nodos de la malla que en el resto del espacio. Esto se logra modelando dicha corriente con una ecuación matemática de comportamiento adecuado. Esta función se define a partir de parámetros que son diferentes para cada neurona y las caracterizan de manera individual. Además de una influencia del ambiente que modifica a la ecuación de igual manera para todas las *grid cells*. Dado que esta función tiene como variable la posición (x, y), se puede comprender la razón de que la placa que modela las *grid cells* requiera de esta información para funcionar.

2.4. Comunicación

Hasta este punto se ha descrito el comportamiento independiente del sistema de navegación y del modelado neuronal. Se pueden identificar entonces dos vías de comunicación imprescindibles para que el robot funcione correctamente. La primera es la comunicación entre los dos sistemas. Este vínculo permite a los modelos contar con la información que requieren para ejecutarse. La segunda es entre el robot y el mundo exterior. Para que de esta última obtengamos alguna utilidad, debe ser posible extraer los datos de las simulaciones de los modelos, ya sea para analizar o para inyectar como entrada a otros sistemas.

Se describen brevemente a continuación estos vínculos, aunque este tema se abordan con mayor detalle en el Capítulo 5.

2.4.1. Posición

Para este enlace de comunicación se utiliza un sistema en paralelo de 2 *bytes* más un *bit* para sincronizar la lectura de los datos. La posición del robot se codifica como un par (x, y) de enteros sin signo de un byte. De este modo se pueden transmitir valores de x e y con precisión de 1 cm tomando valores entre 0 y 255. Se escogió este protocolo debido a la necesidad de realizar esta transferencia en el menor tiempo posible. Para esto se utiliza un *byte* de un puerto de 32 *bits* del microcontrolador, que puede ser leído en una única instrucción minimizando así el tiempo de lectura que es un recurso esencial para el modelado de las neuronas. Utilizando el pin de sincronización, el sistema de navegación le indica al de modelado cuando hay una nueva posición para leer en el puerto. Con esto se logra

evitar que se lean del puerto datos inválidos.

2.4.2. Spikes

Como se menciona anteriormente, ya sea para ser analizados o para ser inyectados en un sistema externo, es necesario reportar el estado de los modelos hacia el exterior. Para esto se establecen dos vías de comunicación posibles. La primera y más simplemente accesible es un puerto serial por bluetooth, que permite adquirir los datos de los *spikes* (o cualquier otro que se desee siempre y cuando se programe adecuadamente el microcontrolador). Esta vía está limitada a una velocidad óptima de 115200 *bps*, que no permite extraer los *spikes* de todas las células en tiempo real pero si un número limitado de las mismas. La segunda es un puerto SPI, que es capaz de alcanzar una velocidad de transferencia mucho mayor pero no es tan amigable como el bluetooth a la hora de ser utilizado.

2.5. Otras Consideraciones

Durante el desarrollo de la plataforma se definió que todas las piezas que integran al RattusBot deben ser reemplazables, dentro de lo posible. Se tomó esto en cuenta, debido a que el robot se concibe como una herramienta de investigación, por lo que su vida útil debe exceder el desarrollo del proyecto. Previendo entonces la posibilidad de que alguno de los componentes se dañe, es necesario que se pueda reemplazar la parte descompuesta sin la necesidad de cortar o soldar para retirar las piezas. De este modo no se corre el riesgo de provocar mayor daño a la plataforma al intentar repararla.

Conectores

Se utilizaron conectores tipo *pinhead* para todas las interconecciones entre placas y sensores. Los PCB implementados se fabricaron teniendo esto en cuenta, por lo que todos cuentan con este tipo de conector.

Alimentación

La alimentación del robot se resolvió utilizando una fuente de computadora tipo ATX. Siguiendo el mismo criterio en cuanto a la intercambiabilidad, para extraer las tensiones deseadas para el RattusBot se fabricó un cable con un conector tipo *molex* de cuatro pines, de modo que la fuente pueda ser fácilmente reemplazada. Se cableó la salida de 12 V para alimentar, tanto los motores como una de las placas. A través de la placa de navegación se alimenta la DUE Alta y los distintos sensores que componen al sistema. Para la conexión al robot se utilizaron 2 *plugs* de alimentación que se conectan a la Arduino asociada al sistema de navegación y a una entrada que alimenta los motores.

Cabe destacar, que durante las experimentaciones iniciales, se quemó el regulador del driver de motores (que otorgaba una salida en 5 V), por lo que no fue posible utilizar dicha funcionalidad. A partir de ese momento, se procedió a alimentar siempre de forma paralela, el driver y las Arduino.

Con esto se han cubierto de manera superficial todos los aspectos que hacen al funcionamiento del RattusBot, incluyendo hardware y software. Con esta información se pretende dar al lector una visión general del funcionamiento de la plataforma, para que pueda abordar los siguientes Capítulos en los que se describe de forma más aislada cada subsistema.

Capítulo 3

Sistema de Navegación

En este Capítulo se profundiza en los temas que hacen al sistema de navegación. Este bloque comprende el conjunto de hardware y software que permite al RattusBot desplazarse por el entorno, e interactuar con él. Se describen los componentes que integran este conjunto, incluyendo las consideraciones e implementaciones realizadas.

3.1. Odometría

El sistema de odometría cumple la función de rastrear la posición del robot en la arena, a partir del ángulo de giro de las ruedas. Para implementar esta función se optó por utilizar encoders rotativos, que permiten rastrear el ángulo de rotación de las ruedas y así determinar el desplazamiento de la plataforma entre dos puntos. Este sistema presenta múltiples inconvenientes ya que es adecuado sólo para pequeños arcos de desplazamiento, y no es válido ante deslizamientos de las ruedas ni sobre superficies irregulares. Esto se debe a que dichas condiciones invalidan las hipótesis del modelo utilizado.

El modelo se toma del libro "Sensors for Mobile Robots, Theory and Application" de H. R Everett [8] (páginas 49 a 54) y es vastamente utilizado para robots de dos ruedas con tracción diferencial. Este tipo de robots, como el que se desarrolla en el proyecto, utiliza la diferencia entre las velocidades de las ruedas para comandar la dirección del movimiento. Es decir que, al contar con un eje fijo para las ruedas, los giros se logran mediante la imposición de velocidades distintas para cada motor. El algoritmo modela sencillamente al sistema, definiendo un eje de coordenadas en el plano.

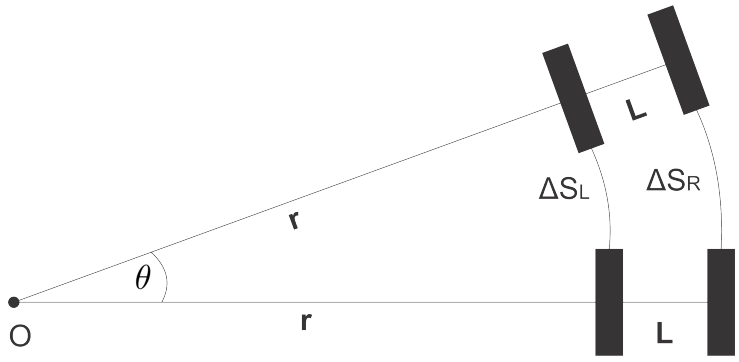


Figura 3.1: Representación geométrica y variables para la navegación odométrica

Las ecuaciones clásicas definen que, para una rueda girando sin deslizar, el desplazamiento del centro de la misma está dado por:

$$x = R\varphi$$

Con R radio de la rueda y φ el ángulo de giro de la misma.

Tomando C_e como la cantidad de pulsos por vuelta del encoder, podemos definir C_m, como el factor de conversión del movimiento angular en desplazamiento lineal del eje del motor en el sentido del giro.

$$\Delta\varphi = \frac{2\pi N}{C_e}$$

Donde N coresponde al número de pulsos generados por el encoder cuando la rueda gira un ángulo

$$\Delta\varphi$$

.

$$C_m = \frac{2\pi R}{C_e} \tag{3.1}$$

$$\Delta S_L = C_m N_L \tag{3.2}$$

$$\Delta S_R = C_m N_R \tag{3.3}$$

Por medio de la Ecuación (3.1) obtenemos expresiones para los desplazamientos lineales de cada rueda. Donde N_L y N_R representan la cantidad de pulsos medidos para cada rueda durante la trayectoria que se está evaluando. A partir de los desplazamientos lineales y suponiendo que el recorrido del robot para un instante i, es un arco de circunferencia diferencial, el desplazamiento de cada rueda genera un arco en torno al punto O. En la Figura 3.1 se puede ver que para la rueda izquierda el radio de curvatura es r y para la rueda derecha el mismo es $r + L$.

En torno a estas suposiciones, tenemos entonces un sistema de dos ecuaciones con dos incógnitas. Operando obtenemos expresiones para $\Delta\theta$ y r.

$$\Delta\theta_i = \frac{\Delta S_{L_i}}{r} \tag{3.4}$$

$$\Delta\theta_i = \frac{\Delta S_{R_i}}{r + L} \tag{3.5}$$

$$\Delta S_{L_i} = r\Delta\theta_i \tag{3.6}$$
$$\Delta S_{R_i} = (r + L)\Delta\theta_i \tag{3.7}$$

Se pueden deducir entonces las siguientes ecuaciones:

$$\Delta\theta_i = \frac{\Delta S_{R_i} - \Delta S_{L_i}}{L} \tag{3.8}$$

$$r_i = \frac{L\Delta S_{L_i}}{\Delta S_{R_i} - \Delta S_{L_i}} \tag{3.9}$$

El desplazamiento del centro del eje se calcula a partir de las Ecuaciones (3.8) y (3.9) como:

$$\begin{aligned} \Delta S_i &= \Delta\theta_i(r_i + \frac{L}{2}) \\ &= (\frac{\Delta S_{R_i} - \Delta_{L_i}}{L})(\frac{L\Delta S_{L_i}}{\Delta S_{R_i} - \Delta S_{L_i}} + \frac{L}{2}) \\ &= \frac{\Delta S_{R_i} + \Delta S_{L_i}}{2} \end{aligned} \tag{3.10}$$

Las Ecuaciones (3.8) y (3.10) permiten calcular entonces las variaciones en el ángulo θ correspondiente al que forma el eje de las ruedas con la horizontal y el desplazamiento del centro del eje a partir del desplazamiento lineal de cada rueda.

Se tiene entonces a partir de las ecuaciones anteriores por medio de igualdades trigonométricas, el algoritmo utilizado, donde:

- R = Radio de las ruedas.
- L = Distancia entre centros de las ruedas.
- C_e = Resolución del encoder.
- N_{R_i} = Cantidad de pulsos contados en la rueda derecha.
- N_{L_i} = Cantidad de pulsos contados en la rueda izquierda.

$$\begin{aligned} C_m &= \frac{2\pi R}{C_e} \\ \Delta\theta_i &= C_m \frac{N_{R_i} - N_{L_i}}{L} \\ \Delta S_i &= C_m \frac{N_{R_i} + N_{L_i}}{2} \\ \theta_i &= \theta_{i-1} + \Delta\theta_i \\ x_i &= x_{i-1} + \Delta S_i \cos\theta_i \\ y_i &= y_{i-1} + \Delta S_i \sin\theta_i \end{aligned} \tag{3.11}$$

Error e Incertidumbre

El anterior modelo presenta en la práctica deficiencias en su función de establecer la posición absoluta del robot. Durante las pruebas realizadas sobre el sistema, se constató que este, tiene un tiempo acotado de funcionamiento. Entonces la diferencia entre la posición real y la calculada comienza a ser muy significativa y el robot deja de comportarse como se espera. La causa de este problema es la acumulación sucesiva de errores durante la operación, así como también la presencia de incertidumbres en algunas de sus variables. Las siguientes listas enumeran dichos factores, agrupándolos en dos categorías, según sean fuentes de errores sistemáticos, o no.

1. Fuentes de errores sistemáticos:
   - Diferencia en el radio de las ruedas.
   - Promedio entre el radio de las ruedas difiere del valor nominal del radio de cada una de ellas.
   - Ruedas desalineadas.
   - Incertidumbre en la distancia medida entre los puntos de apoyo (debida a deformaciones en las cubiertas).
   - Resolución de los encoders finita y limitada.
   - Limitaciones en la velocidad de muestreo de las señales de los encoders.
2. Fuentes de errores no sistemáticos:
   - Movimientos sobre suelos desparejos.
   - Deslizamientos de las ruedas, debidas a:
     - Suelos resbaladizos.
     - Aceleraciones desmedidas.

- Giros rápidos.
- Fuerzas externas (choques fuertes contra objetos).
- Fuerzas internas (debidas al apoyo en la rueda libre).
- Pérdida del contacto de alguna de las ruedas con el suelo.

El artículo "Measurement and Correction of Systematic Odometry Errors in Mobile Robots" [7] de Johann Borenstein y Liqiang Feng, aborda el problema de la medida y corrección de este tipo de errores en sistemas como el implementado. Según exponen estos autores, de las múltiples fuentes de error que afectan a un sistema con dos ruedas controladas con odometría, hay dos que son dominantes respecto al resto. La primera es que los radios de ambas ruedas no son exactamente iguales, introduciendo diferencias tanto en el desplazamiento lineal como en los giros. Debido a que las ruedas son recubiertas de goma para mejorar la tracción es muy difícil fabricarlas exactamente iguales. Además debido a las diferencias en la fuerza a la que son sometidas las ruedas su radio varía, incluso durante el funcionamiento del robot. Al ser diferentes sus radios, un pulso del encoder puede significar diferentes desplazamientos lineales según de que rueda se trate, resultando en que las trayectorias rectas se releven como curvas.

La segunda fuente dominante es el deslizamiento de las ruedas, esto hace que la correlación entre el desplazamiento lineal de la rueda y el ángulo de giro medido por los encoders cambie. Esta fuente genera efectos similares a los anteriores en las trayectorias rectilíneas y es principalmente significativo en los ángulos de giro. La Figura 3.2 muestra el efecto del error de estas dos fuentes en una trayectoria programada. En verde se representa un recorrido programado formando un cuadrado de 4×4 m; en rojo se presenta el efecto del error por deslizamiento que modifica los ángulos de giro de 90° programados a 87°; en violeta se muestra la curvatura de las trayectorias rectilíneas causada por la diferencia en los radios de las ruedas.

Como se puede observar estas fuentes de error perturban la correlación entre la posición real del robot y la calculada, rotando y trasladando las coordenadas de referencia. Es claro que un error muy pequeño en el ángulo de giro produce diferencias apreciables cuanto más lejos esté el robot del punto de rotación. A causa de la acumulación de estas diferencias y dado que estas nunca son corregidas ni compensadas, el sistema tiene una "vida útil" limitada para cada corrida de la plataforma, que oscila entre 2 y 5 minutos dependiendo fuertemente de la superficie en la que se desplaza.

El artículo propone medidas para minimizar el efecto de estas fuentes de error. Desafortunadamente por limitantes en cuanto a la duración de este proyecto, no fue posible tenerlo en cuenta para el producto final.

Para reducir el error por deslizamiento se buscó que la plataforma realizará movimientos lentos y se experimentó con distintos suelos. Sin embargo, resultó que el caso en el que se dan menos deslizamientos los motores presentan dificultades para cambiar el sentido de giro. Es decir, para reducir deslizamientos, se disminuyó la

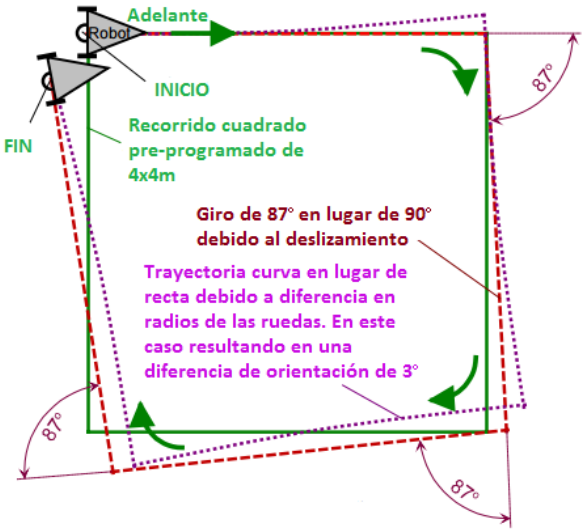


Figura 3.2: Efectos de errores en un circuito cuadrado unidireccional (Adaptado de [7])

velocidad de trabajo de los motores. Estos a su vez, a bajas velocidades (menor alimentación), presentan dificultades para iniciar el movimiento.

El problema se debe a que para comenzar a moverse, los motores deben vencer el torque producido por la fuerza de rozamiento del eje del motor, producido por el rozamiento estático cuando la plataforma está quieta. Cuando está en movimiento el par mínimo para mantener este funcionamiento es el generado por el rozamiento dinámico con el eje, y el excedente genera la tracción que propulsa la plataforma hacia adelante. Este par mínimo es menor al de arranque y por esta razón los motores deben vencer un torque mayor cuando se inicia el movimiento que cuando el eje ya está girando. Por otro lado, la relación entre el par (C) y la velocidad (n) de un motor de continua es como se muestra en la Figura 3.3, donde C_0 y n_0 son como se plantea en las Ecuaciones (3.12) y (3.13)

$$C_0 = \frac{V\phi k_b}{2\pi R_m} \tag{3.12}$$

$$n_0 = \frac{V}{\phi k_b} \tag{3.13}$$

Si se analiza la Figura 3.3 y las Ecuaciones (3.12) y (3.13), se puede observar que la pendiente de la curva depende del voltaje de alimentación V, la resistencia del motor R_m, la constante de la máquina k_b y el flujo magnético ϕ. Si se varía la alimentación, para un determinado par resistente, la curva se mueve y se pueden lograr distintas velocidades, sin embargo modificando la tensión cambia también

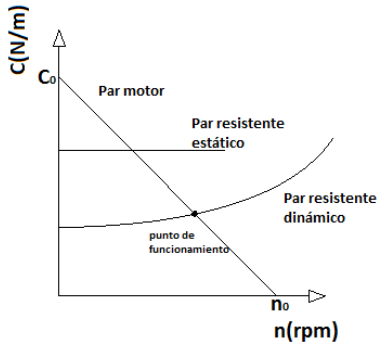


Figura 3.3: Curva C-n de un motor DC

el par C_0.

Al momento del arranque el par de la máquina es C_0 y para que el motor inicie su movimiento este debe ser mayor que el par resistente, en este caso el producido por el rozamiento estático del eje. Entonces se debe fijar V para cumplir esta condición ya que ϕ no puede ser variado, porque los motores utilizados son de imanes permanentes. Cuando se vence el rozamiento estático, el torque baja y la velocidad en el punto de operación queda determinada por la tensión de alimentación y el par resistente. Como no se puede controlar la pendiente de la curva sin controlar ϕ, solo se puede imponer la velocidad resultante y el C_0 a la vez. En el caso de los motores utilizados la velocidad aumenta rápidamente luego de iniciado el movimiento, por lo que se debe trabajar lo más cerca posible del límite para lograr velocidades bajas y reducir los deslizamientos. Esto produce problemas ya que a causa de las condiciones de la arena, el par resistente puede aumentar provocando que el robot se detenga y sea necesario "darle un empujón" para que retome su movimiento.

Este problema puede ser resuelto o bien cambiando el tipo de motor o colocando una caja de reducción que aumente el par a costa de velocidad de giro. Dado que los motores fueron especificados en la planificación del proyecto, que los recursos económicos fueron limitados y que el tiempo dedicado al sistema de navegación excedió lo planificado, no fue posible evaluar alternativas para este problema. Por esta razón simplemente se trabajó lo más cerca posible del límite de funcionamiento, dejando una holgura para que la plataforma se mueva con fluidez.

3.1.1. Encoders

Los encoders son los componentes principales del sistema de odometría. . Estos son los que permiten rastrear este ángulo de giro de las ruedas y así implementar el algoritmo descrito en la Sección 3.1. Los encoders son discos ranurados que interrumpen al emisor de un optoacoplador. Se puede entonces obtener una señal pulsada en el receptor que está en V_{CC} cuando el disco está interrumpiendo al optoacoplador y en GND cuando no lo está (es decir, pasa la luz). La Figura 3.4

muestra un esquema del optoacoplador y el disco que conforman un encoder.

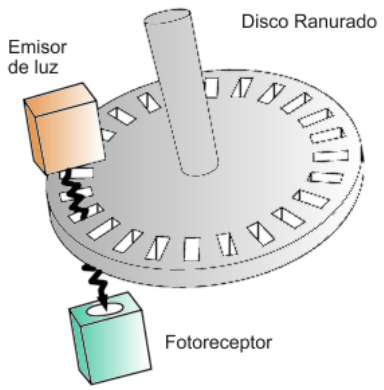


Figura 3.4: Esquema de encoder

Para la adaptación de la señal de salida del optoacoplador se fabricó el PCB correspondiente al circuito de la Figura 3.5 donde $R_1 = 1\ k\Omega$ y $R_2 = 200\ \Omega$. En la Figura 3.6 se muestra esquemáticamente el encapsulado utilizado [10].

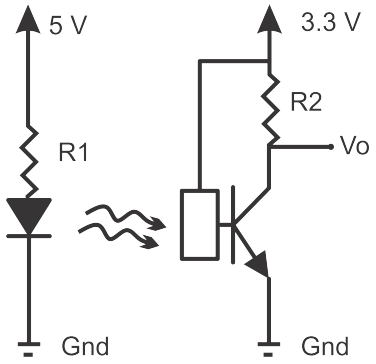


Figura 3.5: Esquemático del PCB del optoacoplador Versión 1

Observando las Figuras 3.5 y 3.6 se puede deducir que cuando la luz llega del emisor al receptor, la tensión en la salida es la de tierra y cuando no lo hace, la resistencia de *pull-up* R_1 mantiene la tension en V_{CC}. La resistencia R_2 impone una corriente de 20 mA en el emisor para lograr un comportamiento adecuado del componente. La señal obtenida permite observar el ángulo de rotación de los ejes de los motores y por lo tanto de cada rueda. Sin embargo la implementación de este sistema presenta ciertos problemas. Principalmente se observó que si la rueda se detiene de modo que el borde de una ranura quede interrumpiendo parcialmente al optoacoplador, se generan pulsos ante la más mínima variación en la vibración

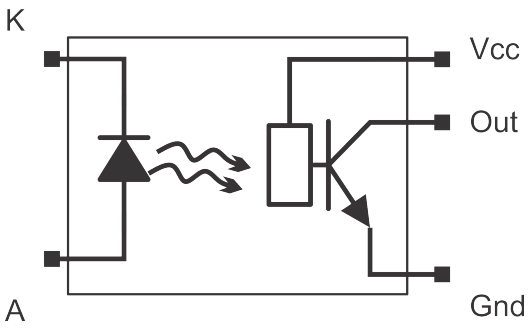


Figura 3.6: Esquemático del encapsulado del optoacoplador

de la rueda. Estos últimos son interpretados como desplazamientos de la misma y por lo tanto generan errores en la cuenta de pulsos.

Esto impide tener un seguimiento certero del ángulo de rotación y por consiguiente de la posicion del robot. Luego de descubierto este problema, se decidió agregar otro optoacoplador por rueda, de modo que las ondas pulsadas de ambos optoacopladores en cada rueda queden en cuadratura. Esto permite tener una salida que queda naturalmente representada en un código Gray. La Figura 3.7 muestra las ondas pulsadas en cuadratura y el valor resultante.

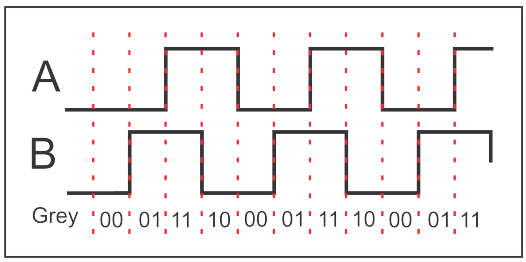


Figura 3.7: Ondas en Cuadratura

Este nuevo sistema permite obtener tanto la variación en el ángulo de rotación del eje, como el sentido de giro del mismo, de modo que los rebotes que se generaban anteriormente no sumen al ángulo medido. Esta configuración además cuadruplica la resolución con la que se mide el ángulo, ya que en el sistema de un encoder por rueda, se contabilizan pulsos solamente con el flanco de subida de la onda generada. Con la nueva configuración de dos optoacopladores por rueda, tenemos señales generadas por cada cambio de estado del conjunto del código Gray. En la Figura 3.7 es sencillo observar que se tienen ahora cuatro cambios de estado, en donde antes se contabilizaba un solo flanco de subida.

Sin embargo, ajustar la posición del segundo optoacoplador de modo que las ondas queden según el modelo deseado, resultó muy engorroso y el sistema se desajustaba ante el menor movimiento. Esto es debido a que las zonas obstruidas del disco son demasiado finas y poner los sensores en cuadratura manualmente es casi

imposible. Se observó además que cuando se lograba la cuadratura en un punto del disco, se desajustaba en otro, debido a la irregularidad en los tamaños de las hendiduras Frente a este nuevo problema se optó por obstruir la mitad de las hendiduras del disco de modo que el ajuste resulte más sencillo y robusto, obteniendo de esta forma el doble de la resolución que en el planteo inicial, es decir 40 pulsos por vuelta.

Es importante aclarar que la modificación de agregar el segundo tracker por eje, se decidió luego de las pruebas iniciales del algoritmo de odometría, y que por lo tanto, su incorporación no estaba planeada al momento de realizar las placas acrílicas que conforman la plataforma robótica. Es por esto que su ajuste mecánico no es tan prolijo como el del resto de los sensores.

A su vez, los PCB realizados para el ajuste de los nuevos optoacopladores difieren en un componente menos que los anteriores. Esto es porque se decidió aprovechar la resistencias de *pull-up* presentes en la placa Arduino, y obtener un PCB más sencillo. El esquemático de dicho circuito se puede observar en la Figura 3.8 donde $R1 = 100\ \Omega$.

