UNIVERSIDAD COMPLUTENSE DE MADRID / UNED

FACULTAD DE INFORMÁTICA

Proyecto Final de Máster

Máster en Ingeniería de Sistemas y Control

"Aprendizaje por Ejemplos e Instrucciones Aplicado al Lego NXT 2.0"

Autor: Alejandro Carpio Sánchez

Directores: Matilde Santos / José Antonio Martin

Septiembre 2012

Proyecto Final de Máster

Máster en Ingeniería de Sistemas y Control

Titulo:

“Aprendizaje por Ejemplos e Instrucciones Aplicado al Lego NXT 2.0”

Autor: Alejandro Carpio Sánchez

Directores: Matilde Santos / José Antonio Martin

El abajo firmante, matriculado en el Máster en Ingeniería de Sistemas y Control de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) y a la UNED, a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Aprendizaje por Ejemplos e Instrucciones Aplicado al Lego NXT 2.0", realizado durante el curso académico 2011-2012 bajo la dirección de Matilde Santos y José Antonio Martin de el Departamento de Arquitectura de Computadores y Automática.

Firmado:

Alejandro Carpio Sánchez

Resumen

La automática es un componente básico de prácticamente cualquier sistema o proceso, como medio para conseguir los resultados que actualmente podemos esperar del mismo. Los avances en este campo, y las necesidades de la sociedad, están forzando a la industria a hacer uso de controladores que emplean técnicas no convencionales para el control de dichos sistemas. Dentro este tipo de metodologías podemos encontrar los sistemas inteligentes, los cuales se hacen valer de disciplinas derivadas de la inteligencia artificial para realizar el control.

Este proyecto hace uso de un sistema de aprendizaje por refuerzo para lograr realizar el control de un robot Lego NXT en configuración de péndulo invertido, el cual se encuentra en un entorno segmentado en estados. El objetivo de nuestro control será conseguir encontrar la ruta más corta entre el estado inicial y el final. Para conseguir que el controlador realice dicha tarea el robot, utilizando la interacción con su entorno, aprenderá mediante exploración dicha ruta. Nuestro objetivo con el proyecto será acelerar el aprendizaje de la ruta óptima, al poder darle ejemplos o muestras de que podemos considerar una acción correcta para dicho estado, al estar nosotros en una posición privilegiada. De esta forma podemos incorporar nuestro conocimiento al proceso que estamos controlando, dentro del controlador inteligente. El principal objetivo con este proyecto es demostrar que podemos acelerar el aprendizaje de la solución con estos ejemplos.

Para lograr el objetivo principal implementaremos una serie de algoritmos de control, destinados a controlar los aspectos necesarios para realizar la tarea como son el control de estabilidad del robot. Al ser un robot en configuración de péndulo invertido, usaremos un controlador PID discreto, y también se hara el control de la posición y el movimiento, con las características propias de un robot de modelo diferencial.

Por todo esto, antes de empezar con el análisis del problema, haremos una revisión de las tecnologías y disciplinas sobre las que se sustenta el proyecto, hablando del aprendizaje por refuerzo, del kit de desarrollo Lego NXT Mindstorms, y acerca de ciertos aspectos de la ingeniería de control. Tras esto, haremos el diseño y desarrollo de la solución propuesta, para a continuación hacer una exposición y discusión de las pruebas hechas para validar el desarrollo. Finalmente hablaremos de las conclusiones a las que hemos llegado, y los aspectos en los que cabría un desarrollo posterior. El proyecto termina con la enumeración de las fuentes consultadas para su realización.

Palabras Clave:

Aprendizaje por refuerzo, Aprendizaje por Confirmación, Control Inteligente, Lego NXT, Controlador PID, Péndulo Invertido

ÍNDICE

Índice de Figuras

Índice de Tablas

Capitulo 1: Introducción

La automática es un componente básico de multitud de sistemas y procesos, sin la cual la mayoría de los mismos no podrían desarrollarse con todas las garantías de precisión, velocidad y calidad que ahora podemos esperar.

El presente proyecto busca dar una solución al problema del control automático de un robot basado en un kit Lego Mindstorm 2.0 en configuración de péndulo invertido en un entorno compuesto por 8 celdas. Para que el robot pueda hacer la tarea encomendada, se han tenido que implementar distintos tipos de controladores, basados en distintas tecnicas; siendo su objetivo final el lograr navegar por el interior del entorno, para llegar a un estado final, de la forma más eficiente.

Para lograr este objetivo utilizamos un sistema de aprendizaje por confirmación, el cual será el encargado de decidir cual es la siguiente acción a realizar para lograr realizar la tarea de forma optima.

1.1 Motivación

Este proyecto nace como una propuesta de los directores del proyecto Matilde Santos y José Antonio Martin, ambos del Departamento de Arquitectura de Computadores y Automática de la Universidad Complutense de Madrid, de utilizar un robot lego NXT Mindstorms 2.0 en configuración de péndulo invertido, para implementar un sistema de aprendizaje por confirmación. La idea me pareció atractiva ya que combinaba una gran cantidad de disciplinas diferentes; como son el control de robots diferenciales, la implementación de controladores sin modelo, la implementación de sistemas de aprendizaje automático para el control de sistemas, etc…

1.2 Objetivos

El objetivo principal del proyecto es conseguir demostrar que es posible acelerar el aprendizaje activo del robot dentro de un entorno, mediante el uso de una combinación de aprendizaje por refuerzo con aprendizaje supervisado. El objetivo de nuestro aprendizaje será que el robot aprenda a planificar las acciones que deberá realizar para conseguir transitar el entorno en el que habita de la forma más eficiente pudiendo nosotros darle información sobre qué acción debe realizar el controlador en cierta situación para acelerar este aprendizaje.

Tras esta definición inicial, un primer estudio de lo que buscábamos con el proyecto nos dio una serie de objetivos secundarios, derivados de lo que buscábamos conseguir con el proyecto. Las tareas, a grandes rasgos, que completaremos a lo largo del desarrollo del proyecto son las siguientes:

- Diseño de los **algoritmos de control de la estabilidad del robot**, al ser este un robot en configuración de péndulo invertido, y por lo tanto inestable. La solución tomada tiene como núcleo un controlador PID discreto, ajustado para linealizar el comportamiento del robot en dicha configuración.

- Creación de los **algoritmos de control de posición y movimiento en el seno del entorno** en el que habita el robot. Esto significa que debemos ser capaces de mover el robot a cualquier posición del entorno, lo cual requiere resolver una serie de problemas de posicionamiento y desplazamiento del robot para arquitectura. El entorno usado está compuesto por una serie de estados o celdas.

- Creación del **algoritmo de aprendizaje por confirmación basado en aprendizaje por refuerzo**, y demostración de que mediante el uso de ejemplos, el robot es capaz de acelerar su aprendizaje.

Todas estas tareas las hemos dividido en tareas más simples, que son definidas en la sección dedicada al análisis y especificación de la solución.

1.3 Estructura de la Memoria

El proyecto está dividido en distintos capítulos, pasando a comentar a continuación de qué trata cada uno:

- El **Capitulo 1** se corresponde con la introducción al problema que nos ocupa. Resume de forma el contenido del proyecto, y se corresponde con el actual capitulo. Además, se hace la exposición de la motivación y objetivos que perseguimos con este proyecto.

- El **Capitulo 2** presenta el estado del arte. En este capítulo hablaremos de forma general acerca de las tecnologías y disciplinas utilizandas en el proyecto, como el aprendizaje por refuezo, su uso para el control de sistemas, del kit de desarrollo Lego NXT, sobre el cual nos basamos para implementar la planta objetivo de nuestro control; y acerca de ciertos aspectos de la ingeniería de control, hablando del control PID y del control de un péndulo invertido.

- El **Capitulo 3** corresponde a las especificaciones del sistema. Aquí expondremos los requisitos de funcionamiento del robot, los comportamientos esperados, y el entorno en el que se mueve nuestro robot. A continuación, plantearemos la solución que nosotros damos para esos requisitos.

- El **Capitulo 4,** se desarrolla la solución. Aquí veremos cómo se ha implementado finalmente todo lo comentado en el capitulo anterior.

- El **Capitulo 5** corresponde a las pruebas funcionales del sistema. Aquí hablaremos en primer lugar de las pruebas que consideramos que deberíamos realizar para cerciorarnos de que el sistema realiza lo especificado en el capítulo 4. Después, hablaremos acerca de los resultados reales de dichas pruebas, y discutiremos su significado.

- El **Capitulo 6**, se corresponde con la exposición de las conclusiones a las que hemos podido llegar con la realización del proyecto; y la exposición de los resultados del proyecto. Además, plantearemos los puntos que tras todo este análisis pensamos que podrían ampliarse o mejorarse con un posterior desarrollo.