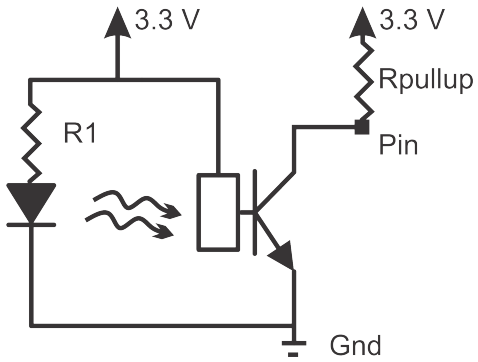


Figura 3.8: Esquemático del PCB del optoacoplador Versión 2

3.2. Sensores

Se estudiaron los múltiples sensores seleccionados para el RattusBot y se realizaron rutinas de software que implementan las funcionalidades deseadas. Además se diseñaron y realizaron circuitos en PCB para adaptar las señales de entrada a la placa Arduino.

3.2.1. Cámara

Como se menciona el el Capítulo 2, la cámara le permite al robot identificar en que ambiente se encuentra. Se eligió utilizar el shield Arducam [1] con la cámara $OV5642$. A través del mismo se realiza el manejo de la cámara y tarjeta SD. A su vez, para realizar pruebas y ajustes se utilizó la versión del shield que cuenta con

un display LCD. Utilizando el LCD para *debuggear* los seteos de la cámara y la calidad de las imágenes obtenidas.

Lamentablemente este shield no cuenta con documentación suficiente que permita aprovechar al máximo sus funcionalidades. Sin embargo, utilizando los ejemplos proporcionados por el fabricante e investigando las librerías correspondientes se consiguió el objetivo principal, de capturar una imagen y llevarla a escala de grises. Se logró a su vez almacenar una matriz de ceros y unos, correspondientes a la imagen en blanco y negro puros. El objetivo de esto último era realizar algún tipo de procesamiento de imágenes.

Finalmente, teniendo en cuenta las limitaciones, tanto de la placa Arduino, como del shield, se optó por un simple conteo de pixeles que se realiza al momento de la captura de la imagen, y por este motivo, en la versión final del código, se omite almacenar la imagen, mejorando así la performance en tiempo del proceso de obtención de la imagen.

Se implementó una rutina que saca una foto en RGB 565 (5 *bits* para codificar el rojo, 6 *bits* para el verde y 5 *bits* para el azul). Se investigaron e implementaron distintas ecuaciones de conversión para obtener una imagen en escala de grises. Mediante la observación de estas imágenes obtenidas, y la comparación de estas con una imagen en blanco y negro puros, obtenida de la anterior utilizando un umbral de conversión. Se eligió la siguiente ecuación de conversión.

$$NivelGris = \frac{(rgbComp.R) * 3 + (rgbComp.G) * 3 + (rgbComp.B))}{10}$$

Luego si $NivelGris > 3$, Se cuenta como pixel blanco, sino como pixel negro. De esta manera, teniendo fotos de patrones prefijados, contando la cantidad de pixeles negros y eligiendo patrones que difieren en una cantidad suficientemente grande de pixeles se puede saber que patrón se está observando. Incluso aunque la cuenta de pixeles entre imágenes consecutivas varía considerablemente. El algoritmo funcionará correctamente siempre que la diferencia de número de pixeles entre patrones sea mayor que la diferencia más significativa entre cuentas del mismo patrón.

Esta cuenta de pixeles negros, es la que se utiliza para seleccionar en que ambiente se esta trabajando. Considerando que las condiciones de iluminación influyen de gran manera en la cuenta de píxeles, se optó por colocar cuatro leds, alrededor de la cámara, que uniformicen la luz para distintas capturas. Los mismos se encienden unos microsegundos antes de la captura, y se apagan luego de finalizada la misma, para evitar el consumo innecesario de corriente. Entonces, al principio de la rutina principal, por única vez, se toma una foto y de la misma solo se guarda el conteo de pixeles.

La rutina encargada de esto, a su vez, compara el conteo con los rangos prefijados y decide en que entorno se encuentra el robot. De esta manera la rutina principal recibe el número de ambiente en el que se encuentra.

Para simplificar el conexionado del shield, se implementó el manejo de la cámara en la DUE encargada de la navegación. Fue entonces necesario transmitir la información del ambiente a la DUE encargada de los modelos neuronales. Para esto, se utiliza uno de los bytes de la comunicación entre arduinos (destinada principalmente a transmitir la posicion (x, y)).

De esta manera, para conseguir una sincronización adecuada, ambas placas tienen programado una primera parte de ajuste de entorno. La DUE de navegación realiza la captura de la imagen y decide el entorno, luego lo manda por el puerto paralelo. La DUE encargada de la red neuronal espera hasta tener un dato en ese puerto, setea el ambiente de trabajo y luego espera obtener un dato de posición antes de empezar con el algoritmo que computa las neuronas.

Ambas rutinas consideran en caso de obtener un dato erróneo, que el ambiente de trabajo es el número 1. Esto es debido a fallos aleatorios que pueden ocurrir en la imagen capturada por el sensor. Se descubrió que el shield encargado del manejo de la cámara presenta errores aleatorios que derivan en imágenes inservibles. Dado que no se encontró forma de sistematizar estos errores, ni detectarlos en tiempo de ejecución, se optó por elegir un entorno por defecto y así evitar fallas en el sistema completo.

Finalmente, cabe destacar que el sistema implementado no funciona correctamente si la imagen capturada no está bien encuadrada. Si la captura es parcial, el algoritmo cuenta un número erroneo de píxeles y devuelve resultados incorrectos. Este problema se considera aceptable teniendo en cuenta que la captura se realiza únicamente al inicio del funcionamiento de la plataforma.

3.2.2. Bumper

Se entiende como Bumper al conjunto de los microswitches con el parachoque. Es el sistema encargado de detectar y amortiguar los choques frontales.

Se implementaron rutinas de interrupción asociadas a los pulsadores del bumper para resolver el comportamiento del robot luego de un choque. Las mismas aprovechan las ventajas de Arduino Due en cuanto a la utilización de interrupciones y la posibilidad de configurar cualquiera de sus pines como señal de interrupción.

Las rutinas de atención a las interrupciones asociadas a cada microswitch, simplemente levantan una bandera diferente para cada pulsador, que habilita en el Round-Robin una rutina de detención y desvío que depende de cual sensor se activó.

Se construyó un parachoque de acrílico, y se adaptó para activar los microswitches según el lugar de impacto. La adaptación se realizó utilizando tornillos a modo de eje, resortes para devolver el acrílico a su posicion natural después del impacto, una palanca de alambre de acero grueso y poco flexible para el accionado de los microswitches, así como también arandelas y tuercas que permiten fijar las partes de modo que el movimiento deseado sea posible.

Para el conexionado de los microswitches con la Arduino se utilizaron las resistencias de *pull-up* incluidas en la placa. De esta manera se pudo obviar la fabricación de un PCB de adaptación. En la Figura 3.9 se puede observar el esquema del mismo.

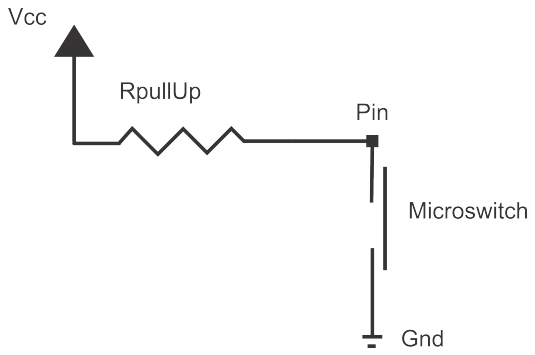


Figura 3.9: Esquema de conexión de microswitches

3.2.3. Sensores de tilt

Los sensores de tilt son resistencias cuyo valor varía al ser deformados. Los mismos son utilizados para evitar choques laterales.

Se construyó un circuito simple de adaptación para leer desde la DUE la variación de esta resistencia. El mismo consiste en un simple divisor de voltaje, alimentado por la fuente de la Arduino. En la Figura 3.10 se puede observar el conexionado del mismo. La Ecuación (3.14) muestra la relación entre el voltaje leído y el valor de la resistencia del tilt, donde $R_{aux} = 3.7\ k\Omega$. Si bien la misma es no lineal, se puede establecer experimentalmente un valor de V_t que asegure que hay deformación.

$$V_t = \frac{V_{cc} R_{tilt}}{R_{tilt} + R_{aux}} \tag{3.14}$$

Se conectaron los V_t correspondientes a cada sensor a entradas analógicas de la placa. En el código principal se resolvió consultar en cada iteración el valor de dicho pin y en función de este decidir si girar el robot hacia el otro lado o continuar con el movimiento.

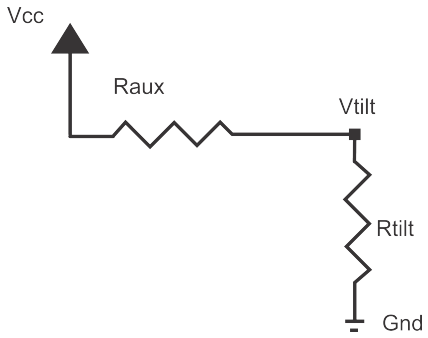


Figura 3.10: Esquema de conexión de tilts

3.3. Motores

Para lograr el movimiento del robot se usaron dos motores DC, dados en la especificación del proyecto. Su funcionamiento es simple y se describe en las secciones siguientes. Cabe destacar que los modelos de motores comprados cuentan con una pequeña caja de reducción integrada, que contiene engranajes de plastico para transmitir el movimiento desde el eje del motor hasta el eje de la rueda propiamente dicha. Estos engranajes son delicados, y es frecuente que se rompa algún sector, inutilizando toda la pieza. Durante la realización de este proyecto se utilizaron en total unos seis motores, teniendo que ir sustituyendolos ante desperfectos mostrados en su funcionamiento.

3.3.1. Motores DC

Los Motores DC son motores de corriente continua muy simples de utilizar. Su principio de funcionamiento implica que su velocidad de giro es proporcional a la alimentación. Por esta razón controlando la magnitud de esa alimentación (y su sentido), se consigue controlar la velocidad de giro y el sentido del mismo.

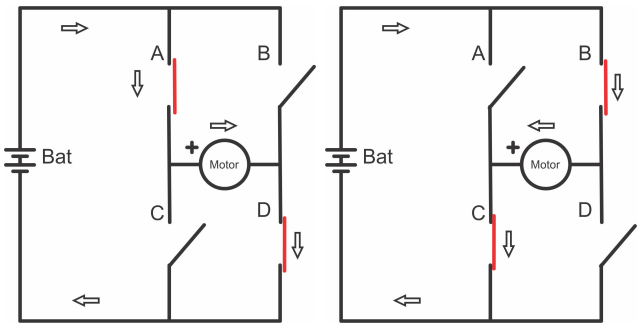
Conseguir que un motor de corriente continua gire es simple, alcanza con conectarlo a una batería (teniendo cuidado de que no se exceda el voltaje máximo de alimentación). Invirtiendo el conexionado se consigue que el mismo gire en la dirección contraria. Cambiando el voltaje de alimentación se consiguen a su vez velocidades distintas de giro (voltajes más grandes provocan giros más rápidos).

Una forma sencilla y muy utilizada para comandar estos motores, consiste en la utilización de drivers basados en Puentes H (H-bridge). El driver es necesario debido a que en general, los microcontroladores no son capaces de entregar la potencia necesaria para alimentar un motor.

Puentes H

Esencialmente un puente H son cuatro llaves, que según que combinación esté cerrada, regulan la circulación de corriente en el circuito. El nombre proviene de la H que se forma en el dibujo de las cuatro llaves.

En la Figura 3.11 se muestra un conexionado de puente H simplificado, en el que A, B, C y D son llaves.



(a) Puente H forward (b) Puente H backward

Figura 3.11: Esquema puente H

Cuando las llaves A y D están cerradas (con B y C abiertas) la corriente circulará siguiendo el sentido de las flechas (Figura 3.11 (a)), moviendo el motor en una determinada dirección.

Al invertir el estado de las llaves (abriendo A y D, y cerrando B y C), el sentido de la corriente que circula por el motor será el opuesto (Figura 3.11 (b)). Consiguiendo así invertir la dirección de giro del motor.

Alternando el estado de las llaves podemos conseguir que el motor avance o retroceda. Cuando las llaves se encuentran todas abiertas el motor no está alimentado, por lo que se detiene. También, cuando tanto A y B están cerradas, como si C y D lo están (con el otro par abierto), la diferencia de potencial en bornes del motor es nula, y por esto no circula corriente, por lo que también se detiene.

Es importante remarcar que las combinaciones de A y C cerradas a la vez provocaría un cortocircuito en la batería, el cual provocaría que la misma se dañe . En la práctica, los drivers implementan este conexionado, pero agregando componentes que protegen de situaciones no deseadas, como por ejemplo el cortocircuito

en la fuente. Los comandos que activan las llaves están combinados para funcionar con una lógica de dos parámetros que determinan cuales compuertas están abiertas y cuales cerradas.

Control de velocidad de un Motor DC.

Como se menciona anteriormente, aplicando energía al motor DC haremos que éste gire, y quitando la energía del motor, haremos que deje de girar. El puente H es capaz de controlar estos dos estados. Para conseguir hacer que el motor pueda girar a distintas velocidades, se utiliza una técnica de modulación de ancho de pulso (PWM por su sigla en inglés Pulse Width Modulation).

La PWM sirve para simular una gama de valores analógicos a partir de circuitos digitales (un valor fijo de alimentación). Conmutando entre encendido y apagado, se puede simular un valor constante o promedio.

La Figura 3.12 muestra tres señales. Las tres señales tienen la misma frecuencia, es decir, conmutan a su valor máximo V_+ de forma periódica. Lo que las diferencia es el ancho del pulso, es decir, el tiempo en el que están en valor nulo. Variando el tiempo en alto, podemos hacer que el motor vea un valor entre cero y el nivel de alimentación.

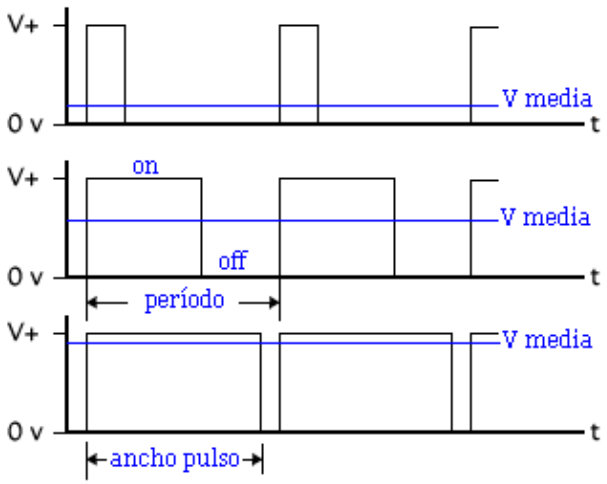


Figura 3.12: Ejemplo de PWM (Tomado de [3])

3.3.2. Comando de Motores

Para comandar los motores se utilizó un driver basado en un "Dual Full-Bridge Driver" (L298). El mismo consiste en dos puentes H de comando independiente y

	IN1	IN2	EN
Avanzar	ALTO	BAJO	$EN > 0$
Retroceder	BAJO	ALTO	$EN > 0$
Frenar(Libremente)	ignorada	ignorada	$EN = 0$
Frenar(Forzado)	$IN1 = IN2$	$IN1 = IN2$	$EN = 255$

Tabla 3.1: Comportamiento lógico de un Motor

una señal de enable para cada puente. La placa utilizada, incorpora este driver, junto con una fuente regulada, puesto que el L298 permite separar la alimentación de los motores, de la lógica y así trabajar con alimentaciones del orden de los 12 V para la placa, y aprovechar la regulación interna de la misma para subir el nivel lógico de 5 V y alimentar la placa Arduino.

La placa recibe como entradas seis señales de control (tres por cada salida de motor), además de la señal de 5 V y GND. Para cada salida de motor se tiene dos señales digitales que controlan el sentido de giro de las ruedas y una señal de *enable* que ajusta el ciclo de trabajo de la onda PWM que controla el motor. Regulando así la corriente que alimenta el motor y por lo tanto su velocidad y dirección de giro.

Para cada motor las tres terminales de control son, $IN1$ e $IN2$ que definen el sentido de la rotación y el pin EN que condiciona la potencia que se suministra al motor (mediante el uso de PWM). La lógica de operación del driver es la que se indica en la Tabla 3.1.

Para la implementación del movimiento del robot, se empezó por definir primero un grupo de funciones globales (avanzar, retroceder, girar en el lugar, hacia la derecha y hacia la izquierda) que contienen funciones que comandan a los motores, realizando este control de a pares. Las funciones realizan el seteo de los pines usados y se les pasa un parámetro asociado al ciclo de la PWM de salida en la Arduino. A su vez, esta salida está asociada a la potencia que se le entrega a los motores por medio del driver (la velocidad del robot depende de este parámetro, pero influye también la situación y condiciones en las que se encuentre trabajando el motor).

En la Tabla 3.1 se puede observar que para la modalidad de freno forzado las únicas condiciones necesarias son que $EN = 255$ y $IN1 = IN2$. Para éste caso se utilizó la combinación $IN1 = IN2 = HIGH$.

Estos motores DC no tienen identificados sus bornes (es decir, cual es el conexionado para que avance) ni todos los drivers tienen diferenciadas sus señales. En general la entrada de habilitación se llama *Enable*, pero las digitales no están diferenciadas. Por lo que se requiere conectar el motor al driver y utilizando una rutina de avance en la Arduino establecer la correspondencia de los pines con el sentido de giro.

El driver utilizado trabaja con señales lógicas de 5 V. Por lo que para el conexionado del mismo con la Arduino Due se utilizaron conversores de niveles lógicos. Se consiguieron en el mercado cuatro placas capaces de realizar estas conversiones. Aunque originalmente fueron compradas para realizar la comunicación entre la Arduino Mega y la DUE, al sustituir la Mega por una segunda DUE, los mismos no fueron necesarios y se aprovecharon para el comando de los motores.

3.4. Comportamiento

Se entiende por comportamiento del sistema de navegación a la descripción conceptual de la lógica de funcionamiento del robot recorriendo la arena. El mismo está basado en una arquitectura de Round-Robin con interrupciones, y pretende simular un movimiento aleatorio. Para la implementación de la navegación se utilizaron los sensores de choque descritos en las Secciones 3.2.2 y 3.2.3, conjuntamente con una rutina de movimiento entre choques, pseudo aleatoria.

Se definió que el comportamiento del robot en la arena será aleatorio, simulando el de una rata que explora un ambiente similar. Durante el recorrido se estima la posición del robot relativa al punto de partida, utilizando la información provista por los sensores. El control del comportamiento de la plataforma se realiza en una de las entradas del Round-Robin.

Entonces, el robot parte de una posición inicial (conocida y predefinida), recorre la arena variando su función de movimiento cada 1 s aproximadamente hasta encontrar un borde de la arena. Está situación de choque provoca a su vez, una rutina de corrección del movimiento para lograr salir del caso de borde. Una vez superada esa situación, se continúa con la función de movimiento.

3.4.1. Rutina aleatoria

Para la implementación de esta rutina se presentaron dos grandes inconvenientes. El primero debido a la limitante en el rango de velocidades a manejar para el robot. Para minimizar los errores en el sistema de odometría, el rango de velocidades en el que pueden funcionar las ruedas está acotado, y es mucho menor al que permite la programación en sí. El siguiente inconveniente es debido a que el tiempo total consumido por el procesador para realizar las tareas necesarias entre llamadas consecutivas de la función de $RutinaAleatoria()$ es muy poco.

Debido a que se utiliza una función aleatoria de distribución uniforme para la variación de la velocidad del sistema, si el randomizado es demasiado frecuente, la velocidad del robot tiende a la media de la distribución y se pierde la aleatoriedad.

Estos inconvenientes sumados, llevaron a la necesidad de incluir un factor de tiempo entre actualizaciones sucesivas de las velocidades correspondientes. A su vez, es fundamental aclarar que luego de ingresado el sistema a la parte correspondiente al *R-R* no puede ser utilizada la función *delay*(), incluida en las librerías de Arduino. Esto es debido a que la suma correspondiente a un nuevo pulso en los encoders se hace en el *R-R*, y durante un *delay*, se estarían perdiendo pulsos, generando así mayores errores en la estimación de la posición.

3.4.2. Arena Virtual

Surge la necesidad de evitar choques fuertes que puedan causar roturas o desajustes en la plataforma. Sumado a dicha necesidad, para facilitar las pruebas de funcionamiento, y dado que no se contó con espacio físico permanente para la construcción de la arena con su correspondiente suelo y paredes, se optó por implementar una simulación de dichas paredes.

Para esta simulación, se utiliza una función que en cada interacción del *loop* verifica de acuerdo a la posición actual, que el robot se encuentre dentro de la arena.

La arena virtual considera un espacio determinado por un cuadrado interior y centrado a la arena original de 40 *cm* menos de lado. Esto es para tener en cuenta las demoras que presenta el sistema mecánico para frenar completamente. En caso de que la plataforma esté fuera de dichos límites, se procede a verificar el ángulo del choque virtual, y en función de este se simula una interrupción del bumper correspondiente. Es decir, se levanta la bandera que habilita en el *R-R* las acciones de evasión.

3.4.3. Round-Robin

El pseudocódigo del algoritmo implementado para el sistema de navegación completo se muestra a continuación, con el objetivo de esclarecer su funcionamiento.

```
While (true){

        if (giro la rueda derecha){
                incremento la cuenta de N_D
        }

        if (giro la rueda izquierda){
                incremento la cuenta de N_I
        }
```

```
        if (alguno de los N varió en más de uno){
                calculo la nueva posición
                envio la nueva posición
        }

        Chequeo los tilts por acercamientos laterales

        if (hubo acercamiento derecho){
                corrijo tiltDerecho
        }

        if (hubo acercamiento izquierdo){
                corrijo tiltIzquierdo
        }

        if (hubo choque izquierdo){
                corrijo bumpIzquierdo
        }

        if (hubo choque derecho){
                corrijo bumpDerecho
        }

        if (no estoy atendiendo a caso de choque){
                continuo el movimiento
                verifico arenaVirtual, deteción choques
        }
}
```

Además de este *loop* principal, se implementaron las rutinas de atención a las interrupciones, descritas en las secciones correspondientes a cada sensor. Así como también funciones que implementan el cálculo de la nueva posición, el envío de la misma, las rutinas de atención a los choques y la función que genera el movimiento aleatorio.

Es importante destacar que la inclusión en el Round-Robin de funciones de atención a eventos, se debe principalmente a un error encontrado durante la implementación. En algunos casos, incluir determinados comportamientos en las rutinas de atención a la interrupciones, provocó defectos en la generación de las ondas PWM por parte de la Arduino. No fue posible determinar la causa y se utilizó un tiempo considerable intentando encontrarla. Finalmente se optó por eliminar de las rutinas de atención a las interrupciones toda instrucción que pudiera ser implementada en el *R-R* y con esto se solucionó el problema, evitando insumir más tiempo en la resolución de este inconveniente.

Capítulo 4

Modelos Neuronales

En este Capítulo, se profundiza sobre todos los aspectos que hacen al modelado de las neuronas implementado en el RattusBot. Se comienza por repasar los aspectos generales presentados en el Capítulo 2 con mayor detalle, para luego abordar las ecuaciones matemáticas que caracterizan el comportamiento espacial de las *grid cells*, así como su comportamiento eléctrico. Se describen también los recursos de optimización utilizados para maximizar la cantidad de *grid cells* modeladas.

4.1. Modelado de neuronas

Como se presenta en el Capítulo 1, una *grid cell* es un tipo de neurona que se encuentra en la corteza entorrinal de muchas especies y a la cual se atribuye la capacidad de un individuo de comprender su posición en el espacio. Si se coloca una rata dentro de un espacio contenido, sensando el comportamiento de una de estas neuronas se puede observar que los puntos en los que se generan disparos forman una grilla de nodos equidistantes como se muestra en la Figura 4.1, donde se mapea en negro el recorrido de una rata, en una superficie confinada, y en rojo las cordenadas para las cuales esa neurona presenta actividad.

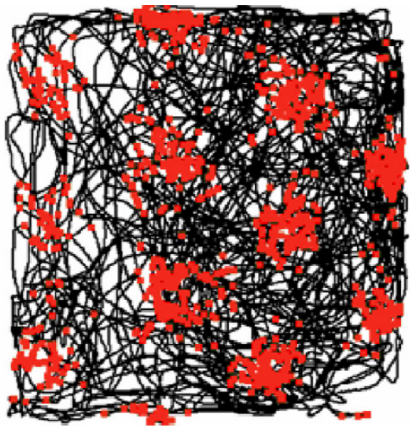


Figura 4.1: Actividad de una *grid cell* en la corteza entorrinal de una rata desplazándose en un espacio contenido.(Tomada de [6])

Teniendo en cuenta que la frecuencia de disparo de una neurona depende de la corriente de estímulo a la que está sometida, es necesario contar con una función matemática que controle esta variable, de modo que la neurona modelada se comporte de acuerdo a lo deseado. Se recurre entonces a la ecuación propuesta por Solstad [12] que se describe a continuación. El modelo fue indicado por Ing. Javier Cuneo que es uno de los clientes del proyecto.

4.1.1. Función g(x,y)

"Las funciones de *grid cell* $g_w(x, y)$ se construyen a partir de una suma de tres sinusoides bi-dimensionales, especificadas por sus vectores de onda k, con 60 y 120 grados de diferencia angular" [12] como se puede observar en la Figura 4.2.

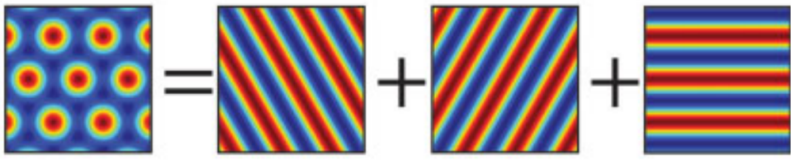


Figura 4.2: Función de red como suma de tres sinusoides bi-dimensionales desfasadas 60 y 120 grados. (Tomado de [12])

En la Ecuación (4.1) se presenta la expresión analítica de la función de red utilizada.

$$g_w(r) = g_w^{max}\frac{2}{3}(\frac{1}{3}\sum_{i=1}^{3}\cos(k_i \cdot (r - r_0)) + \frac{1}{2}) \tag{4.1}$$

Donde $r = (x, y)$ es el vector posición, $r_0 = (x_0, y_0)$ es un punto de referencia espacial y los elementos k_i son de la forma:

$$k_1 = \frac{k}{\sqrt{2}}[\cos(\theta + \frac{\pi}{12}) + \sin(\theta + \frac{\pi}{12}), \cos(\theta + \frac{\pi}{12}) - \sin(\theta + \frac{\pi}{12})] \tag{4.2}$$

$$k_2 = \frac{k}{\sqrt{2}}[\cos(\theta + \frac{5\pi}{12}) + \sin(\theta + \frac{5\pi}{12}), \cos(\theta + \frac{5\pi}{12}) - \sin(\theta + \frac{5\pi}{12})] \tag{4.3}$$

$$k_3 = \frac{k}{\sqrt{2}}[\cos(\theta + \frac{3\pi}{4}) + \sin(\theta + \frac{3\pi}{4}), \cos(\theta + \frac{3\pi}{4}) - \sin(\theta + \frac{3\pi}{4})] \tag{4.4}$$

Donde se puede ver que $\|k_1\| = \|k_2\| = \|k_3\| = k$, siendo k el valor tal que la cantidad de nodos por unidad de distancia λ es de la forma:

$$\lambda = \frac{4\pi k}{\sqrt{3}}$$

El valor θ en las Ecuaciones (4.2), (4.3) y (4.4) corresponde al eje mayor de la red.

Se puede entonces definir una expresión reducida para la Ecuación (4.1), simplificando el producto interno y normalizando a $g_w^{max} = 1$. Se expresa esta ecuación teniendo en cuenta los parámetros que caracterizan a cada *grid cell*.

$$g_i(x,y) = \frac{2}{3}(\frac{1}{3}\sum_{i=1}^{3}\cos(k_i^{'}(x,y) + \frac{1}{2}) \tag{4.5}$$

Con $k_i^{'}(x,y)$ funciones de x e y de la forma:

$$k_{1_i}^{'} = \frac{4\pi\lambda_i}{\sqrt{6}}\Big[\big(\cos(\theta_i+\frac{\pi}{12})+\sin(\theta_i+\frac{\pi}{12})\big)\,(x-\varphi_{i,x})+\big(\cos(\theta_i+\frac{\pi}{12})-\sin(\theta_i+\frac{\pi}{12})\big)\,(y-\varphi_{i,y})\Big] \tag{4.6}$$

$$k_{2_i}^{'} = \frac{4\pi\lambda_i}{\sqrt{6}}\Big[\big(\cos(\theta_i+\frac{5\pi}{12})+\sin(\theta_i+\frac{5\pi}{12})\big)\,(x-\varphi_{i,x})+\big(\cos(\theta_i+\frac{5\pi}{12})-\sin(\theta_i+\frac{5\pi}{12})\big)\,(y-\varphi_{i,y})\Big] \tag{4.7}$$

$$k_{3_i}^{'} = \frac{4\pi\lambda_i}{\sqrt{6}}\Big[\big(\cos(\theta_i+\frac{3\pi}{4})+\sin(\theta_i+\frac{3\pi}{4})\big)\,(x-\varphi_{i,x})+\big(\cos(\theta_i+\frac{3\pi}{4})-\sin(\theta_i+\frac{3\pi}{4})\big)\,(y-\varphi_{i,y})\Big] \tag{4.8}$$

La Ecuación (4.5) presenta la expresión de $g_i(x,y)$ que se define como la ecuación de red asociada a la neurona i. En función de esta discriminación por célula se definen los parámetros que dependen de cada una. Siguiendo este razonamiento, las Ecuaciones (4.6), (4.7) y (4.8) representan las funciones $k_{1_i}^{'}$, $k_{2_i}^{'}$ y $k_{3_i}^{'}$ asociadas también a la neurona i. Sabiendo que las *grid cells* difieren en el punto de referencia (r_0), ángulo de fase (θ) y distancia entre nodos (λ) de la función g, se definen los parámetros $r_{0_i} = (\varphi_{i,x}, \varphi_{i,y})$, θ_i y λ_i.

4.1.2. Modelado de parámetros

Recordando que los parámetros de la función de red dependen por un lado de las características intrínsecas de la neurona y por otro lado de la influencia del entorno se pueden modelar $\varphi_{i,x}$, $\varphi_{i,y}$, θ_i y λ_i de la siguiente manera (según planteado en "The computational influence of neurogenesis in the processing of spatial information in the dentate gyrus" [11]):

$$\lambda_i \ [\frac{grids}{m}] = 3 - 1.5\chi_i$$

$$\theta_{\chi_i} \ [rads] = 2\pi\eta(0,1)$$

$$\theta_{env} \ [rads] = 2\pi\eta(0,1)$$

$$\theta_i \ [rads] = \theta_{env} + \theta_{\chi_i}$$

$$\varphi_{i_x} \ [m] = \eta(0,1)$$

$$\varphi_{i_y} \ [m] = \eta(0,1)$$

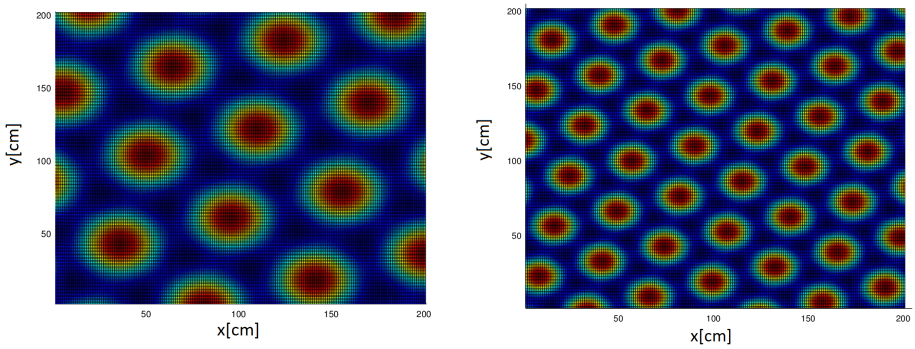
$$\varphi_{env} \ [m] = \eta(0,1)$$

$$\varphi_{i,x} \ [m] = \varphi_{env} + \varphi_{i_x}$$

$$\varphi_{i,y} \ [m] = \varphi_{env} + \varphi_{i_y}$$

Donde χ_i es un parámetro que depende de la posición de la célula en la corteza entorrinal, que a los efectos del modelo toma valores entre 0 y 1. Para la célula i el valor de χ_i es $\frac{i}{N}$ donde N es la cantidad de neuronas modeladas; λ_i es como se indica anteriormente la distancia entre nodos de la grilla, tomando valores entre 1.5 y 3, $\eta(0,1)$ es un número aleatorio de distribución normal con media 0 y desviación estándar 1; θ_{env} es el desfasaje angular en la grilla que el ambiente produce para todas las células; θ_{χ_i} es la orientación de la célula sin influencia del ambiente; θ_i es el ángulo total del eje mayor de la grilla; φ_{i_x} es el offset horizontal de la grilla para la célula sin influencia del ambiente; φ_{i_y} es el offset vertical de la grilla para la célula sin influencia del ambiente; φ_{env} es el offset que el ambiente produce en la grilla; $\varphi_{i,x}$ es el offset horizontal total de la célula; $\varphi_{i,y}$ es el offset vertical total de la célula (adaptado del material suplementario de [11]).

Las funciones obtenidas se pueden graficar obteniendo resultados como los mostrados en las Figuras 4.3, (a) y (b). En ellas se representa una función $g_i(x, y)$ con un λ menor en la Figura 4.3 (a) y otra con un λ mayor en la Figura 4.3 (b).



(a) con $\lambda = 1.6\ m$ (b) con $\lambda = 2.9\ m$

Figura 4.3: $g_i(x, y)$

Se tiene entonces un modelo completo de la función de red asociada a cada neurona, generado de forma aleatoria e influenciado por el ambiente. Para la implementación en el RattusBot se elaboró una planilla de cálculo que permite sortear masivamente los valores necesarios y los presenta de forma medianamente ordenada. Se implementó entonces a partir de los datos obtenidos un archivo de encabezado (.h) que permite incluir cómodamente toda la información al código que simula las neuronas.

4.2. Simulación de neuronas

Contando entonces con un modelo matemático de la malla de cada *grid cell*, se puede utilizar esta función como modulador de la corriente que estimula a cada neurona. Se plantea entonces una corriente de excitación para cada *grid cell* de la siguiente forma:

$$I_{ex_i}(x, y) = I_{max} g_i(x, y)$$

Donde I_{max} es una corriente en pA. Se tienen ahora los elementos necesarios para simular el comportamiento eléctrico de una neurona, utilizando $I_{ex_i}(x, y)$ como estímulo y un modelo de mayor complejidad que el ya descrito *integrate and fire*.

4.2.1. Izhikevich

Para el comportamiento eléctrico de las neuronas se utilizó el modelo propuesto por Eugene M. Izhikevich en "Simple Model of Spiking Neurons" [9] para neuronas pulsantes. Este modelo fue indicado por el Ing. Biomédico Javier Cuneo y considera aspectos biológicos que hacen al comportamiento eléctrico de las neuronas. Se estudió e implementó el algoritmo aunque sus fundamentos exceden los conocimientos del equipo ya que son de carácter biológico y no fueron investigados en profundidad.

El modelo propone las Ecuaciones (4.10) y (**??**) que rige la tensión de membrana de la célula. Este voltaje se corresponde en el modelo de *integrate and fire* con el presente en bornes del capacitor C_m visto en la Figura 2.3.

$$v^{'} = \alpha v^2 + \beta v + \gamma - u + I \tag{4.9}$$

$$u^{'} = a(bv - u) \tag{4.10}$$

Donde α, β y γ son parámetros que caracterizan a la neurona, e I se modela como I_{ex_i}, vista en 4.2. Con u y v en mV. Estas variables se deben resetear luego de un pico con el siguiente criterio.

$$Si\ se\ cumple\ que\ v\ \geq\ v_{peak}\ entonces:$$
$$v = c$$
$$u = u + d$$

El siguiente modelo es una aproximación por el método de Euler hacia atrás de la ecuación diferencial original, con algunas modificaciones propuestas por el Ing. Javier Cuneo para adaptarlas a las *grid cells*. La aproximación por Euler es la misma que Izhikevich utiliza en las simulaciones en tiempo real que reporta en su artículo. Se discretiza el tiempo en pasos de 1 ms con lo que se obtiene un comportamiento aceptable aunque en ocasiones presenta ciertos errores transitorios. Las Ecuaciones (4.11) y (4.12) corresponden al algoritmo discretizado utilizado.

$$v_i = v_{i-1} + \frac{t}{C}\Big[\ k(v_{i-1} - v_{off} - v_r)(v_{i-1} - v_{off} - v_{th}) + I - u_{i-1} + b(v_{off} + v_r)\Big] \tag{4.11}$$

$$u_i = u_{i-1} + t(a(b(v_{i-1}) - u_{i-1})) \tag{4.12}$$

Con el correspondiente reseteo.

$$Si\ se\ cumple\ que\ v\ \geq\ v_{peak} + v_{off}\ entonces:$$
$$v_i = c + v_{off}$$
$$u_i = u_i + d$$

En el planteo anterior, v_i representa el potencial de membrana en el paso i; u_i es una variable para la recuperación de membrana en el instante i; t es el tiempo entre el paso i y el $i+1$; I representa las corrientes sinápticas que excitan a la neurona; v_{off} es un $offset$ agregado al modelo para que los niveles de tensión sean siempre positivos; v_{peak} es la tensión de membrana máxima alcanzada cuando la neurona dispara; v_r es la tensión de descanso de la neurona; v_{th} la tensión de umbral de disparo; c es la tensión de recuperación a la que baja el potencial de membrana luego de un disparo; d es el equivalente a lo anterior para u; a, b y C son constantes que caracterizan a las curvas. Traducido y recopilado de la fuente [9]

Se consideran los siguientes valores para los parámetros del modelo, sugeridos por el Ing. Javier Cuneo que está familiarizado con el algoritmo y maneja los conceptos biológicos que lo definen. El artículo de Izhikevich también sugiere valores para estos parámetros, en general dentro de márgenes, debido a que el mismo modelo puede modelar neuronas de diferentes características.

$$a = 0.03\ \frac{1}{s}$$
$$b = -2\ n\Omega^{-1}$$
$$k = 0.7\ \frac{\mu\ A}{V^2}$$
$$v_r = -60\ mV$$
$$v_{th} = -40\ mV$$
$$C = 10\ nF$$
$$v_{off} = -v_r$$
$$c = -50\ mV$$
$$d = 100\ pA$$
$$v_{peak} = 35\ mV$$
$$I_{max} = 0.17\ pA$$

Implementando este algoritmo se obtienen valores de v como los que se muestran en la Figura 4.4, en la que se pueden observar claramente los *spikes* que producen las neuronas. Se puede ver también que los pulsos no tienen el mismo

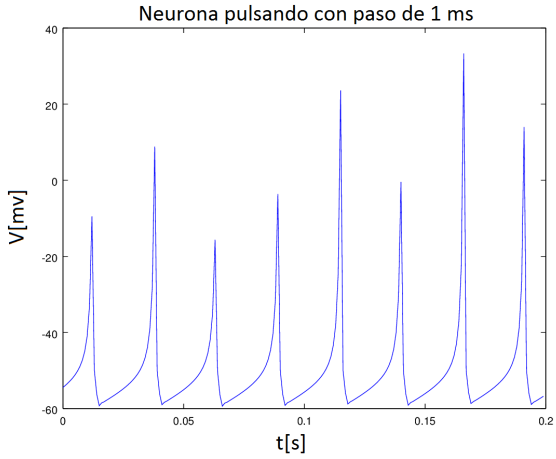


Figura 4.4: Actividad de una *grid cell* simulada en un punto fijo con un paso de 1 ms.

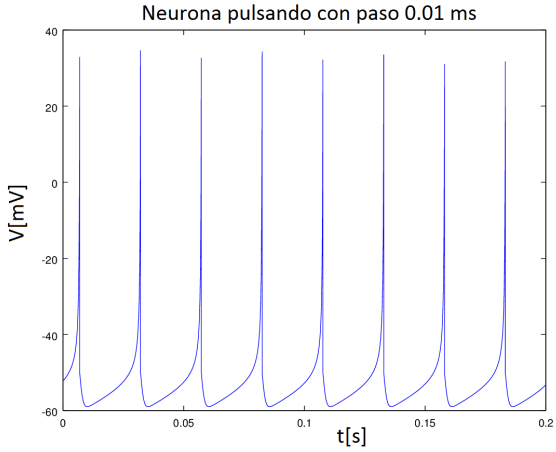


Figura 4.5: Actividad de una *grid cell* simulada en un punto fijo con un paso de 0.01 ms.

valor de pico. Esto es debido al paso de 1 ms utilizado en el método y mejora si este se reduce. Si por ejemplo se utiliza un valor de 0.01 ms los resultados son mucho más precisos, como se puede observar en la Figura 4.5.

De todos modos, dado que la información que se necesita extraer, en este caso, de la simulación es, en que instantes de tiempo hay disparos, no es necesario obtener resultados tan precisos. Como se puede observar en la Figura 4.4, los picos pueden ser detectados perfectamente con un paso de 1 ms.

4.2.2. Optimización

Visto que dentro de los objetivos del proyecto se propone modelar la mayor cantidad posible de neuronas, fue necesario un proceso de optimización del códi-

go de modo de minimizar el tiempo de cómputo de cada neurona. Para esto se realizaron modificaciones con los criterios descritos a continuación.

Escalado

En primer lugar se escalaron todas las funciones de modo que todos los valores pudieran ser representados con valores enteros. Esto mejora ampliamente el rendimiento respecto a realizar las operaciones en punto flotante, ya que el procesador utilizado cuenta con un multiplicador por hardware. Este módulo le permite realizar el producto en tan solo unos ciclos de reloj, reduciendo el tiempo de cómputo en general. Este punto fue el que más ganancia dio en cuando al tiempo de ejecución, se redujo de 40 μs por neurona a 5 μs. El escalado se realizó de acuerdo al siguiente razonamiento, si tomamos las funciones que definen el comportamiento de las neuronas:

$$v_i = v_{i-1} + t[\frac{k(v_{i-1} - v_{off} - v_r)(v_{i-1} - v_{off} - v_{th}) + I - u_{i-1} + b(v_{off} + v_r)}{C}]$$

$$u_i = u_{i-1} + t(a(b(v_{i-1} - v_{off} - v_r) - u_{i-1} + b(v_{off} + v_r)))$$

Se busca que la variable v resulte multiplicada por un factor que genere un error aceptable al truncar a entero los valores en punto flotante. Se eligió utilizar un factor de 1000 para llevar este error al orden de los 10^{-7} V, con lo que se desprecian valores de magnitud 1000 veces menores a la dimensión de v, lo que el equipo consideró aceptable. No se utilizó un factor mayor debido a la posibilidad de que en operaciones intermedias se alcancen valores mayores a 2^{16} que es el máximo valor posible de una variable de tipo entero (int) en el procesador utilizado. Si esto sucede, el $overflow$ en la operación tiene consecuencias graves en los resultados del modelo.

$$10^3 v_i = 10^3 v_{i-1} + 10^3 t[\frac{k(v_{i-1} - v_{off} - v_r)(v_{i-1} - v_{off} - v_{th}) + I - u_{i-1} + b(v_{off} + v_r)}{C}] \tag{4.13}$$

Desarrollando un poco más la Ecuación (4.13) de modo que todas las variables queden escaladas adecuadamente, se obtiene la Ecuación (4.14) planteada a continuación.

$$\begin{aligned} 10^3 v_i &= 10^3 v_{i-1} + \frac{t}{C}[\frac{10k(10^3 v_{i-1} - 10^3 v_{off} - 10^3 v_r)(10^3 v_{i-1} - 10^3 v_{off} - 10^3 v_{th})}{10^4} \\ &+ 10^3 I - 10^3 u_{i-1} + b(10^3 v_{off} + 10^3 v_r)] \end{aligned} \tag{4.14}$$

Se puede observar que todas las variables y parámetros resultan multiplicados por un factor. Este fue seleccionado adecuadamente de modo que el error por redondeo no sea significativo.

Realizando un razonamiento análogo para u, se obtiene la siguiente expresión.

$$10^3 u_i = 10^3 u_{i-1} + \frac{t(100a(b(10^3 v_{i-1} - 10^3 v_{off} - 10^3 v_r) - 10^3 u_{i-1} + b(10^3 v_{off} + 10^3 v_r)))}{100} \tag{4.15}$$

Se puede observar que la Ecuación (4.15) es equivalente a la (4.14), presentando la expresión con todos los valores escalados de los parámetros y variables.

Se definen entonces las nuevas variables y parámetros.

$$\begin{aligned} v_i^{'} &= 10^3 v_i \\ v_{i-1}^{'} &= 10^3 v_{i-1} \\ v_{off}^{'} &= 10^3 v_{off} \\ v_{th}^{'} &= 10^3 v_{th} \\ v_r^{'} &= 10^3 v_r \\ I^{'} &= 10^3 I \\ u_i^{'} &= 10^3 u_i \\ u_{i-1}^{'} &= 10^3 u_{i-1} \\ k^{'} &= 10k \\ a^{`} &= 100a \end{aligned}$$

Con estos parámetros y variables escaladas se puede definir las nuevas funciones que rigen el comportamiento del modelo.

$$v_i^{'} = v_{i-1}^{'} + \frac{t}{C}[\frac{k^{'}(v_{i-1}^{'} - v_{off}^{'} - v_r^{'})(v_{i-1}^{'} - v_{off}^{'} - v_{th}^{'})}{10^4} + I^{'} - u_{i-1}^{'} + b(v_{off}^{'} + v_r^{'})] \tag{4.16}$$

$$u_i^{'} = u_{i-1}^{'} + \frac{t(a^{'}(b(v_{i-1}^{'} - v_{off}^{'} - v_r^{'}) - u_{i-1}^{'} + b(v_{off}^{'} + v_r^{'})))}{100} \tag{4.17}$$

Esta nueva configuración formada por (4.16) y (4.17) permite manejar todas las variables y constantes de tipo entero con lo que el tiempo de cómputo se reduce sustancialmente.

Para profundizar la optimización se escaló también con el mismo criterio la función $g(x, y)$. De la Ecuación (4.1) se deduce.

$$10^3 g(x,y) = \frac{2}{3}(\frac{1}{3}\sum_{i=1}^{3} 10^3 \cos 10^3 k_i(x,y) + \frac{10^3}{2}) \tag{4.18}$$

Esto permite manejar también valores enteros para esta función mejorando aún más el rendimiento del sistema.

Otros recursos de optimización

Si bien el escalado incrementa en gran medida el número de neuronas que es posible computar en un paso de tiempo, los resultados obtenidos fueron mejorados utilizando otras herramientas de optimización.

Minimalismo
En general se implementó el código que computa el comportamiento de neuronas de modo que no se realizarán operaciones innecesarias o repetidas. Esto parece obvio pero al escribir código no necesariamente se tiene en cuenta el tiempo que se desperdicia si no se desarrolla de una forma minimalista. Algunos ejemplos de este concepto son:

Realizar operaciones dentro de un *loop* que son comunes a cada iteración. Dado que el procesado de las neuronas es un bloque de código que se ejecuta tantas veces como neuronas se simulan, esta estrategia es importante.

Algunos de los cálculos en la simulación tienen el mismo resultado para cada neurona, independientemente del momento y condiciones en que se calculen. Estos valores fueron calculados fuera del código e incluidos como constantes características de las neuronas. En general se buscó no repetir innecesariamente ninguna operación.

Se obtuvieron resultados apreciables realizando esta simple corrección, del orden de las centenas de microsegundos en el procesado total.

Lookup table
Se puede observar por ejemplo que la Ecuación (4.18) involucra un coseno, que es computado por las librerías de C utilizando un método iterativo. Esto implica un retraso en el código y por lo tanto también fue optimizado. Se utilizó una "lookup table" para determinar el valor de $cos(10^3\theta)$. Esto consiste en una tabla precalculada que en la posición $10^3\theta$ se encuentra el valor de $cos(10^3\theta)$. Con esto se calcula

el coseno sin recurrir a métodos numéricos que consumen más tiempo. Mejorando también en el orden de las centenas de microsegundos en el procesado completo

Expansión de funciones iterativas
Por último, para optimizar aún más las secciones del código que se ejecutan más frecuentemente se escribieron por extensión los ciclos de for y $while$. De este modo se elimina el salto durante la ejecución del código que implica el vaciado de la pipeline. Este reseteo no es deseado ya que significa mayor cantidad de ciclos de reloj para ejecutar cada operación. Este proceso significó una mejora menor que los anteriores, en el orden de las decenas de microsegundos.

Luego de este proceso de optimización se consiguió simular con pasos de 1 ms, 200 neuronas, insumiendo aproximadamente 900 μs en el cómputo de estas. Habiendo partido de aproximadamente 8 ms para 200 neuronas se verifica una mejora del entorno de un décimo del tiempo inicial. Se entiende que es un número más que aceptable ya que fue validado por los interesados del proyecto y roza los límites de lo que se puede mejorar sin recurrir a programar el procesador directamente en $assembler$. De todos modos siempre existe espacio para mejorar en este tipo de cometidos.

Capítulo 5

Comunicación

En este Capítulo se detallan los diferentes vínculos de comunicación que permiten a los bloques de la plataforma interactuar entre sí, así como con el exterior. Concretamente se incluye una interfaz SPI, una comunicación por Bluetooth y un puerto de tipo paralelo.

5.1. SPI

Serial Peripheral Interface (SPI), es un protocolo de comunicación serial usualmente utilizada para comunicar a un microcontrolador, con uno o más periféricos. Presenta la ventaja de ser muy veloz a corta distancia. También puede ser utilizado para la comunicación entre microcontroladores.

Este protocolo, establece que siempre debe existir un único dispositivo actuando como maestro, especificando así la condición de esclavo del resto de los elementos involucrados. En una transmisión SPI intervienen usualmente tres señales comunes para todos los dispositivos involucrados, además de una señal de selección por cada dispositivo esclavo. Las mismas son:

- MISO (*Master In Slave Out*): Señal mediante la cual, los periféricos esclavos mandan información al maestro.
- MOSI (*Master Out Slave In*): Señal de salida de datos, del maestro hacia los periféricos.
- SCK (*Serial Clock*): La señal de reloj, emitida por el microcontrolador maestro, que sirve de sincronización para la comunicación.
- SS (*Slave Select*): Un pin de selección por cada esclavo, este pin es manipulado por el maestro para avisarle al periférico que se va a comunicar con él y no con otro.

Cuando el pin *Slave Select*, correspondiente a un dispositivo, está en 0, significa que ese dispositivo es el que se comunica con el Maestro. Cuando está en 1 el periférico ignora al maestro.

El protocolo de comunicación establece un principio de funcionamiento muy simple, en el que cada *bit* es escrito (y leído) en un flanco de la señal SCK. Comúnmente, existen cuatro modos de funcionamiento del protocolo SPI, que corresponden a las combinaciones típicas de fase y polaridad de la señal de reloj. En función de estas combinación se definen la Tabla 5.1, en la Figura 5.1 se observan las formas de onda correspondiente a cada uno de estos modos, ver [2].