- Por último, se incluye la exposición de la **Bibliografía** consultada, así como las referencias a estas indexadas a estas en el texto; y los **Anexos**. Aquí podremos ver el código implementado en Matlab, así como el algoritmo NXT-G implementado en el bloque NXT.

Capitulo 2: Estado de arte

Este capítulo lo vamos a dedicar las técnicas y disciplinas aplicadas en el proyecto. Se divide en tres partes:

- La primera habla acerca del aprendizaje por refuerzo, lo que es, que características tiene, y los elementos intervinientes. Además se hace una comparación con el aprendizaje por ejemplos, las diferencias, y los problemas/ventajas que tienen ambos.

- La segunda describe el sistema o planta sobre la cual aplicamos nuestros algoritmos de control: el kit Lego Mindstorms NXT. Hablaremos de sus características, los sensores y actuadores disponibles, y sobre las alternativas de las que disponemos para su programación, centrándonos en los que hemos usado a lo largo del proyecto.

- La tercera trata de la ingeniería de control asociada al proyecto. Hablaremos acerca del control PID, y del control de un péndulo invertido.

Hay que tener en cuenta que intentaremos hacer este capítulo lo más divulgativo posible, profundizando lo suficiente para que se entienda cada punto, pero simplificando en la medida de los posible aquellos aspectos que no sean completamente necesarios para entender en su totalidad el proyecto, y indicando las referencias apropiadas donde encontrar información más detallada.

2.1 Aprendizaje por Refuerzo

2.1.1 Introducción

Cuando hablamos de aprendizaje, uno de los primeros ejemplos que nos viene a la cabeza es el de aprendizaje mediante la inmersión e interacción con un entorno, a partir del cual el aprendiz puede conocer las consecuencias de sus acciones, y que debe hacer para conseguir sus objetivos. El aprendizaje por refuerzo [RUSS04, SUTT98] es una aproximación a este paradigma del aprendizaje mediante la interacción con el entorno, con un enfoque a conseguir objetivos y metas.

Se trata de un aprendizaje que tiene como objetivo maximizar una recompensa representada mediante un valor numérico. El agente no sabe a priori cuales son las acciones que debe tomar en cada situación y, en su lugar, debe aprender mediante exploración cuales son las que le proporcionan la máxima recompensa. Estas acciones normalmente afectan a la situación actual del agente respecto a su entorno, lo cual condiciona las recompensas que podrá obtener en un futuro.

La principal característica del uso de este sistema para la resolución de problemas es que, mediante la definición de un objetivo (junto al resto de parámetros de los que hablaremos a continuación), el agente es capaz de obtener una solución a partir de su exploración. Esto lo diferencia del aprendizaje supervisado, el cual ha sido objeto de estudio ampliamente [RUSS04, NILS01], ya que el aprendizaje supervisado intenta obtener sus resultados mediante la presentación de ejemplos por parte de un entrenador externo. Este paradigma no es útil para aquellos entornos en los que las premisas y principios que definen el mundo hacen que, difícilmente, un entrenador pueda dar ejemplos del comportamiento que puedan ser generalizados al resto de situaciones. Es para estos problemas donde la resolución mediante aprendizaje por refuerzo nos puede resultar de utilidad, al ser el agente el que, mediante la exploración, irá aprendiendo. Para conseguir buenos resultados deberemos conseguir un buen compromiso entre exploración y explotación de los resultados. Un agente se verá tentado a explotar repetidamente aquellas acciones que realizó en una cierta situación, y de la cual obtuvo una recompensa positiva. No obstante, el hecho explotar dichas acciones puede estar

provocando que no sea capaz de encontrar otras acciones que, de ser explotadas, le reportarán una recompensa mayor (ya sea a corto o largo plazo). Es por ello que el agente debería explotar aquello que ya ha realizado, y le ha resultado útil; pero además debe realizar cierto grado de exploración para obtener información del resto de acciones posibles.

El problema existente es que nuestro agente deberá cometer repetidos errores tanto si quiere explorar como explotar las acciones. Por ello lo más útil sería inicialmente ir probando distintas acciones, para ir progresivamente realizando con mayor frecuencia las acciones que parecían ser mejores. Este problema del que hablamos es algo que no existe en el aprendizaje supervisado como tal.

Existen múltiples ejemplos de aplicaciones de los algoritmos de aprendizaje por refuerzo en los ámbitos académicos y universitarios [RUSS04]. En el ámbito industrial y empresarial se han empezado a ver algunos ejemplos de plantas que hacen uso de estas técnicas, primordialmente para la optimización de parámetros de operación. Un ejemplo fue su uso en la optimización del control de la supervisión para un proceso de Horno Rotativo, implementado sobre el DCS Foxboro I/A Series[Xiao06].

Una característica que es muy importante que tengamos en cuenta de los sistemas de aprendizaje por refuerzo es el hecho de ser altamente realimentados (figura 2.1). Cuando el sistema obtiene información del entorno para evaluar las acciones tomadas, el sistema puede evaluar cuanto de buena ha sido la acción tomada. Lo que no puede saber es si los resultados obtenidos son los mejores o los peores de entre los posibles. Cuando esta realimentación es capaz de modificar los comportamientos futuros, hablamos de aprendizaje activo, y es esta realimentación la que le permite su funcionamiento como método de optimización.

Como decíamos, no podemos saber cómo de buena o de mala es una acción respecto a las demás si no es probándolas todas, y luego comparando. Esto hace que tengamos que definir una forma de almacenar esta información, y hacerla utilizable por parte de nuestra política de elección de acción. En un entorno de aprendizaje supervisado, esta realimentación con el entorno nos diría exactamente cuál hubiera sido la acción que deberíamos hacer realizado, lo cual elimina cualquier posible exploración.

En el caso del aprendizaje por refuerzo, el agente es instruido por el entorno, el cual no le va a dar la respuesta óptima.

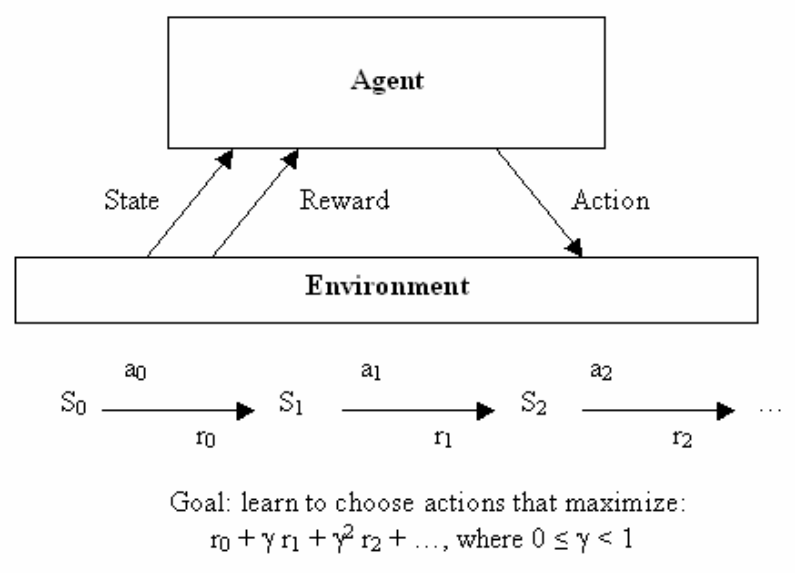


Figura 2.1. Diagrama del proceso de actuado/realimentación.

2.1.2 Elementos intervinientes

Además del agente y el entorno, en el ámbito del aprendizaje por refuerzo tenemos otros elementos a tener en consideración dentro de nuestro sistema:

- **La política.** Esta define el comportamiento (es decir, la próxima acción) del agente para la situación en la que se encuentra. Esta puede implementarse de multitud de formas, puede obtenerse como resultado de la ejecución de una función, extraer el dato de una tabla; o mediante la realización de procesos o cálculos mucho más complejos.

- **La función de recompensas.** Esta es la que define de forma explícita el objetivo de nuestro problema de aprendizaje por refuerzo. El objetivo que tiene nuestro agente es maximizar la recompensa total que adquirirá a lo largo de su vida. La función de recompensas es la que define la bondad (de forma positiva o negativa) de cada estado. La utilidad de un estado debe ser, a priori, inalterable a lo largo de la vida del agente, valiéndose de ésta para aprender.