Modo	Polaridad	Fase
0	0	0
1	0	1
2	1	0
3	1	1

Tabla 5.1: Descripción Modos de Funcionamiento del SPI

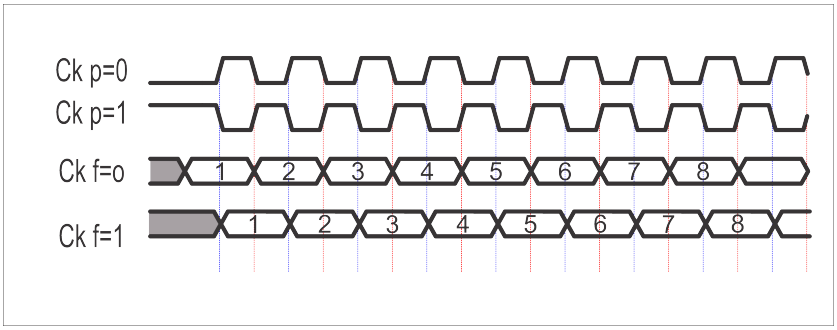


Figura 5.1: Formas de Onda según el modo de funcionamiento de la señal SCK

5.1.1. Spikes

Como se describe en el Capítulo 4, los *spikes* son los pulsos en la tensión de membrana de las neuronas. A los efectos de este proyecto, los *spikes* se almacenan en 25 *bytes*, los mismos representan para cada instante de tiempo, el estado de cada una de las 200 neuronas. Cabe aclarar que a estos se les suman 2 *bytes* representando la posición para la que se calculó dicho estado. Es importante aclarar, que si bien las señales generadas por los algoritmos de redes neuronales, son según lo descrito en la Sección 4.2.1 del Capítulo 4, a los efectos de la comunicación hacia fuera, y a instancias del interesado, se resolvió representar con el valor lógico 1 a las neuronas que "dispararon", y con un 0 a las que no. De esta manera, se obtiene una palabra de 27 *bytes* cada 1 *ms*, que es envia al exterior utilizando el SPI.

5.2. Bluetooth

Se identificó la necesidad de contar con una comunicación que permitiera el *debug* de la plataforma funcionando en conjunto. Para esto, se compró un módulo de comunicación bluetooth *HC*-05, que permite usar las librerías de comunicación serial incluidas en Arduino. El mismo permitió la recepción desde una PC de algunos datos de prueba. Surgió entonces la necesidad de estudiar las posibilidades de configuración del mismo. La configuración final, consiste en una comunicación serial, de 115200 *bps* sin *bit* de paridad, ni de parada. La idea fundamental, es perder el menor tiempo de ejecución posible en las tareas de transmisión de datos.

5.3. Posición

Dado que, tanto el proceso de adquisición de los valores de los trackers, como las operaciones pertinentes al cálculo de la evolución de la posición, son realizadas por el sistema de navegación. Resultó necesario implementar un sistema de comunicación suficientemente veloz.

Para la comunicación entre Arduinos se necesitan mandar de una DUE a la otra la posición (x, y) del robot periódicamente. Como la DUE que hace el procesamiento neuronal precisa completar todas sus funciones en menos de un periodo de integración del IK, es fundamental ajustar los tiempos empleados por las comunicaciones.

Como solución a este problema, se implementa una comunicación en paralelo utilizando los pines disponibles en ambas Arduinos. El planteo se justifica, siempre que sea posible escribir al mismo tiempo varios pines. Para esto fue necesario estudiar previamente la implementación del manejo de pines, incluida entre las librerías de la plataforma. Un estudio de tiempos sobre esta, concluyó que el manejo de pines utilizando dichas funciones no era suficientemente rápido. Es por esta razón que se procedió a estudiar alternativas para la manipulación directa de estas señales.

5.3.1. Manipulación de puertos

Se consiguió implementar rutinas de comunicación utilizando manipulación directa de puertos. El procesador prevé esto y agrupa muchos de sus pines en puertos de 32 *bits*. Es decir, que para estos se puede utilizar una sola instrucción y manipular a la vez todos las señales del puerto. Para muchos de estos *bits*, Arduino prevé una conexión directa a pines externos de la placa. Se estudiaron estas conexiones y se seleccionaron pines dentro de los puertos disponibles para facilitar la escritura y lectura de la posición por ambas DUE.

Dado que la arena es de 2×2 m, se eligió codificar la posición cada un centímetro, que es aproximadamente el doble de la precisión obtenida en la odometría. Por ende para cada dirección (x, y) se tiene mínimo dos bytes de información para mandar entre las DUE ($2^8 = 256$).

Los puertos utilizados para escribir y leer las posiciones son, el puerto C para la coordenada x, y el puerto A para la coordenada y. Sin embargo, estos puertos son de 32 *bits* (aunque no todos los *bits* estan disponibles como pines en la Arduino). Para facilitar la lectura y escritura de los bytes, así como las pruebas realizadas se

eligieron los pines correspondientes a *nibbles* de la palabra de 32 *bits*.

De esta manera por medio de máscaras y $shifts$ en las palabras leídas se puede reconstruir rápidamente lo escrito en los puertos.

Puertos Paralelos (A y C)

Nibble	bits	Uso
7	31..28	No conectado
6	27..24	SPI
5	23..20	No conectado
4	19..16	17 y 18 SDA1 y SCL1
3	15..12	High nibble pos x
2	11..8	RX0, TX0, RX1, TX1
1	7..4	No conectado
0	3..0	Low nibble pos x

Tabla 5.2: Descripción Puerto A

Cada puerto se puede manipular directamente como una palabra de 32 *bits*. Dado que para el Cortex [5] los enteros son de 32 *bits* se utiliza este tipo de dato como variable, en las Tablas 5.2 y 5.3 se muestran que *nibbles* de las palabras afectan en cada puerto a las señales utilizadas, estas se corresponden a su vez con los pines físicos descritos en el Apéndice **??**.

Cabe destacar que ninguno de los dos puertos utilizados contaba con la conexión física de los 16 *bits* necesarios, organizados correlativamente de forma que fuera sencilla su manipulación.

Nibble	bits	Uso
7	31..28	No conectado
6	27..24	No conectado
5	23..20	No conectado
4	19..16	High nibble pos y
3	15..12	Low nibble pos y
2	11..8	No conectado
1	7..4	Disponible
0	3..0	No conectado

Tabla 5.3: Descripción Puerto C

Se buscaron pines que fueran correlativos a nibbles de la palabra de 32 *bits* para facilitar la depuración y manejo de código. Es por esta razón que en las Tablas 5.2 y 5.3 se aclara alguno de los motivos por los cuales no fue usado determinado nibble. Cabe aclarar que al decir "No conectado"se refiere a alguno de los *bits* de la palabra pero no necesariamente a todos ellos, y significa que Arduino no cableo a sus pines físicos los correspondientes en el micro. Lo mismo sucede al aclarar la función asociada a determinado pin.

En la etapa de comunicación, se incluyó además, un pin de sincronización, cuya funcionalidad es la de generar una interrupción en la Arduino encargada de la red neuronal. Es de esta manera que se evita leer valores erróneos en los correspondientes puertos. Sería posible crear con este sistema, una comunicación bilateral entre los micros, es por esta razón que se incluyeron en ambos programas las funciones tanto de lectura como de escritura en los puertos. Aunque no se esté utilizando dicha funcionalidad en las presentes especificaciones, la misma fue probada.

Finalmente, es importante destacar, que esta comunicación se utilizó también para pasar de una DUE a la otra, la información de selección de ambiente.

Capítulo 6

Pruebas

En este Capítulo se detallan todas las pruebas realizadas en el proyecto. Como es habitual en los proyectos, y es una buena práctica en el desarrollo de los mismos, se fragmentaron el sistema y los problemas a resolver, en subsistemas, y se abordaron las pruebas pertinentes a cada uno para poder asegurar su correcto funcionamiento o resolución. Al hacer dichas particiones en subsistemas, se logra tener subproblemas a resolver que son mucho más fáciles de visualizar sus resoluciones y/o sus pruebas.

Cabe destacar, y se verá a lo largo de todo este Capítulo, el fuerte uso del lenguaje de programación *Python*. Ninguno de los integrantes del grupo de proyecto sabía programar tanto en ese lenguaje como en otros. Sin embargo, se decidió dedicar tiempo en aprenderlo ya que resultó muy útil para realizar algunas de las pruebas.

6.1. Pruebas de Subsistemas

En esta Sección se describen y detallan todas las pruebas concernientes a los subsistemas. Dentro de estos se encuentran: la odometría, la comunicación entre placas y los modelos neuronales.

6.1.1. Odometría

Para las pruebas en lo que respecta a la odometría, se precisaba tener a la plataforma moviéndose por la arena y al mismo tiempo enviando datos de su posición para luego poder analizarlos. Para ello lo primero que se hizo fue conseguir una torre de computadora, que cumplió el rol de proporcionar alimentación a las distintas placas así como también de receptora y capturadora de datos. Se utilizaron las varias salidas, tanto de 5 V como de 12 V, con las que contaba la fuente de PC. Se decidió tener alimentaciones de una misma fuente para tranquilidad de que todo estuviera alimentado con la misma referencia de tierra.

Luego de tener resuelta la alimentación, se trabajó en la obtención de datos de la DUE encargada de la navegación. Se procedió entonces al armado de un cable USB de un largo aproximado de 3 m. Lo suficientemente largo como para no tener problemas en el seguimiento del robot por la arena. Dicho cable se construyó en base a un UTP, donde una de sus terminales se conectó a los pines de USB internos de la PC, y en la otra se le colocó un conector USB tipo “hembra”. En este último se coloca el cable USB que termina de formar la conección entre la PC y la Arduino.

Después de tener resueltos, tanto la alimentación como la comunicación para obtener los datos, se procedió a realizar el experimento de navegación del robot y la recolección de datos de posición calculados en la misma. Se alimentaron los motores desde la PC con 12 V, la Arduino con el cable USB y se recolectaron datos de la posición enviados por serial hacia la PC. Luego de varios experimentos obteniendo distintos datos, se constató que no se estaban recibiendo bien los

datos. Existía una gran diferencia entre lo recibido y lo constatado visualmente en el recorrido del robot, por lo cual nos fue posible darnos cuenta de ese problema.

Descartando previamente que fuera problema con los trackers en el cálculo de la posición, se verificó que el problema era en el cable USB armado. Dada su longitud, no se recibían bien los datos. Se pensó entonces que el problema se daba por una falta de corriente desde la Arduino, ya que esta estaba alimentada por USB. Fue así que se decidió alimentar la Arduino desde la fuente de PC, quedando conectado el USB para la transmisión de datos. Esto último se hizo con una previa investigación. En la misma se supo que la placa Arduino contaba con elementos para control y protección de componentes cuando se alimentaba en simultáneo desde el USB y la entrada de alimentación. A pesar de esto, en una de las pruebas, se quemó una de las protecciones que se mencionan y no se pudo seguir experimentando de la manera planeada, aunque la placa Arduino siguió funcionando perfectamente. Fue entonces que se buscó una alternativa para el envío de datos, hubo que encontrar un nuevo canal de comunicación. Se optó por un dispositivo bluetooth.

Se consiguió uno en el mercado local y se estudió para poder configurarlo y utilizarlo. Este nuevo canal de comunicación dió la posibilidad de recolectar datos en las notebooks. Alternativa que al inicio no se tenía puesto que las notebooks cuentan con una fuente de alimentación diferente de la de la PC y eso puede traer problemas debido a la diferencia entre tierras de las mismas.

El cambio también tuvo influencia en lo que se refiere a hardware, ya que por un tema de espacio y comodidad para la colocación del bluetooth, se decidió situar este en la placa Arduino encargada de los modelos neuronales. Se tuvo entonces que solucionar la comunicación entre las dos Arduino para el correcto funcionamiento de las pruebas. La misma se solucionó utilizando los pines de los puertos paralelos, que se detallan en el Capítulo 5, y sus pruebas para validar el correcto funcionamiento se describen más adelante.

Para las pruebas, posteriores a todos los inconvenientes anteriormente descritos, se comenzó por verificar si el recorrido real del robot se correspondía con el obtenido graficando los datos. Las capturas de información enviada por bluetooth y los gráficos de posición los realizó un script en *Python*. Con alimentación en los motores de 12 V, se hizo al robot funcionar con su rutina aleatoria por la arena. Se programó la DUE de navegación para avanzar hasta que su coordenada x o y fuera un valor determinado, así como también se configuró para girar en torno a sí mismo hasta que su coordenada θ fuera otro valor definido.

Sin alimentación en motores se probó la odometría, moviendo manualmente al robot, recolectando los datos, en tanto se comparaban con la mediciones en el suelo de marcas y/o referencias.

Cabe destacar que en el transcurso de las pruebas la placa Arduino que había sufrido la avería, dejó de ser programable de un momento al otro y definitivamente. Para solucionar este problema se decidió comprar otra placa Arduino Due en un distribuidor local, lo que hizo que se obtuviera en un lapso de tiempo corto.

Se propuso luego hacer mover al robot en la arena virtual, detallada en la Sección 3.4, verificando la permanencia del robot en el área virtual y detectando cuando se saliera. El tiempo en el cual se mantenía dentro, depende de factores de la arena en la cual se mueva. Como pueden ser, rigurosidad, inclinación, imperfecciones, dimensión.

Con la arena virtual se procedió igual que con la no virtual en lo que respecta a la recolección y análisis de los datos, de igual manera que para las primeras pruebas, con un programa escrito en *Python* que se encargaba de procesar los datos.

Además de graficar la trayectoria del robot, se contrastaron estos resultados con la función $g(x, y)$ para verificar que las neuronas dispararan en los lugares correctos. El contraste mencionado se puede observar en la Figura 6.1.

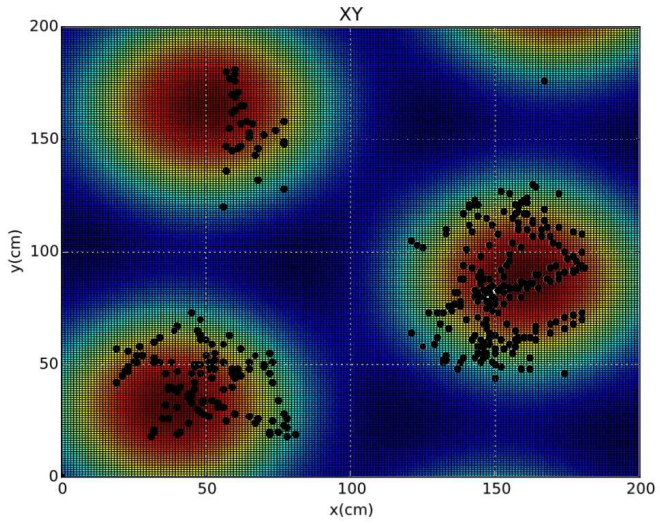


Figura 6.1: Tensión de membrana comparada con función $g(x, y)$

Para verificar la comunicación por los puertos paralelos entre las placas DUE, se programó a la Arduino Baja para que transmitiera un dato previamente conocido, y se constató que efectivamente lo recibido por la Arduino Alta fuera correcto.

Ajuste de la odometría

Para el ajuste de la cuadratura de los encoders se tomó como base la Figura 3.7. Por cada rueda, se convirtieron las salidas de los optoacopladores en un

número (0, 1, 2, ó 3) representativo del estado en que se encuentran. Así entonces, se movían manualmente las ruedas en una dirección, se obtenían por serial dichos estados y se ajustaba la posición de uno de los optoacopladores hasta obtener la secuencia adecuada correspondiente al movimiento realizado.

Se decidió luego de observar ligeros errores en la odometría, mejorar el procedimiento para cálculo de parámetros que intervienen en el modelo, como L y R. Primeramente se determinaron con regla o cinta métrica. Luego se planteó un análisis un poco más estadístico y con movimiento del robot donde se utilizaran los encoders para el cálculo.

Para la medición de L, se programó al robot para dar 10 vueltas completas, controlando por software que la variable θ sea menor a $10 \times 2\pi$. Se ajustó por aproximaciones sucesivas el parámetro L, de modo que se repitió el experimento anterior hasta que se observó en la práctica que el robot hubiera girado 10 vueltas completas. Durante el proceso se pudo verificar que el giro era muy sensible al parámetro L. De tal manera que se tuvo que modificar hasta las centésimas de nanómetros para lograr el comportamiento deseado.

Como un cambio de enfoque se decidió determinar directamente C_m, que es la relación entre la cantidad de pulsos y la distancia recorrida ($\frac{m}{pulso}$). El parámetro R no es explícitamente necesario sino que es un medio para determinar C_m. Hallando entonces directamente este parámetro se reduce el error introducido. En la Ecuación (3.1) se puede ver la relación que existe entre los parámetros mencionados.

Para determinar C_m, se hizo recorrer al robot en línea recta una distancia de $3m$ y se transmitió por serial la cantidad de ranuras del encoder que se contaron en esa distancia, por cada rueda. Teniendo en cuenta las Ecuaciones (3.2) y (3.3), sabiendo que $\Delta S_L = \Delta S_R = 3\ m$, que N_L y N_R son los transmitidos por serial, se despejaron los C_m correspondientes a cada rueda. El anterior procedimiento se repitió varias veces, para luego promediar y obtener un valor por rueda de C_m. Para unificar y obtener un único valor de C_m (esto me lo impone el modelo), se promediaron los últimos valores mencionados.

6.1.2. Modelo Neuronal

Luego de implementar en la Arduino los modelos neuronales, en las distintas etapas del proceso de optimización, se buscó verificar que dichas implementaciones fueran correctas. Para esta verificación se corroboraron la función $g(x, y)$ y la tensión de membrana v, para algunas de las células implementadas. Esto quiere decir que se compararon las obtenidas por la plataforma con las obtenidas a base a los modelos que se tenían al inicio del proyecto. Graficando con ayuda de herramientas tipo MatLab u Octave, los datos obtenidos experimentalmente y comparándolas con aquellas implementadas directamente en estos programas utilizando los mismos parámetros.

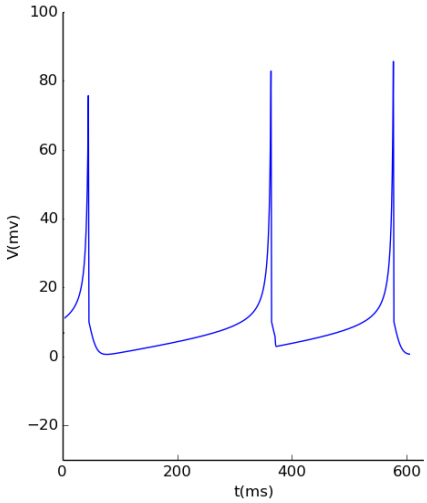


Figura 6.2: Tensión de membrana con paso de integración de 0.1 ms

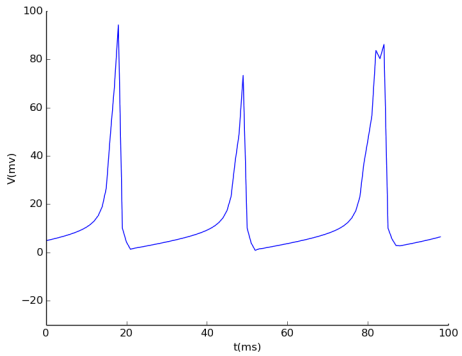


Figura 6.3: Tensión de membrana con paso de integración de 1 ms

Para la validación de la tensión de membrana, se realizaron tres comparaciones. Una que validara la forma de onda, con un paso de integración más chico que el que se iba a utilizar realmente. Otra que la validara en tiempo real, con el paso de integración adecuado. Y la última para validar que las tensiones se dieran en el lugar correcto de la arena. Para la última comparación se realizó el experimento de hacer navegar a la plataforma robótica por la arena de 2×2 m, enviando por bluetooth los datos de tensión de membrana. Mientras que para los casos de validación de forma de onda, se obtuvieron datos de una sola neurona pulsando, con la plataforma detenida en un único (x, y). El cálculo de v, para la primera comparación se hizo cada 0.1 ms, mientras que para los dos restantes el tiempo de integración era de 1 ms, siendo este último el valor a utilizar acordado en el inicio del proyecto. Los datos eran capturados por un programa escrito en *Python* que

además graficaba los datos obtenidos en función del tiempo y para la última de las pruebas se agregaba en código, lo necesario para que además se graficara el recorrido del robot en la arena. Los resultados de estos experimentos se pueden ver en las Figuras 6.2, 6.3. Comparando estas imágenes con la Figura 4.5 en el Capítulo 4 se tiene una validación cualitativa de la implementación. Esta información se puso a disposición del cliente para su evaluación.

Comunicación

Para validar que los datos calculados por software, eran efectivamente los que se enviaban por SPI, se contaba para recibir los datos con una placa Arduino Due (la utilizada para la navegación) y con una Arduino Mega. Es importante aclarar que la Arduino transmisora (la encargada de la Red Neuronal) debía ser configurada como Master y la receptora como Slave, puesto que el control de tiempos de la generación de neuronas lo tiene la placa que envía datos.

Primero se intentó utilizar como receptora de los datos a la DUE, ya que esta es capaz de manejar la misma velocidad de reloj que utiliza la Arduino Alta. Luego de investigar como hacer lo anterior, no se logró configurar dicha placa como Slave. Acción que sí se pudo concretar con la Mega. El problema que tuvo esta alternativa, fue que Arduino Mega no puede manejar velocidades de reloj tan altas como la DUE. Por este motivo se realizó una comprobación de datos del SPI en un experimento determinado y puntual, con la velocidad de reloj más alta que se pudo trabajar sin pérdida de datos en la Mega y corroborar así el correcto envío de datos.

El experimento constó en enviar 27 *bytes* a 1 MHz de la DUE a la Mega por el bus SPI y enviar por serial a una PC lo recibido por la última. Se enviaron 27 caracteres conocidos (a, b, c, ...), y se validó así la correcta comunicación entre placas. Verificando de esta manera, que la comunicación SPI estaba configurada de acuerdo a lo deseado.

Validación Modelo Neuronal

Para la validación final de los datos obtenidos de los modelos neuronales implementados por la plataforma robótica, se optó por entregar varios archivos de texto plano con información de la posición del robot y los spikes de varias neuronas (1 si la neurona dispara en la posición indicada o 0 si no lo hace), al Ing. Biomédico Javier Cuneo para que este los procesara. Con los datos se obtiene un número elevado de simulaciones para el ambiente (uno elegido), con los cuales distribuye los disparos de las neuronas en cada lugar del ambiente y realiza un mapa de frecuencias para cada neurona. Este último resultado se corrobora con lo que cada neurona debería entregar en el ambiente.

El formato de los diferentes archivos se puede ver en la Figura 6.4, que fue acordado con Cuneo.

$$
\begin{array}{rcl}
x_0, y_0, spike_{neurona_1}, spike_{neurona_2}, & \cdots & , spike_{neurona_{n-1}}, spike_{neurona_n}; \\
x_1, y_1, spike_{neurona_1}, spike_{neurona_2}, & \cdots & , spike_{neurona_{n-1}}, spike_{neurona_n}; \\
 & \vdots & \\
x_m, y_m, spike_{neurona_1}, spike_{neurona_2}, & \cdots & , spike_{neurona_{n-1}}, spike_{neurona_n};
\end{array}
$$

Figura 6.4: Formato de los archivos utilizados para la validación de los modelos neuronales.

Para la generación de los diferentes archivos anteriormente mencionados, se realizaron varias repeticiones del siguiente procedimiento. Se colocó al robot en la arena, partiendo de un punto inicial elegido, $(x_0, y_0) = (40, 40)$, $\theta_0 = 0$, y se le programó moverse aleatoriamente por la arena mientras la plataforma enviaba por Bluetooth, cada 1 ms los datos que generaba. Estos últimos eran capturados por un *scirpt* escrito en lenguaje *Python*, que era el encargado de escribir en un archivo los datos que recibía, en el formato adecuado.

Cada envío comenzaba primero con una sincronización entre transmisor y receptor con el envío del caracter 255. El caracter de sincronización corresponde a un *byte* de todos 1's. Se eligió de esta manera porque representa una situación que no puede darse en las pruebas. Es decir que 8 neuronas consecutivas disparen en un punto de la arena o que alguna de las coordenadas cartesianas sea 255. La última situación no puede darse por las dimensiones de la arena, y la primera es muy poco probable por los resultados vistos a lo largo de todo el proyecto.

El procedimiento terminaba al cumplirse 5 $mins$ de iniciado y se repetía nuevamente generando un nuevo archivo. El número de neuronas de las cuales se obtuvieron los spikes, fue 56. Este número fue determinado como el número máximo de neuronas que se podían utilizar para enviar cada 1 ms los datos de posición y spikes de las neuronas, calculando por cada paso estos últimos, a una velocidad serial de 115200 bps, con una sincronización entre el envío de datos y el programa capturador en cada paso. La velocidad serial fue elegida de modo que fuera la mayor posible, para el dispositivo de comunicación bluetooth, en la que se pudo constatar que los datos recibidos por el capturador no tenían errores.

Cabe aclarar, que el hecho de contar con un menor número de neuronas, que en el caso general con comunicación SPI, se debe al tiempo consumido por las librerías que implementan la comunicación serial tipo UART. Para obtener el máximo número de neuronas posibles, utilizando este mecanismo, se procedió iterativamente a eliminar el cómputo de algunas neuronas, reduciendo a la vez, el tiempo

total de cálculo, y el tiempo consumido por la comunicación.

No se validaron los modelos neuronales utilizando la salida del bus SPI porque no se contaba, al momento de las validaciones, con algún método para capturar los datos del SPI, a la velocidad que este trabaja. Se intentó usar una placa modelo CP2130, la cual en el tiempo que se investigó se llegó a la conclusión de que no se podía usar para lo que se necesitaba puesto que no era configurable en modo Slave para recibir datos por el bus SPI, ni tampoco era viable configurarla en modo Master y obtener de la DUE su reloj para el manejo de tiempos. En este caso el reloj que maneja los tiempos, para el correcto funcionamiento de los modelos neuronales es la Arduino, por esto se buscó el transmitir la información de reloj de la Arduino al CP2130.

No obstante, después de enviados los datos adquiridos al cliente para su análisis, se realizó un procesado de los datos, para algunas de las células, con la ayuda de una planilla de cálculo. Se presentan a continuación los resultados obtenidos.

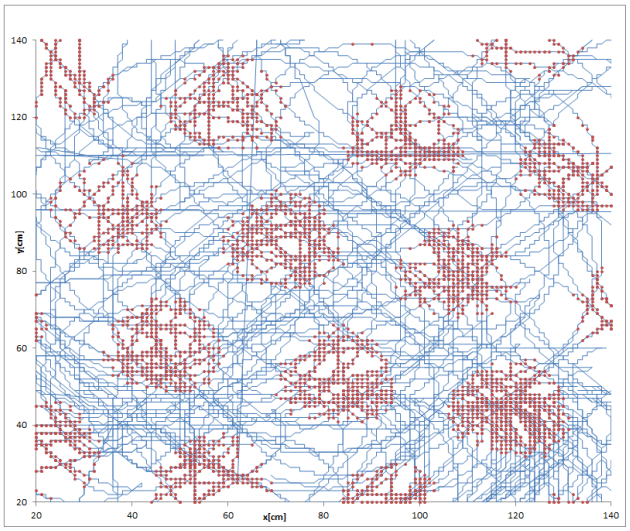


Figura 6.5: Resultados experimentales de una neurona con $\lambda = 2.69\ m$

La Figuras 6.5 y 6.6 muestran la superposición de tres experimentos de 5 minutos cada uno, para distintas *grid cells*, en rojo se pueden observar los puntos en los que se verificó un *spike* y en azul la trayectoria del robot. A simple vista se puede observar la similitud con las imágenes presentadas en el Capítulo 4, Sección 4.1.2, ya sea las de la función $g(x, y)$ en la Figuras 4.3.(a) y (b) o la de la actividad real en una rata en la Figura 4.1.

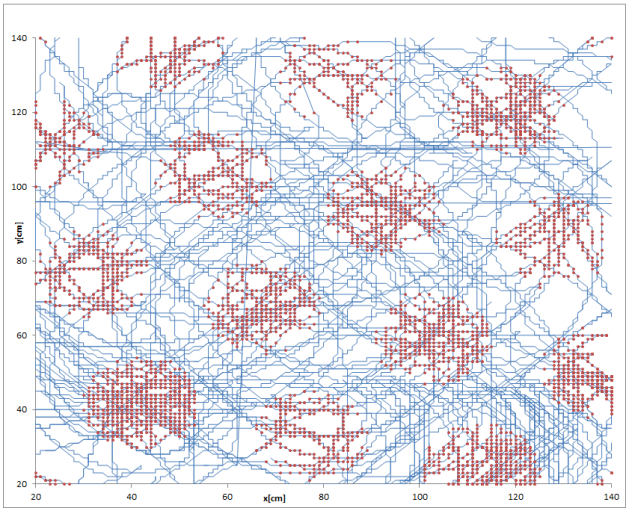


Figura 6.6: Resultados experimentales de una neurona con $\lambda = 2.74\ m$

Capítulo 7

Conclusiones

Capítulo 7. Conclusiones

Durante el desarrollo de este documento se abordan los detalles del proyecto, comentando los desafíos encontrados y las soluciones aplicadas. En este Capítulo se exponen las conclusiones alcanzadas y algunas reflexiones realizadas. Para comenzar se recuerda el objetivo principal del proyecto.

"Desarrollar un robot demostrador que incorpore uno de los modelos de sistemas neuronales propuestos por IUEMHI para que pueda someterse a operación en un ambiente real."

Analizando únicamente el objetivo general del proyecto se considera que este fue alcanzado. El robot desarrollado implementa los modelos propuestos dentro de las limitaciones del hardware seleccionado. Estas limitaciones se pueden resumir en dos grandes grupos, limitaciones en la odometría implementada y limitaciones en el poder de cálculo de los procesadores utilizados.

En cuanto a la odometría la limitación es principalmente el error. Como se describe en el Capítulo 3, este error es acumulativo y tarde o temprano resulta en que se pierda la correspondencia entre la posición supuesta y la real. La persistencia y variabilidad de este fenómeno hace que sea muy difícil corregirlo. Existe bibliografía que aborda el tema del error en sistemas con ruedas motorizadas controladas con odometría, detallando métodos que reducen este error. Este fue definitivamente de los más grandes problemas que se encontraron y afinar el funcionamiento del sistema hasta lograr un rendimiento aceptable fue la tarea que insumió la mayor cantidad de tiempo en el proyecto.

Por otro lado, la limitación de la capacidad de los procesadores fue mucho más evidente. Para procesar neuronas en tiempo real se deben realizar todos los cálculos necesarios en pasos muy cortos de tiempo. Para mejorar en este aspecto existen dos caminos claros, uno es profundizar la optimización en la implementación de las operaciones correspondientes para esta plataforma, y el otro es mejorar la tecnología. En cuanto a la optimización, se utilizó una porción considerable del tiempo del proyecto para estudiar e implementar, hasta alcanzar una barrera en que las mejoras dejaron de ser significativas para la cantidad de neuronas modelada (en el orden de los $900\mu s$ para procesar las 200 neuronas y $100\mu s$ para comunicarlas junto a la posición). La tecnología es una limitante a estudiar ya que implica considerar múltiples variables, por ejemplo el peso, costo, prestaciones y disponibilidad.

Recapitulando entonces, los dos pilares de este proyecto son la odometría y el modelado neuronal. Ambos fueron desarrollados para cumplir con el alcance y los objetivos del proyecto, aunque constituyen simplemente una plataforma sobre la cual se puede aún construir mucho más. Mejorar la odometría y extender, ampliar o modificar los modelos neuronales implementados, son caminos claros de mejora para el futuro.

Particularmente para el sistema de odometría, se plantean los siguientes puntos

que pueden mejorar en gran medida el desempeño. En primer lugar, la incapacidad de los motores de lograr el par necesario para un desplazamiento adecuado a velocidades bajas, es muy limitante. Cambiarlos por componentes más adecuados, como por ejemplo motores paso a paso o motores DC más sofisticados, es la solución evidente.

También se pueden mejorar los resultados si se coloca un encoder que cumpla la función de los dos optoacopladores en cuadratura en un solo encapsulado. Este tipo de componentes se encuentran en el mercado, y funcionan con dos sensores alineados y un disco con ranuras en cuadratura. El error que introduce la diferencia de tamaño entre las parte huecas y sólidas en el disco ranurado puede ser significativa. Además el método con dos sensores es limitado en su robustez, ya que un desajuste en uno de los elementos puede resultar en que se salgan de cuadratura y las lecturas en el ángulo de giro de la rueda sean erróneas.

Otro camino a seguir es agregar marcadores que permitan al robot, actualizar su posición en ciertos puntos del ambiente. Esto interrumpe la acumulación del error, incrementando el tiempo de funcionamiento exitoso de la plataforma.

Finalmente, incorporar otros sensores que permitan al RattusBot afinar su posición puede mejorar también el sistema. Depender exclusivamente de la odometría es excesivamente limitante. Se encontraron trabajos recientes relacionados con un tipo de odometría basados en medios visuales. Si bien esta no es una opción a realizar con las plataformas Arduino actuales, debido sobre todo a la poca memoria con la que cuentan, se recomienda indagar la posibilidad de incorporar dicho procesamiento a la hora de considerar nuevo hardware para la implementación de este sistema [4].

Debido al alto nivel de portabilidad e intercambiabilidad de todos los componentes de la plataforma, incorporar o retirar piezas es una tarea fácilmente realizable. Debido a este criterio de diseño, con tan solo aflojar algunos tornillos y desconectar los cables adecuados se puede remover cualquiera de estos componentes. Esto le da gran escalabilidad a la plataforma y facilita pensarla a futuro.

En cuanto al proyecto en sí, se valora como una experiencia muy provechosa y enriquecedora, desde el punto de vista académico y profesional. La gran cantidad de disciplinas y especialidades abordadas durante el desarrollo del proyecto hizo que resultara muy interesante enfrentar cada desafío y permitió aplicar muchos de los conocimientos adquiridos durante la carrera.

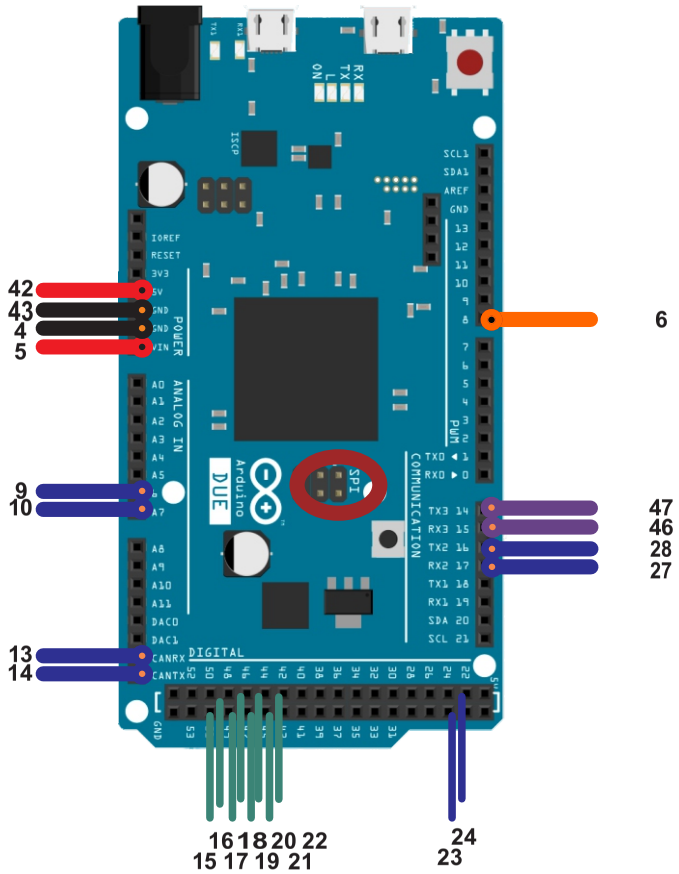
Se destaca la esencialidad de una buen planificación y particularmente de una buena especificación del alcance y los objetivos. Se vivió en carne propia el aspecto crucial y los retrasos que puede implicar en el desarrollo de un proyecto de mediano plazo, no tener objetivos claros y bien definidos.

Como última reflexión se valora la importancia del trabajo en equipo, que es determinante a la hora de hacer frente a las dificultades encontradas en una experiencia de esta magnitud. Este aspecto es tal vez el más importante, ya que es la sinergia, que se logra con la cooperación, la que permite abatir los ires y venires encontrados en este tipo de actividad.

Apéndice A

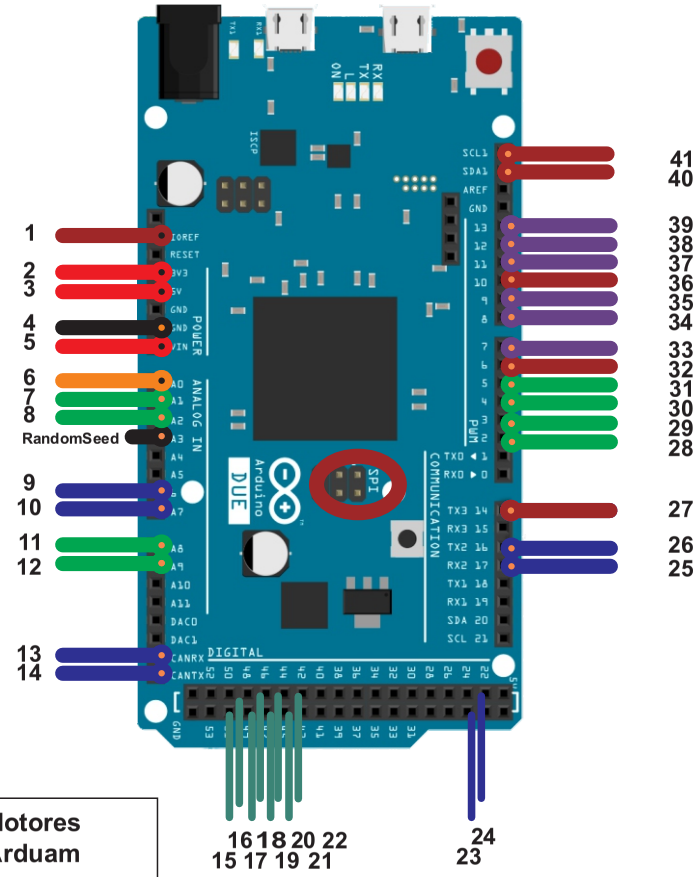
Diagramas de Conexionado

Arduino Alta



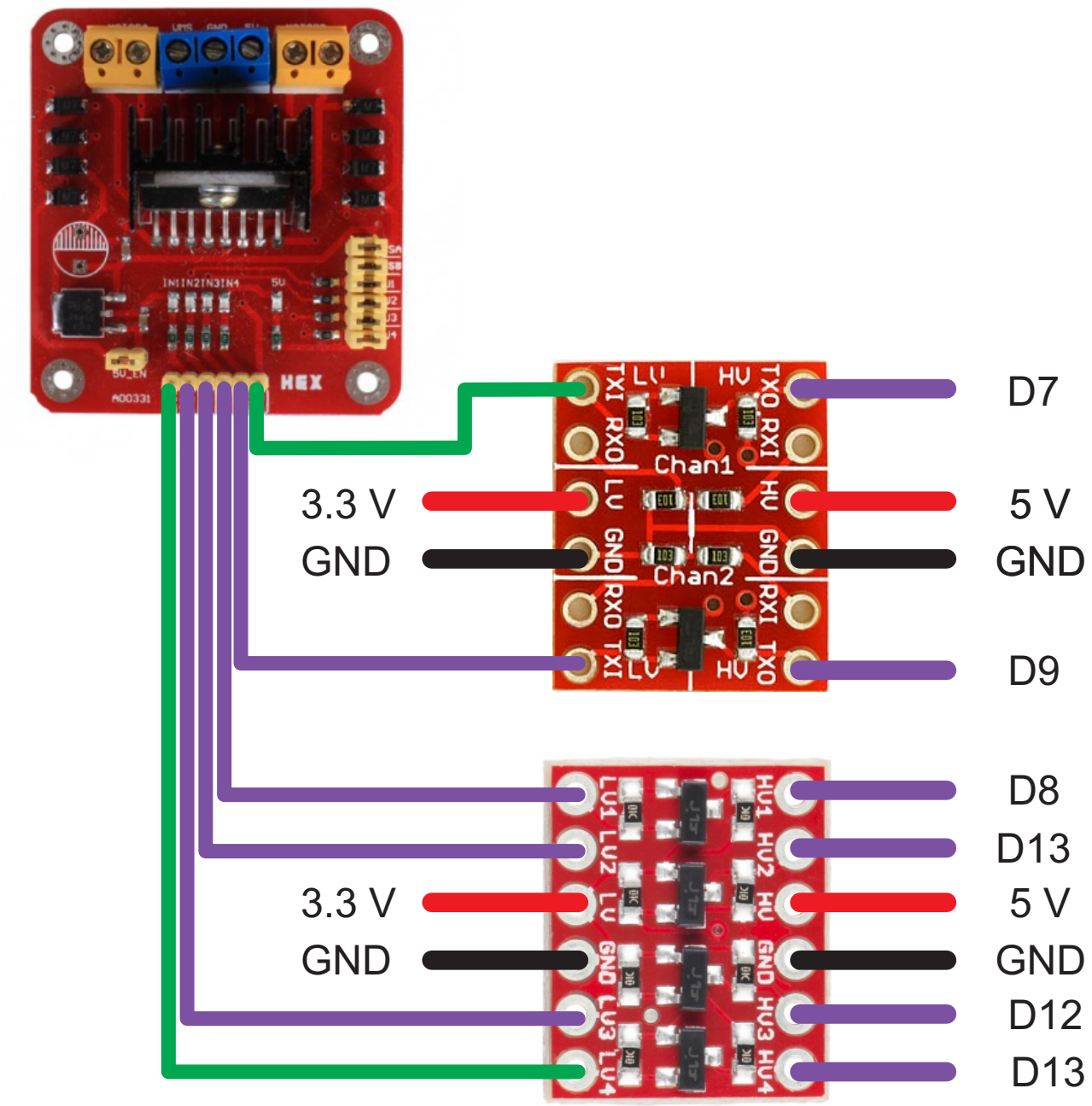
BlueTooth
PuertoA
PuertoC
Alimentacion
Gnd
SincCom

Arduino Baja

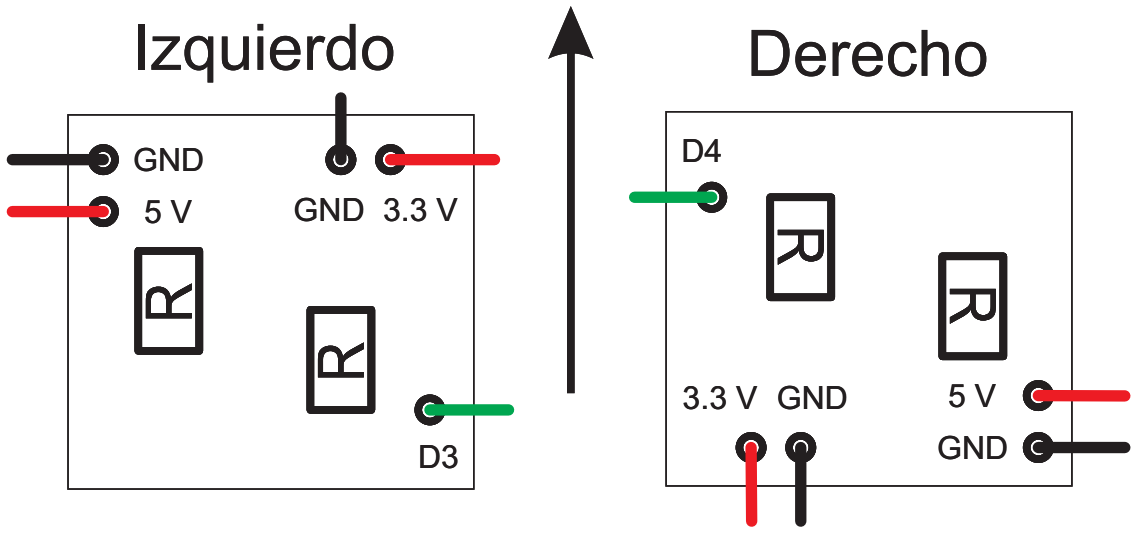


Motores
Arduam
Sensores
PuertoA
PuertoC
Alimentacion
Gnd
SincCom

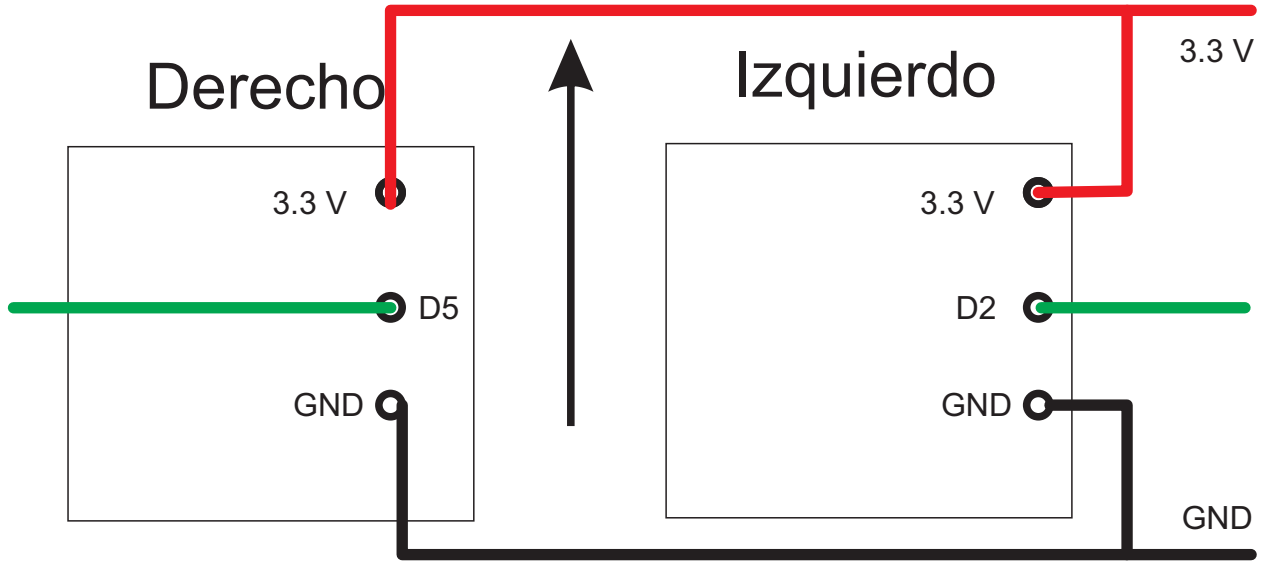
Cable	Nombre	Origen	Destino	Pin Destino
1	Ioref	DueBaja	ArducamShield	Ioref
2	3.3V	DueBaja	BarraAlim	
3	5 V	DueBaja	BarraAlim	
4	GND	DueBaja	BarraAlim	
5	Vin	DueBaja	DueAlta	5
6	SincCom	DueBaja	DueAlta	6
7	A1	DueBaja	Tilt Derecho	
8	A2	DueBaja	Tilt Izquierdo	
9	PuertoA	DueBaja	DueAlta	9
10	PuertoA	DueBaja	DueAlta	10
11	A8	DueBaja	BumperDerecho	
12	A9	DueBaja	Bumper Izquierdo	
13	PuertoA	DueBaja	DueAlta	13
14	PuertoA	DueBaja	DueAlta	14
15	PuertoC	DueBaja	DueAlta	15
16	PuertoC	DueBaja	DueAlta	16
17	PuertoC	DueBaja	DueAlta	17
18	PuertoC	DueBaja	DueAlta	18
19	PuertoC	DueBaja	DueAlta	19
20	PuertoC	DueBaja	DueAlta	20
21	PuertoC	DueBaja	DueAlta	21
22	PuertoC	DueBaja	DueAlta	22
23	PuertoA	DueBaja	DueAlta	23
24	PuertoA	DueBaja	DueAlta	24
25	PuertoC	DueBaja	DueAlta	27
26	PuertoC	DueBaja	DueAlta	28
27	D14	DueBaja	Leds	
28	D2	DueBaja	Opto IzqCuadratura	
29	D3	DueBaja	Opto Izquierdo	
30	D4	DueBaja	Opto Derecho	
31	D5	DueBaja	Opto DerCuadratura	
32	SPI_CS	DueBaja	ArducamShield	Pin D9
33	D7	DueBaja	Conv UART	EnB
34	D8	DueBaja	Conv Bidireccional	In3
35	D9	DueBaja	Conv UART	In4
36	D10	DueBaja	ArducamShield	Pin D10
37	D11	DueBaja	Conv Bidireccional	EnA
38	SDA	DueBaja	ArducamShield	SDA
39	SCL	DueBaja	ArducamShield	SCL
40	D12	DueBaja	Conv Bidireccional	In1
41	D13	DueBaja	Conv Bidireccional	In2
42	5 V	DueAlta	BlueTooth	Vcc
43	GND	DueAlta	BlueTooth	Gnd
44	Rx	DueAlta	BlueTooth	Rx
45	Tx	DueAlta	BlueTooth	Tx


IN1IN2IN3IN4
5V
HEX
Chan1
Chan2
TXI RXO LV
HV TXO RXI
GND
3.3 V
GND
D7
5 V
GND
D9
D8
D13
3.3 V
5 V
GND
GND
D12
D13

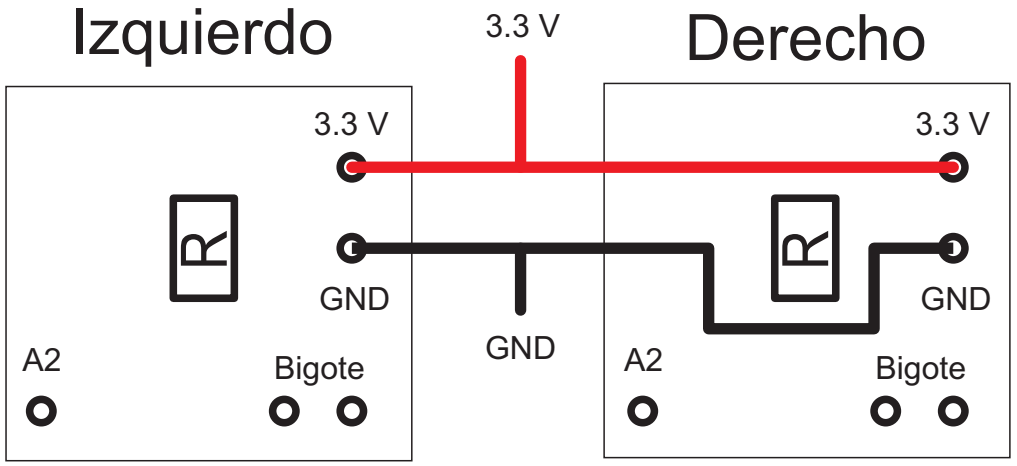
Optoacopladores



Optoacopladores Cuadratura (vista desde abajo)



Bigotes



Apéndice B

Plan Proyecto

Proyecto de fin de estudios en la carrera Ingeniería Eléctrica

Plan de Proyecto

Autores:

Nicolás Blanco
Mariana del Castillo
Joaquín Quagliotti

Tutor:
Leonardo Barboni
Co-tutor:
Pablo Monzón

09/05/2014

Resumen

• Estudiantes

Nicolás Javier Blanco Antonini
C.I. 4.221.451-8 , Cel. 099032734, blanco.nicolas.c@gmail.com

Mariana del Castillo Larumbe
C.I. 4.192.483-7 , Cel. 099607323, mdelcalaru@gmail.com

Joaquín Quagliotti Chanes
C.I. 4.408.029-8 , Cel. 098383019, joaquin.quagliotti@gmail.com

• Clientes

- Ing. Biomédico Javier Cuneo (Argentino, IUEMHI , estudiante de Doctorado del Grupo de Microelectrónica , DA, Co-DT Fernando Silveira).
- Dr. Pablo Francisco Argibay (Argentino, IUEMHI , co-tutor de tesis de Doctorado de Javier Cuneo).
- Grupo de Microelectrónica del IIE (FING-UdelaR)

• Tutor

Dr. Ing. Leonardo Barboni

• Fecha prevista de finalización

1 de Abril del 2015

• Total de horas a realizar previstas por el grupo de proyecto

Se estima 1600 horas hombre a lo largo del proyecto

• Fecha y descripción de los entregables intermedios

Hito 1 *(15/9/14)*:

En este hito, se presentará el robot con funcionalidades básicas de movimiento implementadas pero sin interactuar aún con la red neuronal.