- **La función de valor.** Esta función dice, para un estado, cómo esperamos que sea de útil a largo plazo. De forma general define la recompensa acumulada que nuestro agente puede esperar recibir en el futuro a partir de dicho estado. Obviamente no es lo mismo que la recompensa, ya que un estado con una pequeña recompensa puede ser el principio en el camino que define, por ejemplo, la mejor ruta hacia un destino; y viceversa. En nuestro aprendizaje, la función de valor es una pieza clave de nuestra política ya que nos permite estimar la recompensa a largo plazo que obtendremos para un estado, ya que nosotros deberemos elegir en nuestra explotación aquellas acciones que nos lleven a nuestra meta, obteniendo la máxima recompensa. Esta estimación, y la forma de calcularlo, es la pieza clave de los algoritmos de aprendizaje por refuerzo.

- **El Modelo del mundo.** Este elemento define una representación del mundo en que el agente existe. Por ejemplo, para un conjunto de estados, y una realización de una acción de entre las posibles; nuestro modelo del mundo puede predecir cuál será el siguiente estado y la próxima recompensa. Estos modelos pueden ser útiles en aquellos casos en que la función de valor busca saber información de las posibles situaciones futuras, sin tener que experimentarlas.

2.1.3 Elección de parámetros

Como ya hemos comentado, el cálculo correcto de una función de valor que "aprenda" es la piedra angular de nuestro sistema de aprendizaje por refuerzo. Por ello, necesitamos almacenar una suerte de recuerdo de las recompensas que se han ido obteniendo a lo largo de nuestra selección de las acciones. El almacenaje de todas y cada unas de las acciones, junto con su recompensa asociada, para su posterior consulta como herramienta en nuestra toma de decisiones, es inviable tanto desde el punto de vista del tiempo de cálculo, como de la capacidad de memoria del sistema que ejecuta el algoritmo. Es por ello que se definen funciones que se actualizan de forma incremental con cada nueva recompensa. De forma general las funciones [SUTT98] son de la forma:

$$NuevoValor \leftarrow \text{AntiguoValor} + \text{Const} * (\text{Recompensa} - \text{AntiguoValor}) \quad (1)$$

El sistema de aprendizaje por refuerzo se define de tal forma que el agente y el entorno interactúan en pasos temporales discretos.

(t= 1,2,3,4….). En cada instante de tiempo discreto, el agente recibe el estado en el que se encuentra s_t, donde $s_t \in S$, siendo S el conjunto de los posibles estados en que nuestro agente puede encontrarse. En base al estado en que se encuentra, seleccionará mediante el uso de la política una acción de entre las posibles para ese estado, $a_t \in A(s_t)$. Hay que tener en cuenta que no todos los estados necesariamente tienen las mismas acciones. Después, el agente recibe la recompensa numérica a su acción $r_{t+1} \in R$, y en ese instante se encuentra en el nuevo estado s_{t+1}. Estos intervalos temporales normalmente no suelen ajustarse a periodos de tiempo real, sino a los eventos de decisión y actuación.

Por lo que hemos comentado, podemos ver que las características de estos sistemas los hacen altamente versátiles. Por ejemplo, podríamos definir el sistema de tal forma que los estados estén asociados a valores de voltaje, presión, temperatura, nivel, etc.… y las acciones asociadas pueden estar conducidas a ir hacia estados en los que estas variables de proceso aumenten o disminuyan para ir hacia un valor de consigna. De esta forma podríamos llevar a crear perfectos sistemas de control realimentados que aprendieran de su entorno la solución a problemas en los que, inicialmente, un operador humano no tuviera una solución clara, por solo tener un conocimiento parcial del proceso o planta.

En el caso de la función de recompensas, el objetivo del agente está expresado en términos de recompensas inmediatas, siendo estos valores numéricos $r_t \in R$, queriendo nuestro agente maximizar su consecución a largo plazo. Como lo que el agente aprende es a maximizar sus recompensas, si queremos que el agente haga algo debemos proporcionarle recompensas que le lleven a ello. De esta forma, nosotros sabemos qué queremos que nuestro agente consiga, pero no sabemos cómo puede conseguirlo. La recompensa final [RUSS04] de nuestro agente será la suma de todas las de los estados por los que ha tenido que pasar para llevar a su objetivo:

$$R_f = r_0 + r_1 + r_2 \ldots + r_T \qquad (2)$$

Donde el estado T es un estado terminal.

En la presente memoria hemos omitido hablar sobre los Procesos de Decisión de Markov (MDP) [SUTT98], o el aprendizaje por Diferencias Temporales [SUTT98]; y su relación con el aprendizaje por refuerzo, para poder centrarnos en los aspectos más prácticos de los sistemas de aprendizaje por refuerzo. Es por ello que vamos a hablar ahora del algoritmo SARSA, que será el fundamento de nuestro algoritmo de aprendizaje por refuerzo.

2.1.4 Algoritmo SARSA

Como ya hemos comentado, el cálculo de la función de valor es determinante para que el algoritmo realice su tarea. La mayoría de algoritmos de aprendizaje por refuerzo se basan en estimaciones de la función de valor, que estiman las recompensan futuras que espera un agente por encontrarse en un estado, o cuanto de bueno es realizar una acción en dicho estado. Es esto último lo que realiza el algoritmo Q-Learning [SUTT98]. El algoritmo define una función que da a cada acción a_t del estado s_t, bajo el paraguas de usar la política actual de decisión de acciones, un valor numérico. Esta función se estima mediante la experiencia. El algoritmo SARSA [APRAUT] define la función Q(S,A) como:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (r_{t+1} + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (3)$$

Esa función actualizará en cada transición el valor para ese $Q(s_t, a_t)$, para cualquier valor de s que no sea un estado terminal. Este algoritmo hace uso de la tupla $s_t, a_t, r_t, s_{t+1}, a_{t+1}$, la cual da nombre a este algoritmo (SARSA).

El algoritmo hará lo siguiente:

Se Inicializa $Q(s_t, a_t)$ de forma aleatoria

Repetir

Inicializar s_t

Escoger a_t de s_t usando la política derivada de Q

Repetir

Tomar acción a_t. Tras esto, observar r_t y s_{t+1}

Escoger de a_{t+1} de s_{t+1} usando la política derivada de Q

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (r + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$

$s_t \leftarrow s_{t+1}$

$a_t \leftarrow a_{t+1}$

hasta que s_t sea un estado terminal

Algoritmo 1: SARSA

Donde α es el factor de recompensa inmediata, y γ es el factor de recompensa futura. Ambos parámetros pueden tomar valores entre 0 y 1. La función $Q(s_t, a_t)$ convergerá hacia una solución óptima, por lo cual, al ser ésta una parte primordial en la elección de la siguiente acción, se podrá derivar una política optima que resuelva el problema.

2.2 El Sistema Lego Mindstorms NXT

2.2.1 Introducción

El sistema Lego Mindstorms NXT [NXTINTR] es un kit de construcción fabricado por la compañía Lego, que aunque sea vendido como un juguete (y que por tanto está destinado a niños), sus capacidades y su amplio desarrollo lo han convertido en una herramienta de investigación y docencia en el ámbito de la robótica y el control.

El primer del kit que se asemejaba al Lego Mindstorms se comercializó en el año 1998 a partir de una colaboración con el MIT. Este contaba con un bloque programable RCX, el cual incluía dentro un micro controlador, al cual ya era posible cargarle los programas que había creado el usuario, y que estos se ejecutaran sobre la arquitectura de sensores y actuadores del robot. En 2006, tras sacar distintas versiones del bloque RCX, Lego comercializa Mindstorms NXT con el actual bloque NXT.

Figura 2.2. Bloque RCX y NXT

Las características del NXT son:

- Procesador ARM-7 de 32 bits (AT91SAM7S256)
- 256 KB de memoria FLASH para programas/firmware
- 64 KB de RAM
- 3 puertos de salida
- 4 puertos de entrada
- Pantalla LCD
- Bluetooth para comunicación PC ↔ NXT y NXT↔NXT
- Puerto USB
- Altavoz con calidad de 8Khz. Canal de sonido de 8 bits, sample rate 2 a 16 khz.
- Alimentación a 66 pilas AA/ Bateria recargable de litio.

El firmware que lleva el bloque NXT por defecto permite crear programas desde el software que acompaña al kit, el software NXT-G, desarrollado por National Instruments. Su interfaz se asemeja a Labview, permitiendo la programación gráfica mediante bloques. Existen lenguajes de programación basados en código, como el NBC (Next Byte Codes) o NXC (Not eXactly C). Además, existen kits de desarrollo para los lenguajes más comunes (Ada, Python, C, Matlab, Lua o Java).

En el apartado de análisis hablaremos con mayor profundidad de los entornos de desarrollo elegidos (NXT-G para el control de la estabilidad posición, y el kit RWTH para Matlab como enlace con los algoritmos de aprendizaje por refuerzo).