Se demostrará el funcionamiento de los sensores seleccionados y la adquisición de datos de los mismos desplegando la información obtenida de una manera verificable.

En esta etapa el código del algoritmo de la red neuronal estará implementado pero no será posible mostrar su funcionamiento facilmente.
Se presentará documentación del código implementado hasta el momento.

Hito 2 *(15/2/15)*:

En este hito se presentará el robot con todas sus funcionalidades implementadas. Al momento de este hito el equipo estará en la última etapa de prueba y depuración por lo que no se puede garantizar el correcto funcionamiento de las partes interactuando entre sí.

Se demostrará el funcionamiento del sistema de navegación

Se mostrará el funcionamiento de la red neuronal en función de su salida frente a un cierto estímulo.

Se presentará documentación del código implementado hasta el momento.

2. Descripción del Proyecto

Una grid-cell es un tipo de neurona que se encuentra en cerebros de roedores (concretamente en la corteza entorrinal) **[1]**. Estas neuronas generan campos de disparos (potenciales de acción) en conjunto con sus neuronas vecinas las cuales codifican la representación del ambiente donde el roedor se mueve. Una de las particularidades de las grid-cells es que se encuentran formando patrones regulares de grilla que no tienen nada que ver con la regularidad del ambiente o la forma en que procesa la información sensorial. Como sistema dinámico su funcionamiento no está entendido completamente y la hipótesis más aceptada es que son "sintéticas a priori", es decir, son estructuras innatas. Por otro lado, en la corteza entorrinal también se encuentran otro tipo de neuronas llamadas place-cells **[1] [2]** que también codifican la posición del animal pero sin capacidad para crear rutas de movimiento en el ambiente. Se estima que ambos tipos de redes neuronales trabajan en forma conjunta para generar un mapa cognitivo que representa la localización del animal en un ambiente con relación a otros objetos.
Sin embargo, esto implica que los pesos sinápticos se deben aprender con cada nuevo ambiente y no soportarían una acción de navegación efectiva en ambientes no visitados previamente. En la realidad esto no sucede y entender como es el mecanismos de auto-localización y navegación vectorizada en el ambiente es un tema abierto de investigación de gran interés para los cuales muchas explicaciones y

modelos han sido propuestos **[3]**. Todo parece sugerir que el animal siempre viaja por la ruta más corta entre dos puntos del ambiente (origen y destino), aunque el ambiente sea novedoso y sin que la red neuronal esté entrenada en ese ambiente ni en ninguno similar. Sumado a esto, recientemente se encontró que existen varias capas de estas redes, algunas de las cuales solo se disparan al ser moduladas por la posición de la cabeza del roedor mientras que otras codifican distancias. En este contexto, el IUEMHI (Instituto Universitario Escuela de Medicina del Hospital Italiano – Bs. As.) realiza investigaciones desarrollando modelos de estos sistemas neuronales, a los que inclusive han incorporado conceptos de neurogénesis (i.e. creación de neuronas no entrenadas y por lo tanto de gran plasticidad que se incorporan a la red para colaborar en situaciones novedosas).
Se identifica la necesidad de contar con una plataforma hardware móvil compacta y de fácil configuración (robot), que contenga modelos de estos sistemas neuronales biológicos programados, así como también recursos para procesamiento de información sensorial. Este robot podrá ser usado por neurocientíficos para estudiar y entender el comportamiento de estas redes como sistemas dinámicos y cual es el nivel de autonomía que se puede alcanzar (por autonomía se entiende habilidad de construir sus propias reglas para solucionar problemas y realizar tareas).

3. Objetivo General

Desarrollar un robot demostrador que incorpore uno de los modelos de sistemas neuronales propuestos por IUEMHI para que pueda someterse a operación en un ambiente real.

4. Alcance

El robot demostrador se construirá con el kit "All-Ready Arduino Robot Chassis Kit" [4]. Se debe seleccionar una plataforma hardware de entre las indicadas en la sección de supuestos. En esta plataforma se programará :

- Control de los motores.
- Control de sensores y procesamiento digital de las señales para adecuarlas en modo de que puedan ser tomadas como entrada para los algoritmos que implementan los modelos de redes neuronales; se estima que se usará un sensor de imagen (cámara) o un conjunto de sensores de intensidad de luz.

- Modelo de las redes neuronales: es tomado de las investigaciones previas de los otros actores en este proyecto. Algunos algoritmos que lo implementan ya están dados en Matlab. Se deberá trasladar la implementación de dicho modelo a un lenguaje adecuado para el procesador elegido, considerando inclusive realizar modificaciones menores en los algoritmos para lograr la implementación, pero sin perder las propiedades del modelo.
- Navegación: la salida de los modelos de redes (las cuales son alimentadas con la información del ambiente que proviene de los sensores) se utilizará como entrada para un algoritmo de navegación, el cual estará conectado con el control de los motores. Este algoritmo debe ser construido, partiendo de los básicos propuestos en **[3]**.
- Se construirá un ambiente o arena de 2m x 2m con paredes que limiten el área de acción del robot y este podrá tener dibujado patrones de imágenes.

5. Criterios de éxito

Los indicadores de éxito son los siguientes:

- Lograr implementar los modelos de redes neuronales propuestos en el robot.
- Lograr que el robot sea capaz tomar decisiones y realizar tareas simples en forma autónoma. Como mínimo se espera que el robot pueda navegar de un origen a un destino (lo que representa la acción de buscar su madriguera o comida).
- Satisfacer las necesidades de los investigadores en neurociencias también actores en este proyecto implementando al menos un modelo de redes neuronales propuestos en el robot.

6. Actores

- Equipo de Proyecto (estudiantes)
- Dr. Ing. Leonardo Barboni (Tutor de Proyecto).
- Dr. Ing. Pablo Monzón (Co-Tutor de Proyecto).
- Ing. Biomédico Javier Cuneo (Argentino, IUEMHI , estudiante de Doctorado del Grupo de Microelectrónica , DA, Co-DT Fernando Silveira).
- Dr. Pablo Francisco Argibay (Argentino, IUEMHI , co-tutor de tesis de Doctorado de Javier Cuneo).
- Grupo de Microelectrónica del IIE (FING-UdelaR).
- Docentes del curso de gestión (M. Barreto y G. Eirea)

7. Supuestos

- Que los algoritmos dados que implementan los modelos propuestos de las redes son válidos y van a poder ser programados en alguna de las plataformas hardware mencionadas en la lista de hardware disponible.
- Esta disponible el siguiente hardware para usar:
    1. Un chasis de robot como el indicado en **[4]**
    2. ArduinoMega (2 placas) y arduino uno (1 placa)
    3. Rasperry Pi (1 placa) con cámara compatible
    4. Arduino due (1 placa)
    5. Una placa Arducam-LF Camera module shield
    6. 3.2 inch LCD for Arduino MEGA2560/ DUE
    7. Una placa Arducam-F Camera module shield for arduino UNO MEGA2560 / DUE
    8. Dos 5 Mega pixel Camera Module OV5642 1080P JPEG Output (compatibles con Arducam)

- Para adaptar las imágenes a entradas útiles para las redes se podrán usar librerías openCV **[5]** programables en C en la plataforma hardware seleccionada.
- Los integrantes del equipo de proyecto tienen la formación de cumplir con el objetivo planteado.
- El robot trabajará conectado a una fuente de alimentación de tensión dedicada, por lo que el bajo consumo o logro de autonomía energética no es un problema a solucionar.
- En primera instancia no se pretende que el robot identifique los bordes de la arena (espacio de trabajo, de 2m x 2m).

8. Restricciones

- El proyecto debe finalizar a mas tardar el 30 de Abril del 2015.
- El grupo de proyecto tiene una disponibilidad de 30 hs. semanales (10 hs de dedicación semanal continua por estudiante durante un año).
- La placa microcontroladora seleccionada deberá ser un Arduino Due ó Raspberry Pi (por la capacidad de cálculo requerida estimada para desarrollar la

aplicación). En caso de necesidad se podrá incorporar una placa extra como la Arduino Mega (para solucionar problemas de compatibilidad de sensores).

- El robot será alimentado por una fuente de tensión externa y se moverá en una arena con piso liso de 2m x 2m donde se establecerán patrones de imágenes y/o iluminación que el robot tomará como estímulos para las redes neuronales y para navegar por dicho ambiente tomando decisiones autónomas. Se podrá evaluar cómo se desempeña el robot para rodear obstáculos.

9. Especificación funcional del Proyecto

El RattusBot deberá ser capaz de navegar en una arena de 2x2 metros de manera autónoma, usando como mecanismo de navegación los algoritmos de redes neuronales.

La Figura 1 muestra un diagrama de bloques del sistema. Se indican los sensores que se definirán posteriormente y es posible que se integren periféricos adicionales (shields, otros sensores y actuadores adicionales).

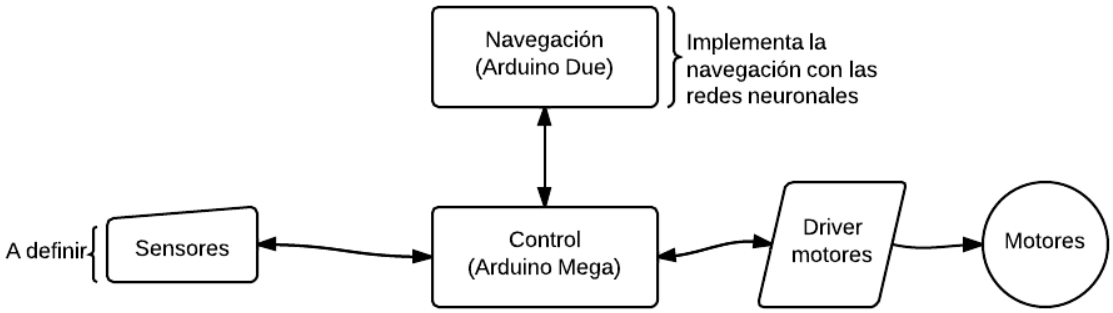


Figura 1 - Diagrama de bloques.

10. Objetivos específicos

- Armar mecánicamente el kit robot y lograr funcionalidades básicas
    - ❏ Entregables:
        - Robot armado.
        - Código que implementa funciones básicas.
        - Demo de las funcionalidades básicas.
- Selección de la plataforma hardware-software e implementación de las redes neuronales en la plataforma elegida.
    - ❏ Entregable:
        - Código de la implementación de las redes neuronales.
- Implementar todo el sistema logrando que interactúen todos los subsistemas, o sea, las señales de los sensores entrantes a la red y sus salidas. A su vez entrantes al bloque de control de los motores para lograr que cumpla exitosamente una tarea de navegación.
    - ❏ Entregable:
        - Código de integración de los subsistemas.
- Armado de la arena.
    - ❏ Entregable:
        - Arena armada.
- Integración, prueba y depuración del sistema completo.
    - ❏ Entregable:
        - Demo del sistema completo.
- Documentación del proyecto.
    - ❏ Entregable:
        - Documentación final.

11. Tareas

- Armado mecánico del robot y funcionalidades básicas:
    - Estudiar los manuales del chasis
    - Montar mecánicamente sus piezas.
    - Implementar el ajuste mecánico de los sensores (por ejemplo de la cámara) y de la plataforma hardware seleccionada al robot.
    - Implementar rutinas básicas de control de movimiento. Lograr avanzar, retroceder y girar con algún control cinemático básico sin uso de la red neuronal o información del ambiente.
    - Armado de notas para la documentación final.

- Selección de hardware:
    - Seleccionar la plataforma hardware de entre las listadas en la sección de supuestos.
    - Estudiar hojas de datos y manuales de todo el hardware seleccionado (incluyendo sensores, motores y su placa controladora y placa hardware con microprocesador).
    - Estudiar tanto el lenguaje como el ambiente de programación de dicha plataforma hardware.
    - Armado de notas para la documentación final.

- Implementar la red neuronal en la plataforma hardware-software elegida
    - Estudiar elementos de dinámica de sistemas, particularmente de redes neuronales.
    - Estudiar el algoritmo que será suministrado y el cual implementa un modelo de red neuronal. Una implementación ya esta en Matlab para estudiar y adaptar.
    - Implementar el código en el lenguaje elegido. Realizar las adaptaciones mínimas para lograr la implementación.
    - Probar y depurar.
    - Implementar la arena de 2m x 2m:
        - Construir el piso y los bordes
        - Pintar patrones de imágenes en el piso y paredes si se usa una cámara como sensor de visión.

      Si se usan sensores de intensidad de luz, se colocarán leds para codificar el ambiente, a los cuales se construirá el driver electrónico para modular la intensidad de luz (el tipo de patrones a colocar debe ser estudiado).
    - Armado de notas para la documentación final.

- Implementar bloques de control y el flujo de información requerido, esto es que debe exister una conexión entre señal de los sensores, la red neuronal y el control de los motores. Sin la interacción de todos los bloques entre sí, el robot seguramente no va funcionar
    - Estudiar señales de salida de los sensores y como adaptarlos para que sea el estímulo de la red neuronal.
    - Implementar los algoritmos que transformen las salidas de la red neuronal a señales aptas para el control de los motores del robot.
    - Construir el sistema de navegación que usa la información del ambiente codificada en las redes neuronales.
    - Probar y depurar.
    - Armado de notas para la documentación final.

- Integración, prueba y depuración del sistema completo
    - Implementar algoritmos de integración y control de las funciones del robot.
    - Ensamble final de robot con red neuronal implementada y asociada a sensores de estímulo y control de motores.
    - Probar y depurar.
    - Armado de notas para la documentación final.

- Documentación
    - Recopilación de las notas realizadas durante el transcurso del proyecto.
    - Preparación de las dos presentaciones de hitos del curso de gestión
    - Redacción y edición.
    - Revisión y corrección.

Resumen de Tareas y Asignación de Recursos:

Taréa	Duración (Dias)	Recursos
Estudiar los manuales del chasis	5	Equipo completo
Montaje de piezas	5	Mariana
Ajuste mecánico de sensores y micro	4	Mariana
Implementación de rutinas básicas de control	30	Equipo completo
Selección de plataforma	1	Equipo completo
Estudio de hojas de datos y manuales	5	Equipo completo
Estudio del lenguaje y del ambiente	5	Equipo completo
Estudio de elementos de dinámica de sistemas	8	Equipo completo
Estudio de algoritmo suministrado	5	Equipo completo
Implementación de código en el lenguaje elegido	10	Equipo completo
Probar y depurar	20	Equipo completo
Armado de la arena	4	Nicolás
Estudio de señales de salida de los sensores	5	Equipo completo
Implementación de los algoritmos de ajuste de entradas al motor	15	Equipo completo
Construcción del sistema de navegación	20	Equipo completo
Prueba y depuración	30	Equipo completo
Implementación de algoritmos de integración	20	Equipo completo
Ensamble final	13	Equipo completo
Prueba y depuración	60	Equipo completo
Recopilación de notas	5	Equipo completo
Presentación final	27	Equipo completo

Hito 2	3	Joaquín
Hito 1	3	Joaquín
Redacción y edición	20	Equipo completo
Revisión y corrección	14	Equipo completo

12. Cronograma detallado del Proyecto

Se adjunta el cronograma detallado del proyecto en el Apéndice A.

13. Análisis de Costos

Este proyecto no involucra grandes costos, el hardware ya fue comprado por el IIE, incluidos repuestos para los componentes de mayor probabilidad de averías. Por ésta razón solamente se desarrolla un análisis a modo de ejercicio, para observar los costos que involucraria el mismo de ser remunerados los RRHH.

RRHH:

(3 integrantes 10 hs. semanales cada uno), son 52 semanas:

1560 horas de recursos humanos, estimadas a US$ 10 la hora resulta en US$ 15.600. Sería razonable agregar un 30% de cobertura para imprevistos dada la vaguedad de la estimación.

INSUMOS:

Se gastaron en el entorno de US$ 300 en materiales y repuestos.

TOTAL: (15.600 + 300)*1.3 = US$ 20.670

14. Análisis de Riesgos

A continuación se detallan los riesgos detectados para la ejecución de este proyecto, así como un plan de contingencia para aquellos que se consideran de mayor incidencia.

1 - Que los algoritmos (que implementan la RN) dados por los actores externos no se comporten correctamente. **Probabilidad baja, alto impacto**.

- **Plan de contingencia:**

  Si esta situación se da y quien provee los algoritmos no puede corregir la situación a tiempo debe redefinirse el alcance del proyecto o utilizar otros algoritmos cuya funcionalidad ya haya sido verificada. El plan de contingencia lleva el riesgo a una **probabilidad baja** con **impacto moderado.**

2 - Que los algoritmos no se puedan implementar en la plataforma elegida (capacidad de cálculo). **Probabilidad baja, impacto moderado.**

- **Plan de contingencia:**

  Se desarrollará el código teniendo especial cuidado en que el sistema implementado sea fácilmente adaptable a otras plataformas y en caso que la situación se de, adaptar el sistema significará una sobre dedicación en el plan de proyecto. El plan de contingencia lleva el riesgo a una **probabilidad baja** con **impacto bajo.**

3 - Que alguno de los elementos de hardware se dañe. **Probabilidad alta, Impacto moderad** .

a) Un chasis de robot como el indicado en [4] (motores)
b) ArduinosMega (2 placas) y Arduino Uno (1 placa)
c) Arduinos Due (Atmel SAM3X8E ARM Cortex-M3 CPU)
d) Una placa Arducam-LF Camera module shield , 3.2 inch LCD for arduino UNO MEGA2560/ DUE
e) Una placa Arducam-F Camera module shield for arduino UNO MEGA2560 / DUE
f) Dos 5 Mega pixel Camera Module OV5642 1080P JPEG Output (compatibles con Arducam)

- **Plan de contingencia:**

  Se adquirieron previo al proyecto repuestos para todos los componentes clave utilizados en el proyecto. Si se dañan todos los repuestos se traspasará el sistema a una plataforma que se pueda adquirir con razonable rapidez (previamente se aclara que el código se implementará para ser fácilmente trasladable). En caso que la situación se de, adaptar el sistema significará una sobre dedicación en el plan de proyecto. El

plan de contingencia lleva el riesgo a una **probabilidad baja** con **impacto bajo.**

4 - Falta de disponibilidad de alguno de los integrantes del equipo:

a) Prolongada, **probabilidad baja, impacto alto.**

b) Breve, **probabilidad moderada, impacto moderado.**

- **Plan de contingencia:**

Si la indisponibilidad es temporal, se ajusta el cronograma distribuyendo las tareas de modo que los integrantes restantes del equipo puedan cubrir la ausencia del tercero sin sobre dedicación. De no ser posible se debe solicitar una prórroga o aumentar las horas dedicadas.

Si la indisponibilidad es permanente el proyecto deja de ser realizable y debe redefinirse el alcance del mismo.El plan de contingencia lleva el riesgo del caso **a)** a una **probabilidad baja** con **impacto moderado** y el caso **b)** a una **probabilidad baja** con **impacto bajo**.

15. Referencias

[1] Tichael J. Milford, Janet Wiles, Gordon F. Wyeth; "Solving Navigational Uncertainty Using Grid Cells on Robots" In Plos Computantional Biology, November 2010, Vol. 6 , Issue 11 , e1000995 www.ploscompbiol.org

[2] Trygve Solstad,. et.al. ;"From Grid Cells to Place Cells: A Mathemcatical Model". In Hippocampus 16:1026-1031 (2006), Wiley InterScience, DOI 10.10002/hipo

[3] Caswell Barry, Daniel Bush1 "From A to Z: a potential role for grid cells in spatial navigation", Neural Systems & Circuits 2012, 2:6 http://www.neuralsystemsandcircuits.com/content/2/1/6

[4] http://www.ebay.com/itm/All-Ready-Arduino-Robot-kit-diy-4-wheel-chassis-sensor-/281082907205?ssPageName=ADME:L:OC:US:3160

[5] http://opencv.org/

Apéndice C

Hito 1

Proyecto de fin de carrera
Ingeniería Eléctrica
Documentación
Hito 1

Autores:

Nicolás Blanco
Mariana del Castillo
Joaquín Quagliotti

Tutor:
Leonardo Barboni
Co-tutor:
Pablo Monzón

15/09/2014

Resumen

• Estudiantes

Nicolás Javier Blanco Antonini
C.I. 4.221.451-8 , Cel. 099032734, blanco.nicolas.c@gmail.com

Mariana del Castillo Larumbe
C.I. 4.192.483-7 , Cel. 099607323, mdelcalaru@gmail.com

Joaquín Quagliotti Chanes
C.I. 4.408.029-8 , Cel. 098383019, joaquin.quagliotti@gmail.com

• Clientes

- Ing. Biomédico Javier Cuneo (Argentino, IUEMHI , estudiante de Doctorado del Grupo de Microelectrónica , DA, Co-DT Fernando Silveira).
- Dr. Pablo Francisco Argibay (Argentino, IUEMHI , co-tutor de tesis de Doctorado de Javier Cuneo).
- Grupo de Microelectrónica del IIE (FING-UdelaR)

• Tutor

Dr. Ing. Leonardo Barboni

• Fecha prevista de finalización

1 de Abril del 2015

• Total de horas a realizar previstas por el grupo de proyecto

Se estima 1600 horas hombre a lo largo del proyecto

1. Introducción

Durante la planificación se propuso para el primer hito, presentar el robot con funcionalidades básicas de movimiento implementadas pero sin interactuar aún con la red neuronal. Demostrar el funcionamiento de los sensores seleccionados y la adquisición de datos de los mismos desplegando la información obtenida de una manera verificable y presentar documentación del código implementado hasta el momento.

Este documento incluye todo lo realizado hasta el momento por el equipo de proyecto en materia de hardware, software e investigación de los temas involucrados. Así como también una revisión de las tareas realizadas hasta el momento.

2. Descripción del Proyecto

Se identifica la necesidad de contar con una plataforma hardware móvil compacta y de fácil configuración (robot), que contenga modelos de sistemas neuronales biológicos programados, así como también recursos para procesamiento de información sensorial. Este robot podrá ser usado por neurocientíficos para estudiar y entender el comportamiento de estas redes como sistemas dinámicos y cual es el nivel de autonomía que se puede alcanzar (por autonomía se entiende habilidad de construir sus propias reglas para solucionar problemas y realizar tareas).

3. Objetivo General

Desarrollar un robot demostrador que incorpore uno de los modelos de sistemas neuronales propuestos por IUEMHI para que pueda someterse a operación en un ambiente real.

4. Software

El RattusBot deberá ser capaz de navegar en una arena de 2x2 metros de manera autónoma, usando como mecanismo de navegación los algoritmos de redes neuronales.

La Figura 1 muestra un diagrama de bloques del sistema. Se indican los sensores, microcontroladores y demás periféricos, mostrando cómo interactúan entre sí.

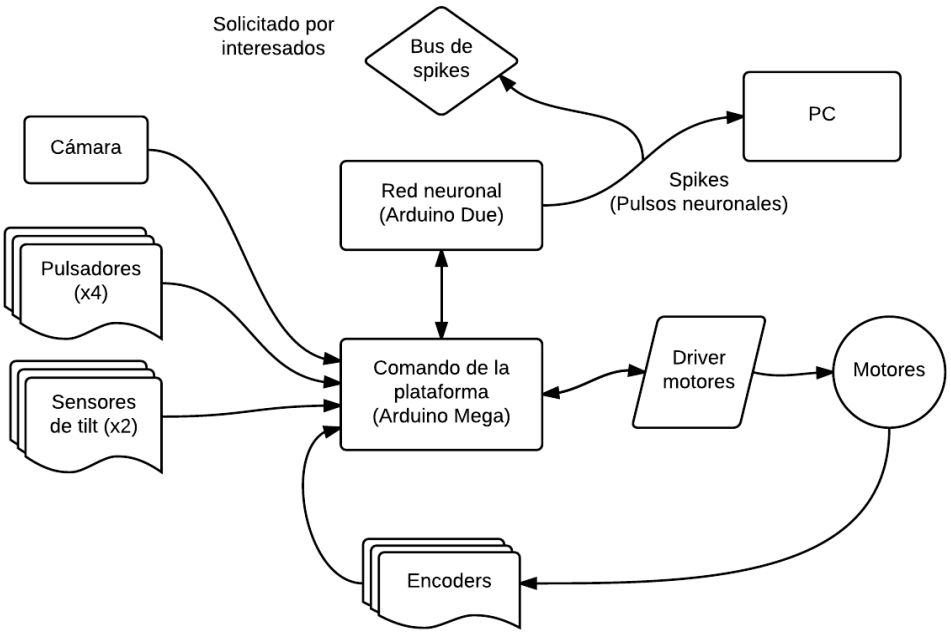


Figura 1 - Nuevo diagrama de bloques.