2.2.2 Sensores y Actuadores

Hablaremos ahora de los **servomores**. Estos proporcionan la fuerza motriz a los robots del NXT. Cada uno dispone de un encoder rotacional para realizar el control de posición. Se pueden obtener datos de los grados o las rotaciones completas (con una resolución de ±1 grado).

Los **sensores básicos** de los que dispone el kit son **sensores de contacto** (que da una salida digital que permite saber cuándo es presionado), **sensores de sonido** (un micrófono), **sensores de Luz**

(permite detectar luz y color, con ciertas limitaciones), y **sensores de ultrasonidos**.

Este **sensor de ultrasonidos** es capaz de proporcionar información relevante al robot (junto a la que le proporciona el sensor de luz). Permite al robot detectar objetos y medir distancias, lo que permitirá al control de movimiento sortear objetos y detectar movimientos. Internamente el sensor está formado por 2 transductores de ultrasonidos, uno usado para emitir y otro para recibir. Se trata de un sensor complejo por lo que utiliza su propio microprocesador (embebido dentro del sensor). Este sensor funciona de la misma forma que un sonar, enviando un pulso de ultrasonidos de 40kHz y mide lo que tarda la onda en viajar, chocar contra un objeto, y volver. El tiempo será proporcional a la distancia a la que se encuentra el objeto. El sensor está ajustado para medir distancias entre 0 y 255 cm con una precisión de ±3 cm. Hay que tener en cuenta que la forma del objeto, y el material del mismo pueden dificultar (o imposibilitar) la detección del objeto.

Aparte de estos sensores básicos, producidos por Lego, existen otros sensores avanzados. Estos normalmente utilizan una carcasa muy similar a la que utiliza Lego. El fabricante más importante de sensores para Lego Mindstorms es HiTechnic [HITECH], el cual es el fabricante de sensores como la brújula, el sensor de infrarojos, el sensor electro-optico de proximidad, o el giroscopio.

El Giroscopio contiene un sensor giroscópico en un solo eje que detecta, en el eje en el que opera, la velocidad de rotación en grados por segundo. Puede medir un máximo de ±360º por segundo.

Figura 2.3. Giroscopio NGY1044 para Lego MindStorm

El ratio de lectura puede ser de hasta 300 veces por segundo, lo cual permite su uso en algoritmos de control muy exigentes con el tiempo de respuesta.

2.2.3 Entornos de Programación

Como ya comentábamos en la introducción, hay una gran cantidad de entornos de programación para el sistema NXT: NXT-G, Labview, RoboC, Lejos. Hablaremos acerca de los que son importantes en el presente proyecto: NXT-G y RWTH para Matlab. Hablar acerca del resto sería una labor muy interesante, pero no es el propósito de este proyecto. Una comparativa puede ser encontrada en [LENGCOMP]

2.2.3 .1 NXT-G

NXT-G [NXTGPR] es el software que proporciona Lego para programar los robots creados con el kit de forma sencilla, mediante USB o Bluetooth. El programa ha sido generado por National Instruments, y su diseño ha tomado como base Labview. Por ello la programación se hace mediante bloques, los cuales se conectan unos a otros, para definir el algoritmo de control (Figura 2.4). Cada bloque realiza una tarea especializada (leer de un sensor especifico, actuar sobre un motor, realizar un cálculo,...), y es la combinación de todos ellos la que realiza la tarea final.

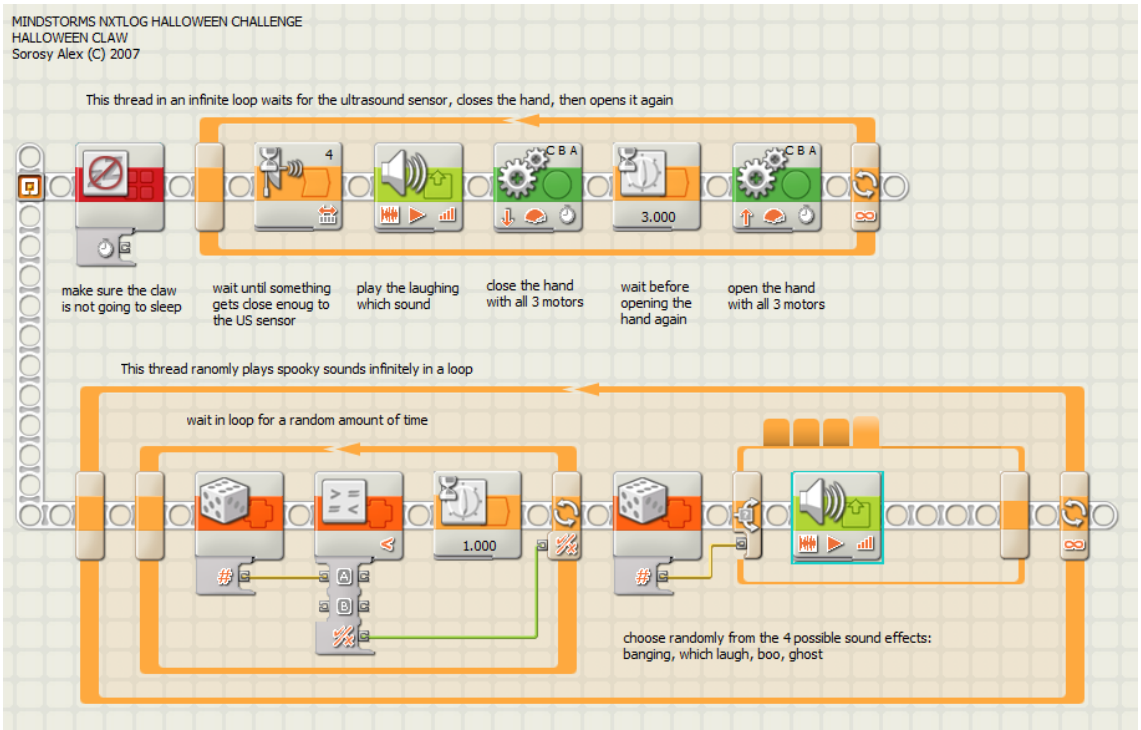


Figura 2.4. Entorno de desarrollo NXT-G

Existen 3 paletas o interfaces: común, completa y a medida. La paleta común dispone de 7 bloques: movimiento, grabación/reproducción, sonido, pantalla, Wait, bucle y condición. La paleta completa agrupa los bloques en categorías según su función: lectura de sensores, actuadores (motores y altavoz), control del flujo, comunicaciones.... También incluye un grupo de bloques con las operaciones matemáticas básicas, y las operaciones lógicas, además de poder crear variables numéricas, lógicas o de cadena de texto.

Este software, y su programación, hacen muy sencillo trabajar con el sistema NXT. Pero, cuando queremos realizar algoritmos de control más complejos o con requisitos especiales, este software se queda corto muy pronto. Da problemas para realizar el seguimiento de programas muy grandes, ya que aunque es posible crear bloques personalizados (que se asemejan a los SubVI de LabView), la interacción entre el programador y el software es lenta y forzada, lo que provoca que se pierda el hilo del algoritmo muy fácilmente. Esto, unido a que la cantidad de bloques de los que dispone el software de diseño es muy limitada, hace que diseñar algoritmos de control sofisticados sea una tarea descomunal. Si bien, como veremos en la parte de análisis, es posible implementar algunos algoritmos de control, relativamente sencillos que nos han dado unos resultados muy satisfactorios, como es el caso de un PID discreto.

Para la mayoría de los ámbitos académicos, el uso de NXT-G da la posibilidad de acercar a los alumnos (prácticamente de cualquier nivel formativo, desde la secundaria a los niveles de postgrado) una forma fácil de implementar y probar algoritmos de supervisión y control, eliminando muchas de las complejidades inherentes al hardware de los sistemas controlados. Y es esta cualidad la que motiva que busquemos una forma de ampliar sus posibilidades por algún camino. Es aquí donde entra el kit de desarrollo RWTH.

2.2.3 .2 RWTH – Mindstorms NXT ToolKit

El RWTH es un Toolkit para Matlab [RWTHW] para el control de robots basados en NXT. Permite la comunicación on-line con el sistema NXT a través del USB o Bluetooth. En esencia es un paquete de funciones para Matlab a las cuales podemos llamar para, una vez establecida la comunicación con el equipo, obtener o enviar datos directamente al NXT. De esta forma podemos hacer un control distribuido del robot, haciendo por ejemplo el control a bajo nivel en el propio procesador del robo, y dejar los algoritmos de control complejos (o cualquier otro algoritmo necesario para nuestra tarea) al PC. Esto nos va a permitir no depender de las capacidades computacionales del NXT, ni del desarrollo de algoritmos específicos para ser ejecutados en el NXT. Podremos hacer uso de todas las capacidades de Matlab (y todo su desarrollo y Toolkits) para usarlos en el control de nuestro robot. Así pues, las ventajas derivadas del uso de RWTH son:

- Aumento de la capacidad de CPU y memoria disponible para el uso de algoritmos de control del robot NXT.
- Tamaño del programa, en la práctica, ilimitado.
- Posibilidad de controlar, y coordinar de forma externa varios robots NXT.
- Posibilidad de obtener datos del robot NXT, mientras seguimos corriendo un programa clásico del NXT; sin obtener importantes penalizaciones en el tiempo de cálculo.
- Extensión de los periféricos que podemos utilizar dentro de un sistema que integre sistemas NXT, a los periféricos con los que podemos comunicarnos desde Matlab.

RWTH nos permitirá crear un programa basado en NXT-G que esté corriendo en el bloque NXT; y mientras establecer una comunicación con el equipo por Bluetooth o USB, para poder enviar o recibir órdenes con el sistema. Si bien un aspecto que deberemos tener muy en cuenta es que no podremos acceder a las variables locales que

utilicemos en el programa NXT-G. Esto es un serio problema ya que necesitaremos con total seguridad poder acceder al espacio de memoria del algoritmo que está corriendo, para poder hacer nuestra tarea.

Una solución a este problema será dada en la sección de análisis, mediante el uso de los buzones NXT.

2.3 Ingeniería de Control

2.3.1 El Péndulo Invertido

El control de un Péndulo Invertido es uno de los problemas clásicos que cualquier estudiante de ingeniería de control [OGATA02] (de cualquiera de sus subdisciplinas) debería plantearse, ya que por sus características, se trata de una planta muy instructiva. Un ejemplo clásico de implementación de la planta es el uso de una varilla, que puede caer en cualquiera de los sentidos en uno de sus ejes, sobre un vehículo o sistema móvil, el cual se mueve en dicho eje bajo una fuerza. El péndulo invertido es un sistema inestable, ya que la varilla puede caer en cualquier momento, salvo que apliquemos un desplazamiento del vehículo solidario, que provocara con su desplazamiento un par de fuerzas que lo elevará. Dicha fuerza es por tanto la variable manipulada, y el ángulo de la varilla respecto a la horizontal la variable controlada.

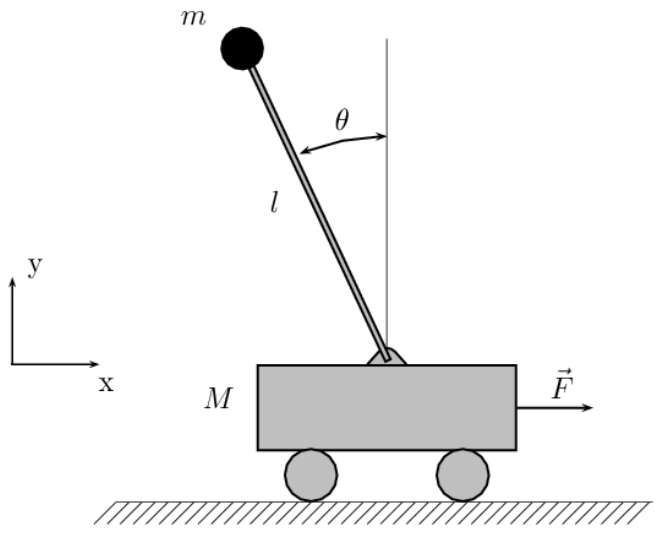


Figura 2.5. Diagrama Péndulo sobre Carro

La resolución de este problema ha sido abordada desde prácticamentetodos las facetas de la ingeniería de control. Existen soluciones basadas en control clásico, control moderno [OGATA02] e incluso control inteligente [INTPEND] (este último, normalmente, sin modelo).

2.3.2 El robot NXT en configuración de Péndulo Invertido

En el caso del NXT[NXTTWO], normalmente se implementa mediante el uso de un robot con 2 ruedas, por lo que se trata de un robot de modelo diferencial. El uso de un robot así introduce algunas nuevas variables en el modelo que define el robot, asociadas a los 2 ejes de giro que tiene la nueva planta, así como los momentos de inercia de los motores.

Nuestro objetivo no es hablar en profundidad del modelo que define un sistema de péndulo invertido, ya que aunque obviamente se trata de un tema muy interesante, este proyecto no lo requiere, por utilizar métodos de control no basados en modelos.

Desde el punto de vista mecánico, el uso del kit Lego Mindstorms nos permite crear robots en configuración de péndulo invertido en infinidad de formas constructivas. Si tuviéramos que modelar todas y cada una de las formas constructivas de robots que quisiéramos probar, tendríamos un serio problema, ya que el modelado matemático del sistema no es una tarea trivial. La solución a este problema recae en el uso de controladores sin modelo.

Las técnicas que actualmente están a la cabeza de este tipo de control son las técnicas de Control Inteligente, que se hacen valer de técnicas como son la Lógica borrosa, las redes neuronales o los algoritmos evolutivos; para realizar el control de sistemas que requerirían de técnicas de control multivariable, o que directamente, no podrían ser abordados por las técnicas de control clásico o moderno.

El segundo caso es el uso de controladores no basados en modelo, que mediante el uso de la realimentación, pueden llevar el proceso en la dirección deseada, tras un trabajo de sintonizado para ajustar el controlador de tal forma que se aproxime al modelo real de

la planta. Hablaremos a continuación de uno de los controladores de este tipo que son más comunes en los lazos de control de la mayoría de industrias del mundo: El controlador PID; el cual será la base del control de estabilidad y posición en nuestro robot.

En cualquier caso, cuando hablemos del AnyWay en el capítulo de análisis y desarrollo, se expandirá la información relativa al uso del PID para el control de un robot NXT.

Figura 2.6. Fotografía AnyWay

2.3.3 El Controlador PID

Estos controladores aparecen en muchas aplicaciones, y son unos de los pilares del control de procesos industriales en la actualidad. [PIDEXAMP]

Las primeras estructuras de control ya presentaban ciertas características que luego aparecerían en el control PID, pero sin embargo, no fue hasta el siglo XIX cuando el control PID tomó importancia. Todavía a día de hoy, aunque disponemos de una gran variedad de formas de diseñar y modelar sistemas de control avanzados, el control PID [PIDCONTR] es utilizado en algún punto de prácticamente todos los procesos industriales.

Las siglas PID hacen referencia a las tres partes que lo forman: una parte proporcional, una parte Integral, y una parte Derivativa. El lazo de control de un PID es un lazo de control SISO de un grado de libertad:

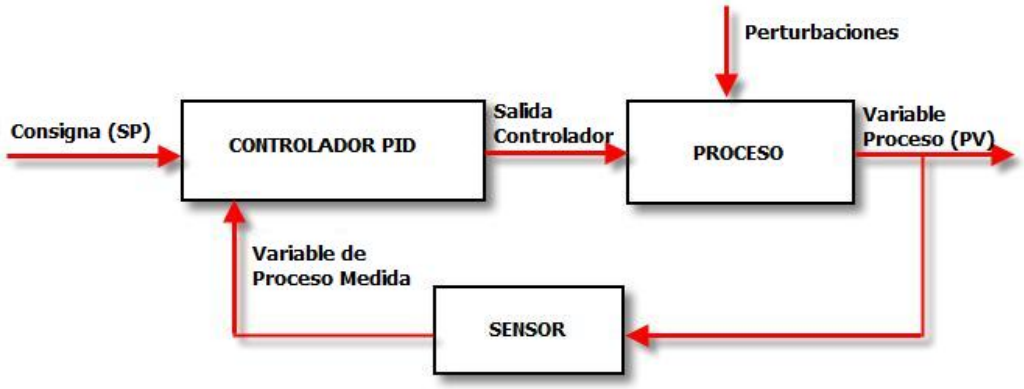


Figura 2.7. Diagrama de bloques de un lazo PID

Así pues, el lazo PID está constituido por:

- El **controlador PID** que implementa un sistema lineal con una determinada función de transferencia, que actúa en función del error entre el valor medido de la variable de proceso (PV) y el valor que se quiere obtener (SP), o de referencia.

- **Un sensor,** que determine el valor que toma la variable de proceso (PV).

- Algún tipo de **elemento actuador**, que permita la manipulación del proceso de forma controlada mediante la salida del controlador PID.