4.1 Implementación

Para la implementación final se eligió utilizar el IDE de Arduino, por su simpleza, y por la disponibilidad de librerías.

El lenguaje de Arduino está basado en C y C++. A su vez Arduino provee una librería básica que facilita la interacción con hardware por medio de Macros.

Entre las tareas que comprenden la realización de este proyecto estuvo el estudio y entendimiento de las herramientas utilizadas. Se estudió la implementación de las librerías,

para comprender su funcionamiento. Así como las arquitecturas de los microcontroladores utilizados, para comprender sus posibilidades.

Control de motores:

El driver de motores utilizado, esta basado en un “Dual Full-Bridge Driver” (L298), el mismo consiste de dos puentes H de comando independiente, y con una señal de enable para cada puente H. La placa utilizada, incorpora este driver, junto con una fuente regulada, puesto que el L298 permite separar la alimentación de los motores, de la lógica, y así trabajar con alimentaciones del orden de los 12 V para la placa, y aprovechar la regulación interna de la misma para subir el nivel lógico de 5 V y alimentar la placa Arduino.

La placa recibe como entradas 6 señales de control (3 por cada salida de motor), ademas de la señal de 5 V y Gnd. Se estudió la lógica de control para cada salida de motor. La placa permite el control independiente de dos motores. Para cada salida de motor se tiene dos señales digitales que controlan el sentido de giro de las ruedas, y una señal de enable que ajusta el ciclo de trabajo de la onda PWM que controla el motor, regulando así la corriente que alimenta el motor (y por lo tanto su velocidad), y su direccion de giro.

Para la implementación del movimiento del robot, se empezó por definir primero un grupo de funciones globales (avanzar, retroceder, girar en el lugar hacia la derecha y hacia la izquierda) que contienen funciones que controlan a los motores, realizando este control de a pares. Las funciones que controlan a los motores realizan el seteo de los pines usados y se les pasa un parámetro asociado a la potencia que se le entrega a los motores (la velocidad del robot depende de este parámetro, pero influye también la fricción que presente la superficie de contacto de las ruedas).

Sensores:

Se estudiaron los sensores a utilizar, para los mismos se cuenta con el software básico para implementar su funcionalidad. Se estudió el funcionamiento de Arducam Shield y la cámara, y se logró almacenar una imagen de manera matricial para su posterior procesamiento. Se implementaron rutinas de interrupción asociadas a los pulsadores para evitar choques.

Implementación de algoritmos de Redes Neuronales:

Actualmente se está trabajando en ésta parte del código. Se realizó parte de la implementación de los modelos de neurona propuestos y se estudiaron en Matlab las interacciones entre neuronas. Se implementó también la función que define el estímulo a la capa de entrada de la red. Queda pendiente consolidar algunos conceptos para comenzar la implementación de las interacciones entre las neuronas que conforman la red.

5. Hardware

Nueva especificación de Hardware

- Plataforma hardware:
    1. Un chasis
    2. Dos motores DC (Vdc Gear motor 3V-6V)
    3. Un driver motores (L298N Motor shield)
    4. Dos ruedas motorizadas
    5. **Una rueda libre**
    6. Arduino Mega (1 placa)
    7. Arduino Due (1 placa)
    8. Conversores de nivel (de 3.3 V - 5 V)
    9. Una placa Arducam shield Rev C con LCD 3.2 pulgadas
    10. Sensor shield compatible con Arduino Mega
    11. Una 5 Mega pixel Camera Module OV5642
    12. Cuatro pulsadores
    13. Sensores de tilt (bigotes)
    14. **Encoders (rueda ranurada + optoacoplador)**
    15. Fuente de alimentación
    16. Baterías

6. Planificación

6.1. Tareas realizadas y Re-planificación

A continuación se presenta el detalle de las tareas realizadas hasta el momento, con las modificaciones dadas por el cambio en el proyecto y la re-planificación del mismo.

- Armado mecánico del robot y funcionalidades básicas:
    - **Estudiar los manuales del chasis.**
    - ~~**Montar mecánicamente sus piezas.**~~
        - Completada, sin embargo se decidió migrar a otra plataforma.

    - **Implementar el ajuste mecánico de los sensores (por ejemplo de la cámara) y de la plataforma hardware seleccionada al robot.**

- **Implementar rutinas básicas de control de movimiento. Lograr avanzar, retroceder y girar con algún control cinemático básico sin uso de la red neuronal o información del ambiente.**
- Armado de notas para la documentación final.

- Selección de hardware:
  - **Seleccionar la plataforma hardware de entre las listadas en la sección de supuestos.**
  - **Estudiar hojas de datos y manuales de todo el hardware seleccionado (incluyendo sensores, motores y su placa controladora y placa hardware con microprocesador).**
  - **Estudiar tanto el lenguaje como el ambiente de programación de dicha plataforma hardware.**
  - **Armado de notas para la documentación final.**

- Implementar la red neuronal en la plataforma hardware-software elegida
  - **Estudiar elementos de dinámica de sistemas, particularmente de redes neuronales.**
  - **Estudiar el algoritmo que será suministrado y el cual implementa un modelo de red neuronal. Una implementación ya esta en Matlab para estudiar y adaptar.**
  - **Implementar el código en el lenguaje elegido. Realizar las adaptaciones mínimas para lograr la implementación.**
    - En proceso

  - **Probar y depurar**
    - En proceso

  - Implementar la arena de 2m x 2m:
    - Construir el piso y los bordes
    - Pintar patrones de imágenes en el piso y paredes si se usa una cámara como sensor de visión.

    Si se usan sensores de intensidad de luz, se colocarán leds para codificar el ambiente, a los cuales se construirá el driver electrónico para modular la intensidad de luz (el tipo de patrones a colocar debe ser estudiado).
  - Armado de notas para la documentación final.

- Implementar bloques de control y el flujo de información requerido.
  - **Estudiar señales de salida de los sensores y como adaptarlos para que sea el estímulo de la red neuronal.**
    - En proceso e implica ahora obtener la posición del robot en la arena (utilizando encoders y algoritmos de odometría)

- ~~Implementar los algoritmos que transforman las salidas de la red neuronal a señales aptas para el control de los motores del robot.~~
    - Desaparece a causa de ajustes en la planificación del proyecto.

- Construir el sistema de navegación que usa la información del ambiente codificada en las redes neuronales.
- Probar y depurar.
- Armado de notas para la documentación final.

- Integración, prueba y depuración del sistema completo
    - Implementar algoritmos de integración y control de las funciones del robot.
    - Ensamble final de robot con red neuronal implementada y asociada a sensores de estímulo y control de motores.
    - Probar y depurar.
    - Armado de notas para la documentación final.

- Documentación
    - Recopilación de las notas realizadas durante el transcurso del proyecto.
    - **Preparación de las dos presentaciones de hitos del curso de gestión**
        - Parcialmente completada
    - Redacción y edición.
    - Revisión y corrección.

6.2 Resumen de Tareas pendientes y Asignación de Recursos

Taréa	**Duración (Dias)**	**Recursos**
Migrar plataforma	3	Mariana
Implementación de código en el lenguaje elegido	28	Equipo completo
Probar y depurar	10	Equipo completo
Armado de la arena	4	Nicolás
Estudio de señales de salida de los sensores	5	Equipo completo
Construcción del sistema de navegación	9	Equipo completo
Prueba y depuración	15	Equipo completo
Implementación de algoritmos de integración	10	Equipo completo
Ensamble final	10	Equipo completo
Prueba y depuración	42	Equipo completo

Recopilación de notas	5	Equipo completo
Presentación final	12	Equipo completo
Hito 2	1	Joaquín
Hito 1	1	Joaquín
Redacción y edición	20	Equipo completo
Revisión y corrección	14	Equipo completo

7. Cronograma detallado del Proyecto

Se adjunta el nuevo cronograma detallado del proyecto en el Apéndice A.

Apéndice D

Hito 2

Proyecto de fin de carrera
Ingeniería Eléctrica
Documentación
Hito 2

Autores:

Nicolás Blanco
Mariana del Castillo
Joaquín Quagliotti

Tutor:
Leonardo Barboni

15/02/2015

Resumen

• Estudiantes

Nicolás Javier Blanco Antonini
C.I. 4.221.451-8 , Cel. 099032734, blanco.nicolas.c@gmail.com

Mariana del Castillo Larumbe
C.I. 4.192.483-7 , Cel. 099607323, mdelcalaru@gmail.com

Joaquín Quagliotti Chanes
C.I. 4.408.029-8 , Cel. 098383019, joaquin.quagliotti@gmail.com

• Clientes

- Ing. Biomédico Javier Cuneo (Argentino, IUEMHI , estudiante de Doctorado del Grupo de Microelectrónica , DA, Co-DT Fernando Silveira).
- Dr. Pablo Francisco Argibay (Argentino, IUEMHI , co-tutor de tesis de Doctorado de Javier Cuneo).
- Grupo de Microelectrónica del IIE (FING-UdelaR)

• Tutor

Dr. Ing. Leonardo Barboni

• Fecha prevista de finalización

1 de Abril del 2015

• Total de horas a realizar previstas por el grupo de proyecto

Se estima 1600 horas hombre a lo largo del proyecto

1. Introducción

Durante la planificación se propuso para el primer hito, presentar el robot con funcionalidades básicas de movimiento implementadas pero sin interactuar aún con la red neuronal. Demostrar el funcionamiento de los sensores seleccionados y la adquisición de datos de los mismos desplegando la información obtenida de una manera verificable y presentar documentación del código implementado hasta el momento.

Este documento incluye todo lo realizado hasta el momento por el equipo de proyecto en materia de hardware, software e investigación de los temas involucrados. Así como también una revisión de las tareas realizadas hasta el momento.

2. Descripción del Proyecto

Se identifica la necesidad de contar con una plataforma hardware móvil compacta y de fácil configuración (robot), que contenga modelos de sistemas neuronales biológicos programados, así como también recursos para procesamiento de información sensorial. Este robot podrá ser usado por neurocientíficos para estudiar y entender el comportamiento de estas redes como sistemas dinámicos y cual es el nivel de autonomía que se puede alcanzar (por autonomía se entiende habilidad de construir sus propias reglas para solucionar problemas y realizar tareas).

3. Objetivo General

Desarrollar un robot demostrador que incorpore uno de los modelos de sistemas neuronales propuestos por IUEMHI para que pueda someterse a operación en un ambiente real.

4. Software

El RattusBot deberá ser capaz de navegar en una arena de 2x2 metros de manera autónoma, determinando su ubicación a partir de sus sensores y modelando el comportamiento eléctrico de la mayor cantidad implementable de neuronas tipo “grid-cell”..

La Figura 1 muestra un diagrama de bloques del sistema. Se indican los sensores, microcontroladores y demás periféricos, mostrando cómo interactúan entre sí.

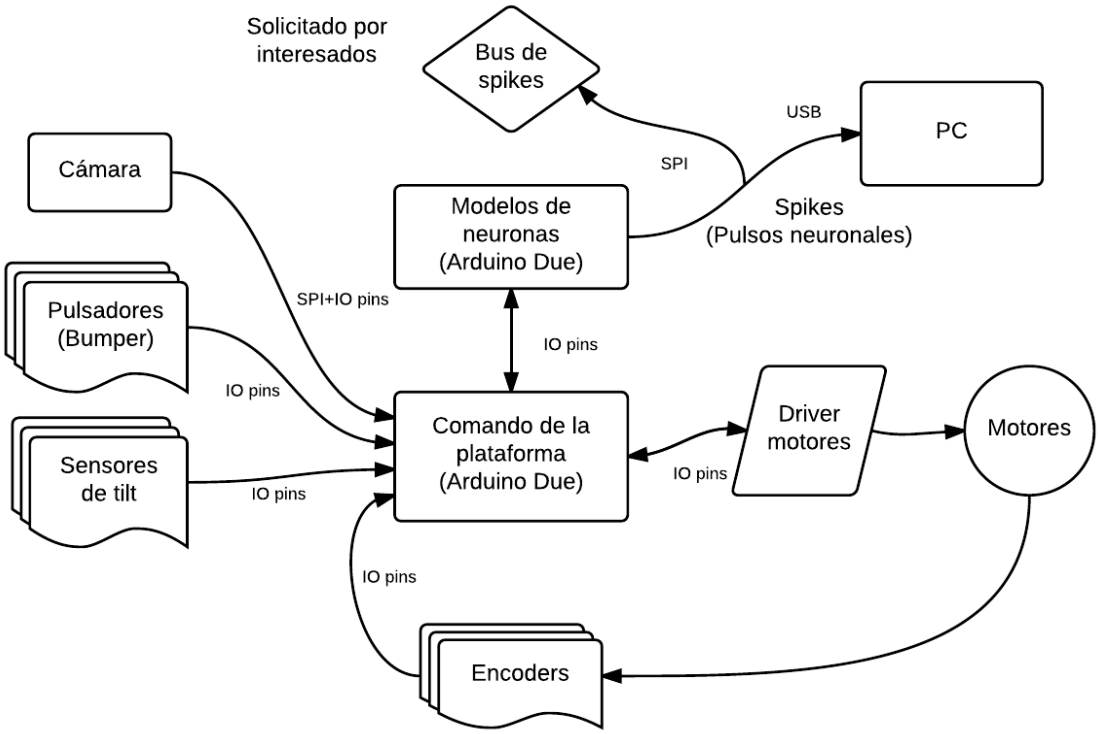


Figura 1 - Diagrama de bloques.

4.1 Implementación

Para la implementación final se eligió utilizar el IDE de Arduino, por su simpleza, y por la disponibilidad de librerías.

El lenguaje de Arduino está basado en C y C++. A su vez Arduino provee una librería básica que facilita la interacción con hardware por medio de Macros.

Entre las tareas que comprenden la realización de este proyecto estuvo el estudio y entendimiento de las herramientas utilizadas. Se estudió la implementación de las librerías,

para comprender su funcionamiento. Así como las arquitecturas de los microcontroladores utilizados, para comprender sus posibilidades.

Para el microcontrolador asociado al comando de la plataforma se implementó una arquitectura de round robin con interrupciones de modo de poder manejar cómodamente la multiplicidad de sensores que generan interrupciones. A continuación se describen los componentes asociados a la plataforma y las soluciones de software implementadas para cada uno de ellos.

4.1.1 Control de motores

El driver de motores utilizado, esta basado en un “Dual Full-Bridge Driver” (L298), el mismo consiste de dos puentes H de comando independiente, y con una señal de enable para cada puente H. La placa utilizada, incorpora este driver, junto con una fuente regulada, puesto que el L298 permite separar la alimentación de los motores, de la lógica, y así trabajar con alimentaciones del orden de los 12 V para la placa, y aprovechar la regulación interna de la misma para subir el nivel lógico de 5 V y alimentar la placa Arduino.

La placa recibe como entradas 6 señales de control (3 por cada salida de motor), ademas de la señal de 5 V y Gnd. Se estudió la lógica de control para cada salida de motor. La placa permite el control independiente de dos motores. Para cada salida de motor se tiene dos señales digitales que controlan el sentido de giro de las ruedas, y una señal de enable que ajusta el ciclo de trabajo de la onda PWM que controla el motor, regulando así la corriente que alimenta el motor (y por lo tanto su velocidad), y su direccion de giro. La lógica de operación del driver es de acuerdo a la Tabla 1.

Pin\Comportamiento	Detenido	Giro horario	Giro antihorario
IN1	0/1	0	1
IN2	0/1	1	0

Tabla 1 - Comportamiento del driver

Para cada motor el driver tiene 3 terminales de control, IN1 e IN2 definen el sentido de la rotación y el pin EN condiciona la potencia que se suministra al motor (mediante el uso de PWM). La posición detener tiene dos comportamientos diferentes según si se fija 0 en el pin EN (frenado suave) o 255 (frenado forzado).

Para la implementación del movimiento del robot, se empezó por definir primero un grupo de funciones globales (avanzar, retroceder, girar en el lugar hacia la derecha y hacia la izquierda) que contienen funciones que comandan a los motores, realizando este control de a pares. Las funciones realizan el seteo de los pines usados y se les pasa un parámetro asociado al ciclo de la PWM de salida en la Arduino. A su vez esta salida está asociada a la

potencia que se le entrega a los motores por medio del driver (la velocidad del robot depende de este parámetro, pero influye también la situación y condiciones en las que se encuentre trabajando el motor).

Se definió que el comportamiento del robot en la arena será aleatorio, simulando el de una rata que explora un ambiente similar. Durante el recorrido se estima la posición del robot relativa al punto de partida utilizando la información provista por los sensores. Se profundizará sobre este punto en las secciones siguientes.

El control del comportamiento de la plataforma se realiza en una de las funciones del round robin cada una cierta cantidad de tiempo.Se continúa trabajando en afinar el control sobre el movimiento de la plataforma.

4.1.2 Sensores

Se estudiaron los sensores a utilizar, para los mismos se cuenta con el software básico para implementar su funcionalidad.

4.1.2.1 Cámara.

Se estudió el funcionamiento de Arducam Shield y la cámara, y se logró almacenar una imagen de manera matricial para su posterior procesamiento. Se estudiaron los ejemplos de la librería Arducam y se implementó una rutina que saca una foto en RGB 565 (5 bits para codificar el Rojo, 6 bits para el verde y 5 para el Azul). Se investigaron e implementaron distintas ecuaciones de conversión para obtener una imagen en escala de grises. Mediante la observación de estas imágenes obtenidas, y la comparación de estas con una imagen en blanco y negro puros, obtenida de la anterior utilizando un umbral de conversión. Se eligió la siguiente ecuación de conversión.

NivelGris = (rgbComp.R)*3 + (rgbComp.G)*3 + (rgbComp.B)) / 10;

Luego si NivelGris > 2, Se cuenta como bit blanco, sino como bit negro. De esta manera, Sacando fotos a patrones prefijados y contando la cantidad de pixeles negros se puede obtener una idea de que patrón se esta observando. Seteando patrones suficientemente diferentes se puede saber que patrón se esta observando, aunque la cuenta de pixeles entre imagenes consecutivas varía considerablemente. Esto funcionará correctamente siempre que la diferencia entre numero de pixeles entre patrones sea mayor que la diferencia más significativa entre cuentas del mismo patrón.

4.1.2.2 Pulsadores y sensores de "tilt":

Se implementaron rutinas de interrupción asociadas a los pulsadores para evitar choques. Las mismas aprovechan las ventajas de Arduino Due en cuanto a la utilización de interrupciones y la posibilidad de configurar cualquiera de sus pines como señal de interrupción.

Las rutinas de atención a las interrupciones asociadas a los microswitches, simplemente levantan una bandera, la cual habilita en el Round-Robin, una rutina de detención y desvío.

Para los sensores de tilt, se consulta la entrada analogica a la que estan conectados y se compara con un umbral. En función del resultado se decide girar el robot hacia el otro lado o continuar con el movimiento.

4.1.2.3 Trackers y encoders.

Para determinar la ubicación espacial del robot relativa a su punto de partida en la arena se optó por implementar un algoritmo que determine esta posición a partir del ángulo de rotación de las ruedas. Para ésto, se colocaron encoders en las ruedas que permiten rastrear este ángulo de giro. Los encoders son discos ranurados que interrumpen al emisor de un optoacoplador. Se puede entonces obtener una señal pulsada que está en VCC cuando el disco está interrumpiendo al optoacoplador y GND cuando está en una ranura. La figura 2 muestra un esquema de un encoder con su tracker respectivo.

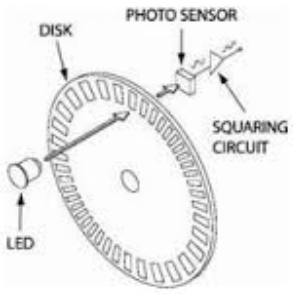


Figura 2 - Encoder.

La señal pulsada obtenida permite observar el ángulo de rotación del eje del motor y por lo tanto de cada rueda, sin embargo la implementación de este sistema presenta ciertos problemas. Principalmente se observó que si la rueda se detiene de modo que el borde de una ranura quede interrumpiendo parcialmente al optoacoplador se generan pulsos ante la más mínima variación en la vibración de la rueda, que no se corresponden a desplazamientos de la misma. Esto impide tener un seguimiento certero del ángulo de rotación y por lo tanto de la posicion del robot. Para solucionar este problema, en lugar de utilizar un solo tracker por rueda se colocaron dos, de modo que las ondas pulsadas queden en cuadratura. Esto permite tener una salida que queda naturalmente representada en un código Grey. La figura 3 muestra las ondas pulsadas en cuadratura y el valor resultante.

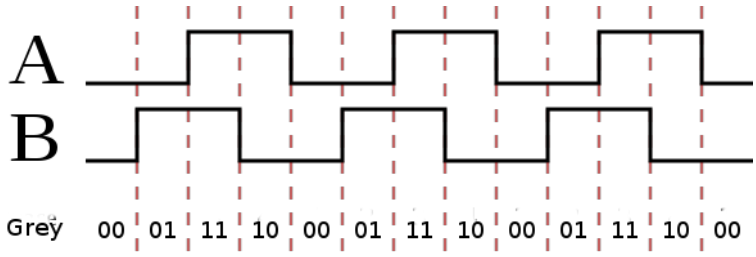


Figura 3 - Ondas en cuadratura.

Este sistema permite obtener la variación en el ángulo de rotación del eje así como el sentido de giro, de modo que los rebotes que se generaban anteriormente no sumen al ángulo medido. Como se puede observar en la Figura 3 este sistema además cuadruplica la resolución con la que se mide el ángulo. Sin embargo ajustar la posición del segundo optoacoplador de modo que las ondas queden según se desea resultó muy engorroso y generalmente el sistema se desajustaba ante el menor movimiento. Ante este nuevo problema se optó por obstruir la mitad de las hendiduras del disco de modo que el ajuste resulte más facil y robusto, manteniendo aún el doble de la resolución que en el planteo inicial.

4.1.2.4 Odometría.

“La odometría es el estudio de la estimación de la posición de vehículos con ruedas durante la navegación. Para realizar esta estimación se usa información sobre la rotación de las ruedas para estimar cambios en la posición a lo largo del tiempo.” [Wikipedia]

Como se comentó anteriormente, para rastrear la posición del robot en la arena se optó por utilizar la odometría. Para esto se utilizan encoders rotativos con trackers en cuadratura que permiten rastrear el ángulo de

rotación de las ruedas y así determinar el desplazamiento de la plataforma entre dos puntos. Este sistema presenta múltiples inconvenientes ya que es válido solo para pequeños arcos de desplazamiento, y no es válido ante deslizamientos de las ruedas ni sobre superficies irregulares. Esto se debe a que dichas condiciones invalidan las hipótesis sobre las que se sienta el modelo realizado.

El algoritmo utilizado modela sencillamente al sistema. Se define

$$C_m = \frac{2\pi R}{C_e}$$

Donde R es el radio de las ruedas y C_e la resolución del encoder (en nuestro caso es el doble de la cantidad de hendiduras en cada disco). Donde C_m permite conocer el desplazamiento lineal de cada rueda utilizando una relación de la forma:

$$\Delta_{sL} = C_m N_L$$

$$\Delta_{sR} = C_m N_R$$

Donde N_L y N_R representan la cantidad de pulsos medidos para cada rueda durante la trayectoria que se está evaluando. A partir de los desplazamientos lineales y suponiendo que el recorrido del robot es un arco de circunferencia diferencial como se puede observar en la figura 4, el desplazamiento de cada rueda es:

$$\Delta_{sL} = r\Delta\theta$$

$$\Delta_{sR} = (r + L)\Delta\theta$$

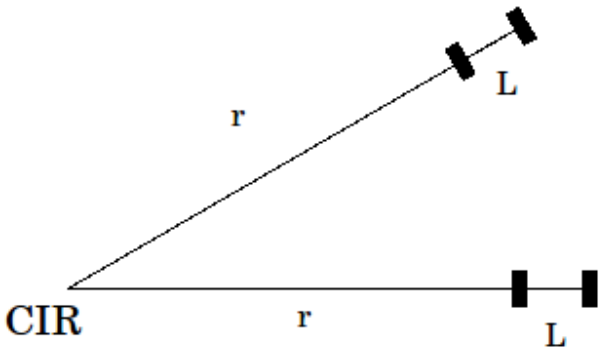


Figura 4 - Modelado del sistema

Se pueden deducir entonces las siguientes ecuaciones:

$$\Delta\theta = \frac{\Delta_{sR} - \Delta_{sL}}{L} \qquad (1)$$

$$r = \frac{L\Delta_{sL}}{\Delta_{sR} - \Delta_{sL}} \quad (2)$$

El desplazamiento del centro del eje se calcula a partir de (1) y (2) como:

$$\Delta_s = \Delta\theta(r + \frac{L}{2}) = (\frac{\Delta_{sR} - \Delta_{sL}}{L})(\frac{L\Delta_{sL}}{\Delta_{sR} - \Delta_{sL}} + \frac{L}{2}) = (\Delta_{sL} + \frac{\Delta_{sR}}{2} - \frac{\Delta_{sL}}{2})$$

$$\Delta_s = \frac{\Delta_{sL} + \Delta_{sR}}{2} \quad (3)$$

Las ecuaciones (1) y (3) permiten calcular entonces las variaciones en el ángulo theta correspondiente al que forma el eje de las ruedas con la horizontal y el desplazamiento del centro del eje a partir del desplazamiento lineal de cada rueda. Se tiene entonces a partir de las ecuaciones anteriores el algoritmo utilizado.

$$R = radio\ de\ las\ ruedas$$

$$L = distancia\ entre\ centros\ de\ las\ ruedas$$

$$C_e = resolución\ del\ encoder$$

$$N_{Ri} = cantidad\ de\ pulsos\ contados\ en\ la\ rueda\ derecha$$

$$N_{Li} = cantidad\ de\ pulsos\ contados\ en\ la\ rueda\ izquierda$$

$$C_m = \frac{2\pi R}{C_e}$$

$$\Delta\theta_i = \frac{C_m N_{Ri} - C_m N_{Li}}{L}$$

$$\theta_i = \theta_{i-1} + \Delta\theta_i$$

$$x_i = x_{i-1} + \Delta_s cos(\theta_i)$$

$$y_i = y_{i-1} + \Delta_s sen(\theta_i)$$

4.1.3 Implementación de algoritmos de Redes Neuronales:

4.1.3.1 Grid-cells y función G

Una Grid-cell es un tipo de neurona que se encuentra en la corteza entorrinal de muchas especies y a la cual se atribuye la capacidad de un

individuo de comprender su posición en el espacio. Si se pone a una rata dentro de un espacio contenido, censando el comportamiento de una de estas neuronas se puede observar que los puntos en los que se generan disparos forman una grilla de nodos equidistantes como se muestra en la Figura 5

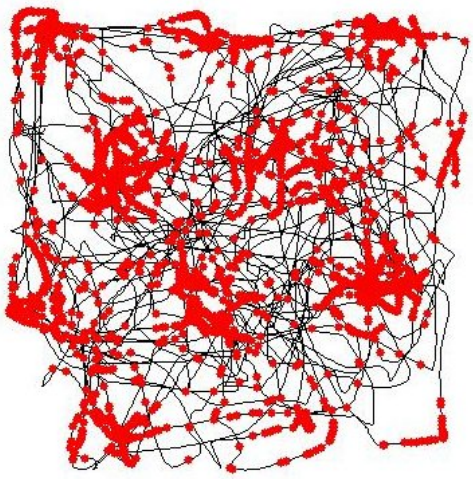


Figura 5 - Actividad de una grid-cell en la corteza entorrinal de una rata desplazándose en un espacio contenido.