En este sistema, el controlador obtiene un valor de consigna (o SetPoint), que está escalado en el mismo rango de valores y es de la misma naturaleza que la señal que le está proporcionando el sensor. El controlador restará ambos valores, lo que le da como resultado una señal de error, y que determina la diferencia que hay entre ambos (lo que quiero obtener, y lo que tengo realmente) de forma analítica. Esta señal de error será utilizada por cada uno de los 3 componentes del controlador PID para obtener su salida parcial, que sumadas dará la salida total del controlador PID, y que será utilizada para actuar sobre el sistema. Los tres elementos que componen un controlador PID son la acción Proporcional, Integral y Derivativa. El peso que estas tendrán sobre la salida final se puede modificar mediante la variación de la constante proporcional, del tiempo integral y del derivativo; respectivamente. El objetivo es que mediante la sintonía de estas 3 componentes el controlador sea capaz de corregir de forma eficaz el error, en el menor tiempo que sea posible.

En muchos casos el proceso a controlar no requiere de las 3 partes para poder ser gobernado de forma eficiente y eficaz, así que podemos tener controladores P, PD o PI de forma muy común.

La acción de **control Proporcional** es de los tipos de control más simples, ya que da una salida proporcional al error recibido y que se puede expresar analíticamente mediante la ecuación:

$$P_{\mathbf{sal}} = K_p\, e(t) \tag{4}$$

donde K_p es la ganancia proporcional del controlador. La parte proporcional no considera el tiempo, lo cual provoca la aparición de un error permanente. Cuanto mayor sea K_p , menor será este error, pero la respuesta del sistema será más oscilatoria. Si este error no se puede reducir hasta un nivel asumible, la mejor manera de solucionar dicho error es hacer que el sistema de control contenga alguna de las otras dos componentes, configurando acciones integral y/o derivativa.

La acción de **control Integral** busca eliminar el error que se nos presenta en estado estacionario al utilizar el modo proporcional. Se puede expresar analíticamente como:

$$I_{\mathbf{sal}} = K_i \int_0^t e(\tau)\, d\tau \tag{5}$$

El control integral actuará siempre que exista algún tipo de desviación entre la variable de proceso, y el punto de consigna, integrando dicha desviación en el tiempo, para sumársela al valor de salida de la acción proporcional. El error es integrado, lo cual tiene el objetivo de aplicar un promediado, o sumarlo por un período de tiempo determinado. Después de aplicar este valor, es multiplicado por una constante K_i.

La acción de **control Derivativo**, actúa cuando existen cambios en el valor absoluto del error. Se puede expresar analíticamente como:

$$D_{\mathbf{sal}} = K_d \frac{de}{dt} \tag{6}$$

La acción derivativa busca corregir el error a la misma velocidad a la que este está apareciendo en el sistema, evitando así que el error pueda aumentar.

Así, el control PID tiene la capacidad de eliminar el error que se pueda dar en el estado estacionario mediante la acción integral, y puede anticiparse al futuro con la acción derivativa. El controlador PID es un elemento importante dentro de los sistemas de control distribuidos. Estos controladores están también presentes en muchos sistemas de control específicos, como puede ser el control de velocidad de un coche.

Definiendo u(t) como la salida del controlador, la ecuación que define a un sistema PID es:

$$\mathrm{u(t)} = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de}{dt} \tag{7}$$

Cuando el controlador PID es implementado en un sistema basado en micro-controlador, o en cualquier soporte que requiera discretizar el sistema, las ecuaciones que se implementan son:

$$u_p(k) = K_p e(k)$$

$$u_i(k) = u_i(k-1) + K_p \frac{T}{T_i} e(k)$$

$$u_d(k) = K_p \frac{T_d}{T}(e(k) - e(k-1))$$

$$K_i = K_p \frac{T}{T_i}$$

$$K_d = K_p \frac{T_d}{T} \tag{8}$$

donde, T es el periodo de muestreo utilizado, K_p,K_i y K_d son constantes de la acción proporcional, integral y derivativa; y, T_i y T_d, el tiempo integral y derivativo. Se ha utilizado la notación "backward" para la derivada.

Capitulo 3: Análisis del Sistema

El problema que nos plantean busca explotar un escenario de aprendizaje por confirmación. El aprendizaje por confirmación es un tipo de aprendizaje supervisado, en cooperación un aprendizaje por refuerzo; donde el instructor da las respuestas que considera adecuadas a las preguntas que hace el aprendiz, y donde este aprendiz intentara repetir dichas respuestas. La idea es que cuando el aprendiz repite la respuesta, introduce una variación (como parte de su exploración), para, por ejemplo, averiguar cuánto margen de error o tolerancia puede aceptar el instructor.

Algunas características del aprendizaje por confirmación son que permite un aprendizaje online y activo, y además, explota las ventajas que tiene el aprendizaje supervisado y el aprendizaje por refuerzo, suavizando algunos de sus problemas al complementarse el uno al otro.

Sus ventajas son que permite una fuerte generalización, facilita y acelera el entrenamiento, y permite al instructor especificar sus preferencias, para que el aprendiz, dentro de su margen de maniobra, actúe en cierta forma como espera su instructor.

3.1 Análisis de los Requisitos

En este proyecto se especifica que el aprendiz es un robot NXT 2.0 en configuración de péndulo invertido, el cual debe ser capaz de aprender la ruta óptima para ir desde el punto en el que se encuentra, a un punto de destino (prefijado e invariable), utilizando en su aprendizaje el aprendizaje por confirmación, demostrando que el uso de el aprendizaje supervisado dentro del algoritmo de aprendizaje por refuerzo es capaz de acelerar el aprendizaje, y que además somos capaces de influir en la toma de decisiones del robot,¡ para que elija de entre las rutas posibles, aquella que nosotros habíamos elegido.

El escenario donde se encuentra el robot es un mundo con 9 celdas en disposición 3x3, y el cual tiene la forma siguiente:

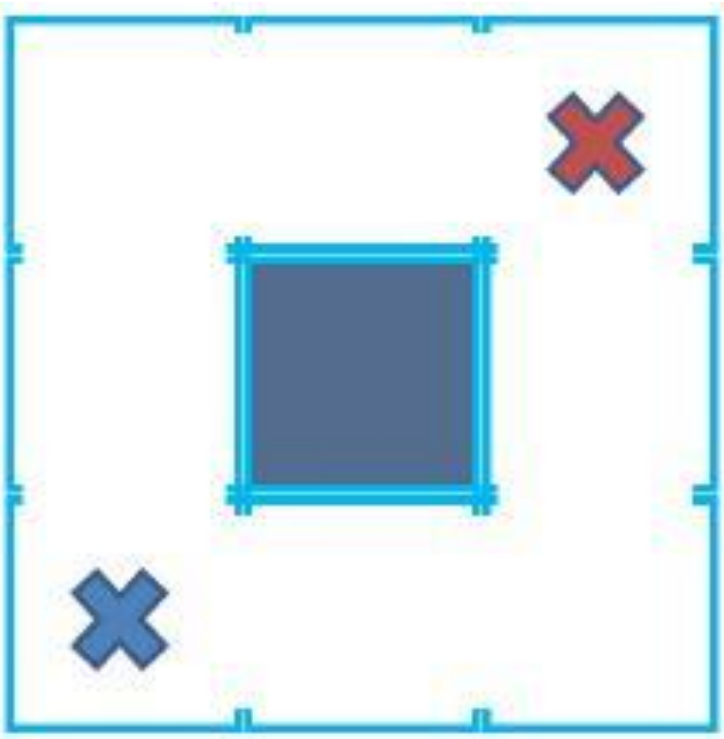


Figura 3.1. Escenario del robot

En este escenario, el robot parte de la esquina inferior izquierda, mirando hacia la esquina superior izquierda y debe llegar a la esquina superior derecha. La posición centro no es visitable por el robot. El tamaño de las celdas es de 30x30 cm.

Este escenario y los anteriores requisitos nos obligan a resolver los siguientes problemas:

1. Debemos crear un **algoritmo de control** del robot en **configuración de péndulo invertido** lo suficientemente robusto como para asegurarnos de que no se va a caer fácilmente., y que se mantenga estable durante su movimiento.

2. Además de controlarse a sí mismo como péndulo invertido (es decir, mantener la verticalidad), debe **controlar su posición** en el eje de actuación. Por tanto, después de moverse hacia adelante/atrás para que el momento de fuerza no haga que caiga; debe desplazarse para volver al

mismo punto en el que se encontraba antes de la corrección, sin perder con esto la estabilidad.

3. Debemos poder **controlar el movimiento del robot dentro del laberinto**, para llevarlo de un estado a otro. Esto significa que debemos poder controlar las ruedas de forma independiente para hacer giros, y avanzar y retroceder, pero sin interferir en el control de la estabilidad. Además el robot no debe poder chocar contra las paredes, ya que seguramente caería.

4. El robot debe poder saber **dónde se encuentra** en todo momento, para poder informar al algoritmo de control de **posición en escenario**; o al menos implementar algún mecanismo de adquisición de datos del entorno que permita a los algoritmos de control de posición saber dónde está, y hacia donde está mirando.

5. Debemos implementar un **algoritmo de aprendizaje por refuerzo**, que mediante la interacción con el escenario 3x3, aprenda la sucesión de acciones que debe ejecutar para salir del laberinto, y que además, esto se haga en el menor número de movimientos. Debemos además ser capaces de darle premisas de cuál es la acción que nosotros consideramos más útil en la posición en la que actualmente se encuentra, implementando un sistema de **aprendizaje por confirmación,** mediante el cual podamos decidir cuál de los dos caminos posibles queremos que recorra nuestro robot (subiendo, y luego desplazándose a la derecha; o desplazándose a la derecha y luego subiendo). Sobre todo, esta es la tarea primordial que perseguimos con el proyecto, y será el objeto de nuestras pruebas en la sección dedicada a ello.

6. Establecer un **nexo entre todos los componentes** del sistema, de tal forma que aquellos sistemas que lo requieren sean capaces de obtener información de los otros sistemas de los que dependen.

Con todos estos requisitos, nosotros vamos a pasar ahora a exponer nuestra solución al problema planteado, junto a las distintas soluciones barajadas durante la fase de análisis.

3.2 Especificaciones del Sistema

Partimos de la necesidad de diseñar unos algoritmos de control, supervisión y adquisición de datos.

Vamos a exponer algunos problemas que tenemos de base antes de hacer un análisis exhaustivo:

1. Podemos prever, sin miedo a equivocarnos, que el procesador del NXT no va a ser capaz de soportar todos los algoritmos que necesitamos, ya que los algoritmos de control de la estabilidad y de control de la posición deben correr en tiempo real. Cuanto más tiempo pasen dichos controladores sin actuar, mayor será la desviación que obtengamos, lo cual puede llevar a comportamientos erráticos. Por tanto debemos buscar una forma de distribuir la ejecución de los algoritmos, ejecutando una parte en el propio NXT, y el resto en otro sistema.

2. Los algoritmos necesarios para el aprendizaje por refuerzo no son fácilmente implementables para hacerlos correr de forma nativa en el sistema NXT. Pero, como se comentaba en el estado del arte, estos algoritmos se ejecutan en tiempo discreto, y por lo que no requieren de una ejecución cercana al tiempo real, por lo que estos algoritmos son entonces candidatos perfectos a ejecutarse de forma externa al NXT.

Tras este pequeño análisis, vamos a ver cómo vamos a resolver cada uno de los puntos a resolver que se expusieron en el análisis de los requisitos.

3.2.1 Control de estabilidad: Anyway

Este es uno de las principales tareas a las que nos enfrentábamos al desarrollar el proyecto. Necesitábamos una solución que permitiera controlar la estabilidad del robot, desde el punto de vista de comportarse como un péndulo invertido, y que además evitara que el robot se desplazara en su actuación para corregir el error.

Tras una pequeña investigación se pudo ver que el problema del que hablamos está ampliamente resuelto por una gran cantidad de autores [NXTTWO], [NXTARDU]. Recientemente han surgido algunas soluciones basadas en Matlab, que mediante el uso de Simulink, permiten crear modelos que corran de forma nativa, en tiempo real, en el bloque NXT [MATLNXT].

Esto, como ya hemos visto, no es deseable por nosotros, que únicamente queremos correr en tiempo real parte de los algoritmos, pudiendo hacer el resto del control de forma externa.

Tras un pequeño análisis de las distintas soluciones para el control de la estabilidad y de la posición en Matlab, se vio que lo mejor era buscar la forma de hacer este control de forma nativa en el propio NXT, lo cual nos lleva al robot Anyway.

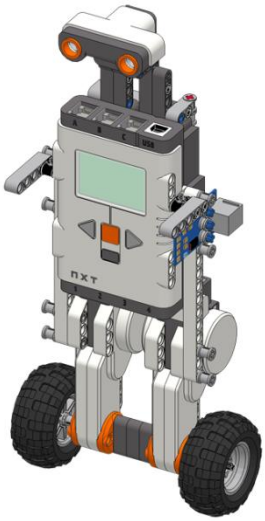


Figura 3.2. Vista de frente del AnyWay.

El robot Anyway [ANYWAYW] es un modelo de robot basado en NXT, montado en configuración de péndulo invertido, desarrollado por Laurens Valk. Este es actualmente un tipo de robot utilizado en los ámbitos académicos y de investigación [CEAANY] [EMBEDCONT].

Lo que convierte al AnyWay en un modelo tan útil es el hecho de utilizar para el control de la estabilidad y la posición un control basado en un controlador PID discreto, como el expuesto en la sección dedicada del estado del arte, pero implementado utilizando NXT-G. Además, el autor posibilita la realización de giros, y avanzar y retroceder, pero sin realizar un control exhaustivo de la posición final en la que queda el robot, lo cual puede darnos problemas en la fase de implementación.

Para su montaje necesitamos de un Sensor HiTechnic Gyroscopic, montado de tal forma que nos de la velocidad de caída. Cuando la velocidad de caída sea 0, entenderemos que no estamos cayendo, por lo que estamos en el punto de equilibrio del robot.

Mediante el sintonizado del controlador PID, podemos ajustar el controlador a cualquier modelo de robot NXT en configuración de péndulo invertido, y además, corriendo dicho controlador en código directamente ejecutable mediante el procesador del bloque NXT.

Haciendo algunas modificaciones sobre el código original, podemos controlar la posición y la estabilidad de forma fiable; dejando la puerta abierta a realizar las modificaciones sobre el código original que nos permita el control del movimiento del robot.

3.2.2 Control de Movimiento y Posición en entorno

Para controlar el movimiento dentro del escenario del robot AnyWay, necesitamos saber la orientación del robot, y en cuál de las 8 posiciones posibles (al ser la central no visitable) se encuentra. De esta forma podremos transitar de un estado a otro cuando lo necesitemos, sabiendo tras realizar la transición, a que estado hemos llegado.

Resolveremos el problema de la orientación realizando giros de 90º, los cuales nos harán cambiar nuestra orientación en el espacio. Estableceremos el siguiente diagrama de puntos cardinales:

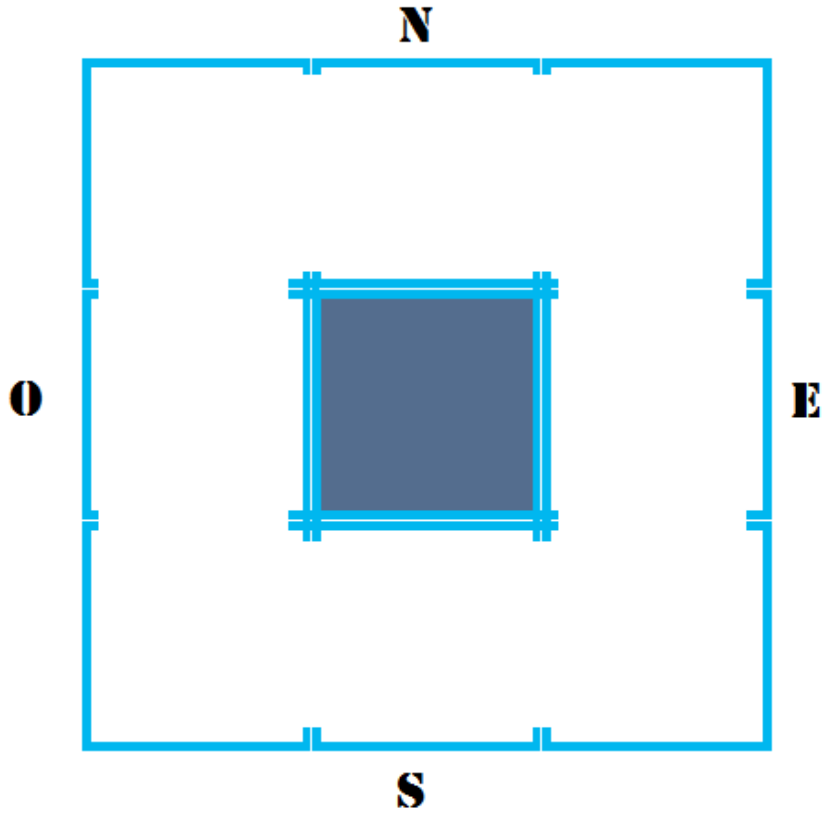


Figura 3.3. Puntos cardinales respecto al escenario

Así, para ir de un estado a otro giraremos en el sentido que resulte en un número menor de giros, luego avanzaremos hasta llegar al centro del siguiente estado. La forma de controlar cuanto hemos avanzado realmente es mediante el uso del sensor de ultrasonidos del que dispone el AnyWay. Por las características del escenario, el rango en el que opera el sensor de ultrasonidos es suficiente para poder ver la pared que tenga enfrente en cualquier caso. No obstante, por las características del sensor, no podemos fiarnos completamente de los datos que nos da el sensor, al ser bastante común que dé errores en la lectura de la distancia.