Es el comportamiento de este tipo de neurona el que se implementa en la plataforma, para esto se utilizó el siguiente modelo, provisto por el Ing. Biomédico Javier Cuneo

$$G(x,y) = \frac{2}{3}\left(\frac{1}{3}\sum_{i=1}^{3} cos(k_i(x,y)) + \frac{1}{2}\right) \quad (4)$$

G es una suma de cosenos desfasados que representa la tasa de actividad de este tipo de neuronas en función de la posición espacial y de características propias de cada célula.

En la ecuación (4) k_i es calculado según la relación mostrada en las ecuaciones (5), (6) y (7).

$$k_1 = \frac{4\pi\lambda_i}{\sqrt{6}}\left(\left(cos(\theta_i + \tfrac{\pi}{12}) + sin(\theta_i + \tfrac{\pi}{12})\right)(x - \varphi_{i,x}) + \left(cos(\theta_i + \tfrac{\pi}{12}) - sin(\theta_i + \tfrac{\pi}{12})\right)(y - \varphi_{i,y})\right) \quad (5)$$

$$k_2 = \frac{4\pi\lambda_i}{\sqrt{6}}\left(\left(cos(\theta_i + \tfrac{5\pi}{12}) + sin(\theta_i + \tfrac{5\pi}{12})\right)(x - \varphi_{i,x}) + \left(cos(\theta_i + \tfrac{5\pi}{12}) - sin(\theta_i + \tfrac{5\pi}{12})\right)(y - \varphi_{i,y})\right) \quad (6)$$

$$k_3 = \frac{4\pi\lambda_i}{\sqrt{6}}\left(\left(cos(\theta_i + \tfrac{3\pi}{12}) + sin(\theta_i + \tfrac{3\pi}{12})\right)(x - \varphi_{i,x}) + \left(cos(\theta_i + \tfrac{3\pi}{12}) - sin(\theta_i + \tfrac{3\pi}{12})\right)(y - \varphi_{i,y})\right) \quad (7)$$

Donde θ_i, $\varphi_{i,x}$, $\varphi_{i,y}$ y λ_i son parámetros asociados a la neurona i que determinan la forma de la grilla. Estos parámetros se generan aleatoriamente utilizando las siguientes funciones:

$$\lambda_i = 3 - 1.5\chi_i$$
$$\theta_{x_i} = 2\pi\eta(0,1)$$
$$\theta_{env} = 2\pi\eta(0,1)$$
$$\theta_i = \theta_{env} + \theta_{\chi_i}$$
$$\varphi_{x_i} = \eta(0,1)$$
$$\varphi_{y_i} = \eta(0,1)$$
$$\varphi_{env} = \eta(0,1)$$
$$\varphi_{i,x} = \varphi_{env} + \varphi_{x_i}$$
$$\varphi_{i,y} = \varphi_{env} + \varphi_{y_i}$$

Donde χ_i es un parámetro que depende de la posición de la célula que a los efectos del modelo toma valores entre 0 y 1, de modo que para la célula i el valor de χ_i es $\frac{i}{N}$ donde N es la cantidad de neuronas modeladas; λ_i es el tamaño de la grilla entre 1.5 y 3, $\eta(0,1)$ es un número aleatorio de distribución normal con media 0 y desviación estándar 1; θ_{env} es la orientación en la grilla que el ambiente produce para todas las células; θ_{χ_i} es la orientación de la célula sin influencia del ambiente; θ_i es la orientación de la grilla; φ_{x_i} es el offset horizontal de la grilla para la célula sin influencia del ambiente; φ_{y_i} es el offset vertical de la grilla para la célula sin influencia del ambiente; φ_{env} es el offset que el ambiente produce en la grilla; $\varphi_{i,x}$ es el offset horizontal total de la célula; $\varphi_{i,y}$ es el offset vertical total de la célula.

Los parámetros intrínsecos de las células fueron generados fuera del sistema de modo que no varíen de ejecución en ejecución del programa.

4.1.3.2 Algoritmo de Izhikevich

Para el comportamiento eléctrico de las neuronas se utilizó el modelo propuesto por Eugene M. Izhikevich para neuronas pulsantes. Este modelo fue indicado por el Ing. Biomédico Javier Cuneo y considera aspectos biológicos que hacen al comportamiento eléctrico de las neuronas. Se estudió e implementó el algoritmo pero sus fundamentos exceden los conocimientos del equipo y no fueron investigados en profundidad.

El modelo propone las siguientes ecuaciones:

$$v_t = v_{t-1} + t\left[\frac{(k(v_{t-1}-v_{off}-v_r)(v_{t-1}-v_{off}-v_{th}) + I - u_{t-1} + b(v_{off}+v_r))}{C}\right]$$
$$u_t = u_{t-1} + t(a(b(v_{t-1} - v_{off} - v_r) - u_{t-1} + b(v_{off} + v_r)))$$

donde se debe resetear luego de un pico con el siguiente criterio

$$
\begin{aligned}
&if\,(v \geq\ v_{peak} + v_{off})\{\\
&\qquad v_t = c + v_{off};\\
&\qquad u_t = u_t + d;\\
&\}
\end{aligned}
$$

En las ecuaciones, v representa el potencial de membrana; u es una variable para la recuperación de membrana; I representa las corrientes sinápticas que excitan a la neurona; v_{off} es un offset agregado al modelo para que los niveles de tensión sean siempre positivos; v_{peak} es la tensión de membrana máxima alcanzada cuando la neurona dispara; v_r es la tensión de descanso de la neurona; v_{th} la tensión de umbral de disparo; c es la tensión de recuperación a la que baja el potencial de membrana luego de un disparo; d es el equivalente a lo anterior para u; a, b y C son constantes que caracterizan a las curvas.

El algoritmo para el comportamiento de las neuronas se implementó con las ecuaciones anteriores donde I, la corriente sináptica que excita a la neurona, se modeló como $I = I_{max}G(x,y,i)$ donde I_{max} es la corriente máxima, (x,y) la posición del robot e i un índice que indica la neurona cuyo comportamiento se está calculando. La función G es la que se describe en la sección 4.1.3.1, de este modo se simula el estímulo que recibe una grid cell y se obtiene el comportamiento deseado.

4.1.3.3 Optimización

Dado que es necesario modelar la mayor cantidad posible de neuronas, fue necesario un proceso de optimización del código de modo de minimizar el tiempo de cómputo de cada neurona. Para ésto se realizaron modificaciones con los criterios descritos a continuación.

Se realizaron modificaciones de forma de no realizar múltiples veces la misma operación. Se encontró que la ineficiencia en este sentido incrementa en gran cantidad el tiempo de cómputo.

Se escalaron todas las funciones de modo que todos los valores pudieran ser representados con valores enteros los y así aprovechar el multiplicador de enteros por hardware con el que cuenta el procesador. Este punto fue el que más ganancia dio en cuando al tiempo de ejecución.

El escalado se realizó de acuerdo al siguiente razonamiento, si tomamos las funciones que define el comportamiento de las neuronas

$$v_t = v_{t-1} + t\left[\frac{(k(v_{t-1}-v_{off}-v_r)(v_{t-1}-v_{off}-v_{th}) + I - u_{t-1} + b(v_{off}+v_r))}{C}\right]$$
$$u_t = u_{t-1} + t(a(b(v_{t-1} - v_{off} - v_r) - u_{t-1} + b(v_{off} + v_r)))$$

Se desea que la variable v esté multiplicada por un factor de 100 para poder manejarla como entera sin generar errores significativos.

$$10^3 v_t = 10^3 v_{t-1} + 10^3 t\left[\frac{(k(v_{t-1}-v_{off}-v_r)(v_{t-1}-v_{off}-v_{th}) + I - u_{t-1} + b(v_{off}+v_r))}{C}\right]$$
$$10^3 v_t = 10^3 v_{t-1} + t\left[\left(\frac{10k(10^3 v_{t-1}-10^3 v_{off}-10^3 v_r)(10^3 v_{t-1}-10^3 v_{off}-10^3 v_{th})}{10^4} + 10^3 I - 10^3 u_{t-1} + b(10^3 v_{off} + 10^3 v_r)\right)\right]\frac{1}{C}$$

Análogamente para u:

$$10^3 u_t = 10^3 u_{t-1} + \frac{t(100a(b(10^3 v_{t-1}-10^3 v_{off}-10^3 v_r)-10^3 u_{t-1}+b(10^3 v_{off}+10^3 v_r)))}{100}$$

Se definen entonces las nuevas variables y parámetros escalados:

$$v'_t = 10^3 v_t$$
$$v'_{t-1} = 10^3 v_{t-1}$$
$$v'_{off} = 10^3 v_{off}$$
$$v'_{th} = 10^3 v_{th}$$
$$v'_r = 10^3 v_r$$
$$I' = 10^3 I$$
$$u'_t = 10^3 u_t$$
$$u'_{t-1} = 10^3 u_{t-1}$$
$$k' = 10k$$
$$a' = 100a$$

Con estos parámetros y variables escaladas se puede definir las nuevas funciones que rigen el comportamiento del modelo:

$$v'_t = v'_{t-1} + t\left[\left(\frac{k'(v'_{t-1}-v'_{off}-v'_r)(v'_{t-1}-v'_{off}-v'_{th})}{10^4} + I' - u'_{t-1} + b(v'_{off} + v'_r)\right)\right]\frac{1}{C}$$

$$u'_t = u'_{t-1} + \frac{t(a'(b(v'_{t-1}-v'_{off}-v'_r)-u'_{t-1}+b(v'_{off}+v'_r)))}{100}$$

Esta nueva configuración permite manejar todas las variables de tipo entero con lo que el tiempo de cómputo se reduce sustancialmente.

Se escaló también con el mismo criterio la función G(x,y), de la fórmula:

$$G(x,y) = \frac{2}{3}(\frac{1}{3}\sum_{i=1}^{3} cos(k_i(x,y)) + \frac{1}{2})$$

Se deduce:

$$10^3 G(x,y) = \frac{2}{3}(\frac{1}{3}\sum_{i=1}^{3} 10^3 cos(10^3 k_i(x,y)) + \frac{10^3}{2})$$

Esto permite manejar también valores enteros para esta función y por último se utilizó una "lookup table" para determinar el valor de $cos(10^3\theta)$. Esto consiste en una tabla precalculada que en la posición $10^3\theta$ se encuentra el valor de $cos(10^3\theta)$. Con esto se se calcula el coseno sin recurrir a métodos numéricos que consumen más tiempo.

Por último, para optimizar aún más las secciones del código que se ejecutan más frecuentemente se escribieron por extensión los ciclos de *for* y *while.* De este modo se elimina el salto durante la ejecución del código que implica el reseteo de la pipeline. Este reseteo no es deseado ya que implica mayor cantidad de ciclos de reloj para ejecutar cada operación.

Luego de este proceso de optimización se consiguió simular con pasos de $1\,ms$ 199 neuronas. Queda pendiente afinar este número luego de implementar la comunicación hacia afuera de los pulsos de las neuronas.

4.1.4 Comunicación entre Arduinos

Para la comunicación entre Arduinos se necesitan mandar de una Due a la otra la posición (x,y) del robot periódicamente. Como la Due que hace el procesamiento neuronal precisa completar todas sus funciones en menos de un periodo de integración del IK (Izhikevich), es fundamental ajustar los tiempos empleados por las comunicaciones. Se consiguió implementar rutinas de comunicación utilizando manipulación directa de puertos. El procesador prevé esto y agrupa muchos de sus pines en puertos de 32 bits. Para muchos de estos bits Arduino prevé una conexión directa a pines externo de la placa.

Se hicieron pruebas para verificar el funcionamiento de estas instrucciones y se seleccionaron pines dentro de los puertos disponibles para facilitar la escritura y lectura de la posición por ambas Dues.

Dado que la arena es de 2m x 2m, se eligió codificar la posición cada un centímetro, que es aproximadamente la precisión obtenida en la odometría. Por ende

para cada dirección (x,y) se tiene mínimo dos bytes de información para mandar entre las Dues (2^8 = 256).

Los puertos utilizados para escribir y leer las posiciones son, el puerto C para la coordenada x, y el puerto A para la coordenada y. Sin embargo estos puertos son de 32 bits (aunque no todos los bits estan disponibles como pines en la arduino). Para facilitar la lectura y escritura de los bytes, así como las pruebas realizadas se eligieron los pines correspondientes a nibbles de la palabra de 32bits.

De esta manera por medio de máscaras y shifts en las palabras leídas se puede reconstruir rápidamente lo escrito en los puertos.

4.1.5 Puertos Paralelos (A y C)

Puerto A

bits 31..28	*bits 27..24*	*bits 23..20*	*bits 19..16*	*bits 15..12*	*bits 11..8*	*bits 7..4*	*bits 3..0*
				High Nibble			*Low Nibble*

Tabla 2 - Nibbles correspondientes a la palabra de 8 bits que codifica la posición según el eje Y

Puerto C

bits 31..28	*bits 27..24*	*bits 23..20*	*bits 19..16*	*bits 15..12*	*bits 11..8*	*bits 7..4*	*bits 3..0*
			High Nibble	*Low Nibble*			

Tabla 3 - Nibbles correspondientes a la palabra de 8 bits que codifica la posición según el eje X

Cada puerto se puede manipular directamente como una palabra de 32 bits. Dado que para el cortex los enteros son de 32 bits se utiliza este tipo de dato como variable, en las Tablas 2 y 3 se muestran que nibbles de las palabras afectan en cada puerto a los bits utilizados, estos se corresponden a su vez con los pines físicos descritos en la sección 5.2.

Cabe destacar que ninguno de los dos puertos utilizados contaba con la conexión física de los 16 bits necesarios, organizados correlativamente de forma que fuera sencilla su manipulación.

Se buscaron pines que fueran correlativos a nibbles de la palabra de 32 bits para facilitar la depuración y manejo de código.

5. Hardware

5.1 Especificación de Hardware

- Plataforma hardware:
    1. Un chasis
    2. Dos motores DC (Vdc Gear motor 3V-6V)
    3. Un driver motores (L298N Motor shield)
    4. Dos ruedas
    5. **Una rueda libre**
    6. Arduino Due (2 placas)
    7. Conversores de nivel (de 3.3 V - 5 V)
    8. Una placa Arducam shield Rev C con LCD 3.2 pulgadas
    9. Una 5 Mega pixel Camera Module OV5642
    10. Dos pulsadores (bumper)
    11. Dos sensores de tilt (bigotes)
    12. **Encoders (2 ruedas ranuradas + 4 optoacopladores)**
    13. Fuente de alimentación
    14. Baterías
    15. PCBs Adaptadores de señales

Para la parte de Hardware se realizaron tareas manuales de adaptación de los distintos componentes. Se construyó,con la ayuda de José Vila, una nueva plataforma de acrílico, en dos niveles, de manera de aprovechar mejor el espacio para colocar todas las componentes.

Se realizó el montaje y ajuste del bumper frontal y los microswitches correspondientes, así como los soportes para los sensores de tilt. Se montaron 4 leds para mejorar la imagen obtenida a través de la cámara, los mismos son conectados en paralelo a un pin de la Arduino, de manera de encenderlos solo al momento de tomar la foto, para así reducir el consumo de corriente del sistema.

Se realizaron los PCBs correspondientes a las adaptaciones de los trackers, y los sensores de tilt, así como de una barra de alimentación de 5V, 3.3V y GND, para facilitar el conexionado de todos los componentes.

Se reparó un PC vieja, y se la acondicionó para poder utilizar una misma alimentación para todo el sistema , y poder a su vez contar con la comunicación serial, entre las Arduino y la PC, para realizar pruebas en el sistema de odometría.

5.2 Mapas de Pines

Arduino Baja	
Pin	Función
A0	Random seed (Reservado)
A1	Bigote Derecho
A2	Bigote Izquierdo
A6	com PuertoA
A7	com PuertoA
A8	Bumper Izquierdo
A9	Bumper Derecho
CANRX	com PuertoA
CANTX	com PuertoA
D2	IN3
D3	LEDS
D4	IN4
D5	IN1
D6	SDShield
D7	IN2
D8	Tracker derecho
D9	Tracker izquierdo
D10	LCD
D11	ENB
D12	ENA
D16	com PuertoA
D17	com PuertoA
D19	Tracker Derecho 2
D20	Tracker izquierdo 2
D23	com PuertoA
D24	com PuertoA

D44	*com PuertoC*
D45	*com PuertoC*
D46	*com PuertoC*
D47	*com PuertoC*
D48	*com PuertoC*
D49	*com PuertoC*
D50	*com PuertoC*
D51	*com PuertoC*

Tabla 4 - Mapa de Pines Arduino Baja

Arduino Alta	
Pin	*Función*
A6	*com PuertoA*
A7	*com PuertoA*
CANRX	*com PuertoA*
CANTX	*com PuertoA*
D16	*com PuertoA*
D17	*com PuertoA*
D23	*com PuertoA*
D24	*com PuertoA*
D44	*com PuertoC*
D45	*com PuertoC*
D46	*com PuertoC*
D47	*com PuertoC*
D48	*com PuertoC*
D49	*com PuertoC*
D50	*com PuertoC*
D51	*com PuertoC*

Tabla 5 - Mapa de Pines Arduino Alta

6. Planificación

6.1. Situación actual

La Tabla 6 incluye las tareas que fueron planificadas en el hito 1 y su estado actual. La única tarea atrasada es el armado de la arena ya que se optó por posponerla dado que no resultó ser tan crítica a causa de algunos cambios realizados en la especificación del proyecto. Inicialmente estaba planteado de modo que se utilizarán modelos de redes neuronales para la navegación. Sin embargo luego de una entrevista con el interesado se verificó que su necesidad es que el robot se mueva aleatoriamente por la arena, determinando su propia posición y devolviendo periódicamente los estados de las neuronas modeladas. A causa de este inconveniente algunas tareas modificaron su duración y por lo tanto el armado de la arena debió aplazarse.

Tarea	Duración (Dias)	Recursos	Estado
Migrar plataforma	3	Mariana	Completada
Implementación de código en el lenguaje elegido	28	Equipo completo	Completada
Probar y depurar	10	Equipo completo	Completada
Armado de la arena	4	Nicolás	Atrasada
Estudio de señales de salida de los sensores	5	Equipo completo	Completada
Construcción del sistema de navegación	9	Equipo completo	Completada
Prueba y depuración	15	Equipo completo	Completada
Implementación de algoritmos de integración	10	Equipo completo	Completada
Ensamble final	10	Equipo completo	Completada
Prueba y depuración	42	Equipo completo	60% completada
Recopilación de notas	5	Equipo completo	Completada
Presentación final	12	Equipo completo	Pendiente
Presentación Hito 2	1	Joaquín	Completada

Presentación Hito 1	1	Joaquín	Completada
Redacción y edición	20	Equipo completo	Pendiente
Revisión y corrección	14	Equipo completo	Pendiente

Tabla 6 - Resumen de tareas al momento

.

6.2. Tareas pendientes y re-planificación

A continuación se presenta en la Tabla 7 el detalle de las tareas pendientes, con las modificaciones realizadas. Se agregó una nueva tarea (Comunicación de spikes) y se redujo la duración de otras para poder entregar el proyecto en el plazo estipulado. El armado de la arena se redujo a 2 días, la prueba y depuración del sistema a 30 y la redacción de la documentación a 10. Las reducciones se compensarán con mayor dedicación horaria.

6.2.1 Resumen de Tareas pendientes y Asignación de Recursos

Taréa	**Duración (Dias)**	**Recursos**
Armado de la arena	2 (4)	Nicolas
Comunicación de spikes	5	Mariana
Prueba y depuración del sistema (60% pendiente)	30 (42)	Equipo completo
Documentación (Redacción y edición)	10 (20)	Equipo completo
Documentación (Revisión y corrección)	14	Equipo completo
Documentación (Presentación final)	12	Equipo completo

Tabla 7 - Resumen de tareas pendientes.

7. Cronograma detallado del Proyecto

Se adjunta el nuevo cronograma detallado del proyecto en el Apéndice A.

Referencias

[1] Arducam-página de referencia. www.arducam.com.

[2] Arduino-librería spi. http://www.arduino.cc/en/Reference/SPI.

[3] Imagen de onda pwm. http://www.hispavila.com/3ds/atmega/automatismo1_files/ondapwm.gif.

[4] Campbell J. Shukthankar R. Nourbakhsh I. Pahwa A. A robust visual odometry and precipice detection system using consumer-grade monocular vision. *Robotics and Automation, 2005. ICRA 2005*, pages 3421–3427, 2005.

[5] Atmel. *Cortex-M3 "SAM3X"*. http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf.

[6] Casswell Barry and Daniel Bush. From a to z: a potential role for grid cells in spatial navigation. *Neural Systems & Circuits*, 2(6), 2012.

[7] Johann Borenstein and Liqiang Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6):869–880, 1996.

[8] H. R. Everett. *Sensors For Mobile Robots, Theory and Application.* A K Peters, 289 Linden Street, Wellesley, 1995.

[9] Eugene M. Izhikevich. Simple model of spiking neurons. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 14(6):1569–1572, 2003.

[10] Omron. *EE-SX398/498(datasheet)*. http://www.omron.com/ecb/products/pdf/en-ee_sx398_498.pdf.

[11] Cuneo JI1. Quiroz NH. Weisz VI. Argibay PF. The computational influence of neurogenesis in the processing of spatial information in the dentate gyrus. *Sci Rep*, 2012.

[12] Edvard I. Moser Trygve Solstad and Gaute T. Einevoll. From grid cells to place cells: A mathematical model. *Hippocampus*, 16(12):1026–1031, 2006.

Glosario

Arduino Alta Es la placa Arduino encargada de la implementación de los modelos neuronales.

Arduino Baja Es la placa Arduino encargada de la implementación de la navegación.

Arquitectura de software Es el diseño de más alto nivel de la estructura de un sistema.

Bluetooth Es una especificación industrial para la comunicación de Redes Inalámbricas de Área Personal (WPAN).

Bumper Conjunto de parachoques + microswitches encargado de gestionar las colisiones.

Cortex Es el procesador en el cual está basado la Arduino Due.

DUE Hace referecnia a Arduino Due.

Encoder Conjunto de disco ranurado + optoacopladores. Es un dispositivo codificador rotatorio usado para convertir la posición angular de un eje a un código digital.

Grid cell Es un tipo de neurona que se encuentra en la corteza entorrinal de muchas especies y a la cual se atribuye la capacidad de un individuo de comprender su posición en el espacio.

IDE Del inglés *Integrated Development Environment*. Es un editor de código fuente.

Integrate and fire Es un modelo para el voltaje de membrana de una célula.

IUEMHI Instituto Universitario Escuela de Medicina del Hospital Italiano – Bs. As.

IZ Izhikevich. Es un modelo para el voltaje de membrana de una neurona.

Macro Una macro es una serie de instrucciones que se almacenan para que se puedan ejecutar de manera secuencial mediante una sola llamada u orden de ejecución.

Mega Hace referencia a Arduino Mega.

Nibble Es el conjunto de 4 *bits*.

Optoacoplador Es un dispositivo de emisión y recepción que funciona como un interruptor activado mediante la luz emitida por un diodo LED.

Pin Es la terminal de cada uno de los contactos metálicos de un conector o de un componente fabricado de un material conductor de la electricidad.

Place cell Son neuronas que se encuentran en el hipocampo y su principal característica es que muestran actividad cuando el animal se encuentra en un área particular del espacio.

Plug Conector eléctrico de alimentación.

Pull-up En electrónica se denomina pull-up a la acción de elevar la tensión de salida de un circuito lógico.

PWM Del inglés $Pulse\ Width\ Modulation$. Es un tipo de modulación de señales.

Python Es un lenguaje de programación interpretado.

Round Robin Es un tipo de arquitectura de software que se basa en un procesamiento secuencial.

Script Es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.

SPI Del inglés $Serial\ Peripheral\ Interface$. Es un estándar de comunicaciones.

Spikes Son los pulsos que se obtienen en base a las salidas de los modelos neuronales.

UART Del inglés $Universal\ Asynchronous\ Receiver\ Transmitter$. Es un protocolo de comunicación serial.

Índice de tablas

Índice de figuras

Esta es la última página.
Compilado el miércoles 30 septiembre, 2015.
http://iie.fing.edu.uy/