Por tanto, mediante estos algoritmos podremos girar y avanzar de forma fiable, sabiendo en qué estado me encuentro al saber de qué estado he venido

3.2.3 Nexo entre componentes: RWTH y buzones NXT

Con todo lo que hemos comentado, ya hemos expuesto la solución a los problemas de control del robot (a excepción de los algoritmos de aprendizaje, de los que ahora hablaremos).

Tanto el control de movimiento como el de control de posición deben hacerse de forma externa al robot NXT, ya que queremos que únicamente corran en el propio sistema aquellos algoritmos que requieran de un tiempo de respuesta lo más pequeño posible, por ser sistema de tiempo real.

Por ello, debemos encontrar una forma de ejecutar los algoritmos de control de estabilidad en el NXT; y el resto de algoritmos en otro equipo, que en nuestro caso va a ser un ordenador; todo de forma simultánea y permitiendo la comunicación entre sistemas.

La forma de conseguirlo pasa por hacer uso de las funciones RWTH, de las que ya habíamos hablado en el estado del arte, las cuales nos van a permitir correr el programa de control de estabilidad y posición, y mientras, poder obtener los datos del sensor de ultrasonidos. Si bien tenemos el problema de que, para realizar el control del movimiento del robot, necesitamos poder acceder a las variables locales del algoritmo nativo NXT-G, las cuales no son accesibles desde fuera del programa mediante la comunicación que establecen las funciones RWTH.

La solución propuesta es hacer uso de los buzones Bluetooth [NXTMAIL] que utilizan los NXT para intercambiar mensajes con otros sistemas NXT. Dichos buzones están en un espacio de memoria al que podemos acceder mediante las funciones RWTH, y lo que haremos será hacer que el programa de control de estabilidad y posición vigile el buzón, y realice la acción que tenga escrita en el mensaje que le llegue. Cuando queramos que el sistema realice una acción (Moverse hacia adelante, Moverse hacia atrás, girar 90º horarios, girar 90º anti-horarios), pondremos un mensaje en dicho buzón, el cual será extraído por el programa NXT-G para, mientras hace el control de estabilidad y posición, hacer la acción de movimiento que le hemos pedido.

De esta forma lo que haremos es implementar el control de movimiento y el control de posición en el NXT, y mantener todos los elementos del sistema de aprendizaje por confirmación (basado en aprendizaje por refuerzo), en Matlab; y utilizaremos las funciones RWTH para la comunicación con el control de posición y estabilidad mediante el uso de los buzones del sistema NXT.

3.2.4 Aprendizaje por Confirmación

Para implementar el aprendizaje por confirmación, podríamos haber optado por distintas alternativas, que tienen como base técnicas de control inteligente. Inicialmente se propuso hacer el aprendizaje supervisado mediante el uso de redes neuronales de tipo perceptron multicapa con aprendizaje por retro-propagación; y Q-Learning o SARSA como algoritmo de aprendizaje por refuerzo.

El principal problema era establecer un nexo entre ambos aprendizajes, por ser tan diferentes en su implementación. Tras una pequeña tarea de estudio del campo, hemos optado por utilizar otra estrategia para conseguir el aprendizaje por confirmación.

La estrategia seguida pasa por modificar los valores de Q(S, A), de tal forma que podamos influir en el comportamiento del robot, pero que tome nuestras indicaciones, más que como ordenes, como consejos de las acciones óptimas. Vamos a ver que somos capaces de influir sobre la ruta que toma el robot tras varios episodios de entrenamiento, suavizando con el tiempo nuestras preferencias. También vamos a poder ver que el robot es capaz ignorar tras varios episodios, las indicaciones dadas cuando estas no son correctas, o no son del todo correctas, por existir otra acción mejor que la dada por nosotros.

Si es un **episodio de exploración**

Selecciono acción con Q(S,A) mayor (*Aprendizaje por refuerzo)*

Si es un **episodio de entrenamiento**

Pido acción al instructor *(Aprendizaje por Confirmación)*

Ejecuto acción elegida por el paso anterior

Si es un **episodio de entrenamiento**

Rebajo el peso Q(S,A) de las acciones no elegidas

Potencio el peso de la acción realizada

Algoritmo 2: Actuación en función del tipo de episodio

Capitulo 4: Desarrollo

En este capítulo expondremos como se han implementado los algoritmos expuestos en el capítulo de análisis.

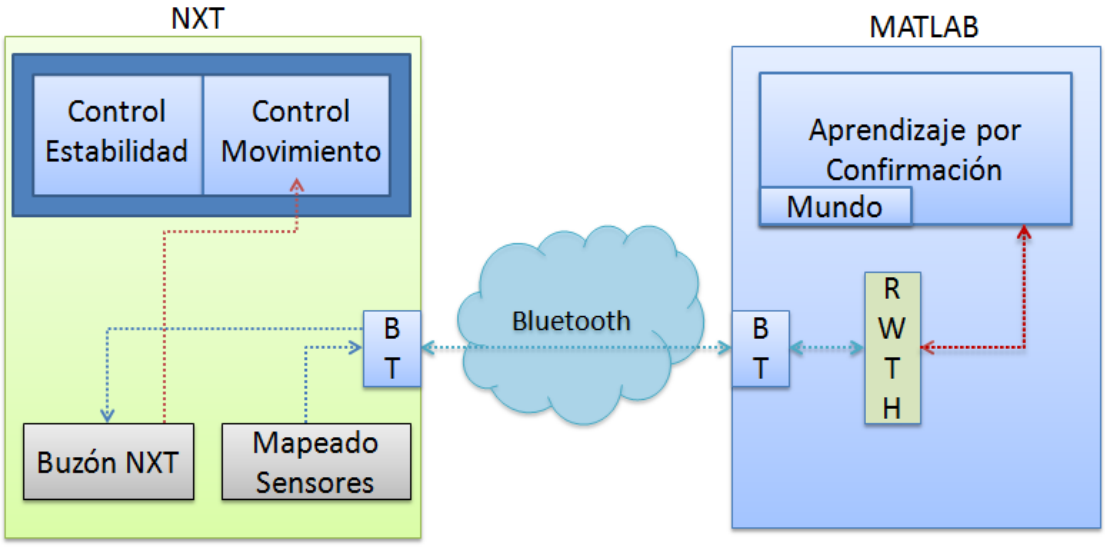


Figura 4.1. Diagrama esquemático del sistema

El **control de estabilidad** se ha implementado mediante la cooperación de dos controladores PID, uno destinado a controlar la estabilidad, usando como variable de proceso la velocidad angular de caída, y otro destinado a controlar el desplazamiento necesario para corregir la caída, que usa como variable de proceso el desplazamiento medido por los encoder rotacionales de los servomotores. Para el **intercambio de información** entre sistemas se han hecho uso de las funciones RWTH, de las que hemos hablado en el estado del arte. Para definir el **sistema de aprendizaje por confirmación** hemos implementado un algoritmo de aprendizaje por refuerzo basado en SARSA, al cual hemos hecho unas modificaciones para poder inferir nuestro conocimiento del proceso, y hacerlo aprovechable por nuestro algoritmo. Por último, los **algoritmos de control de movimiento y**

posición en el entorno están distribuidos entre Matlab y el bloque NXT, encargándose el segundo únicamente del control a bajo nivel del movimiento.

4.1 Control de Estabilidad: Anyway

Nuestra primera tarea será montar el robot Anyway; que será la planta sobre la que efectuaremos nuestras labores de control. Por sencillez, y por ser la configuración que usualmente se utiliza con el controlador del robot AnyWay, montaremos el robot en la configuración de péndulo invertido, con la distribución de sensores para el que ha sido ajustado el controlador PID. Las instrucciones de montaje pueden consultarse en el sitio web de RobotSquare [ANYWAYW] . De entre todas las opciones de montaje, hemos montado el robot AnyWay que tiene las ruedas del kit Lego MindStorms 2.0 (que son de menor radio, pero más anchas). Además, como ya hemos comentado anteriormente, nuestro robot tiene un sensor de ultrasonidos, y el gyrosensor. Tras el montaje, el robot tiene el siguiente aspecto:

Figura 4.2. Fotografía de Frente del Anyway

Una vez tenemos la planta montada, siguiendo las instrucciones de montaje, procedemos a implementar el controlador dentro del bloque NXT.

El programa debe ser cargado desde la aplicación NXT-G, de la ya hemos hablando anteriormente. El código tiene la siguiente distribución:

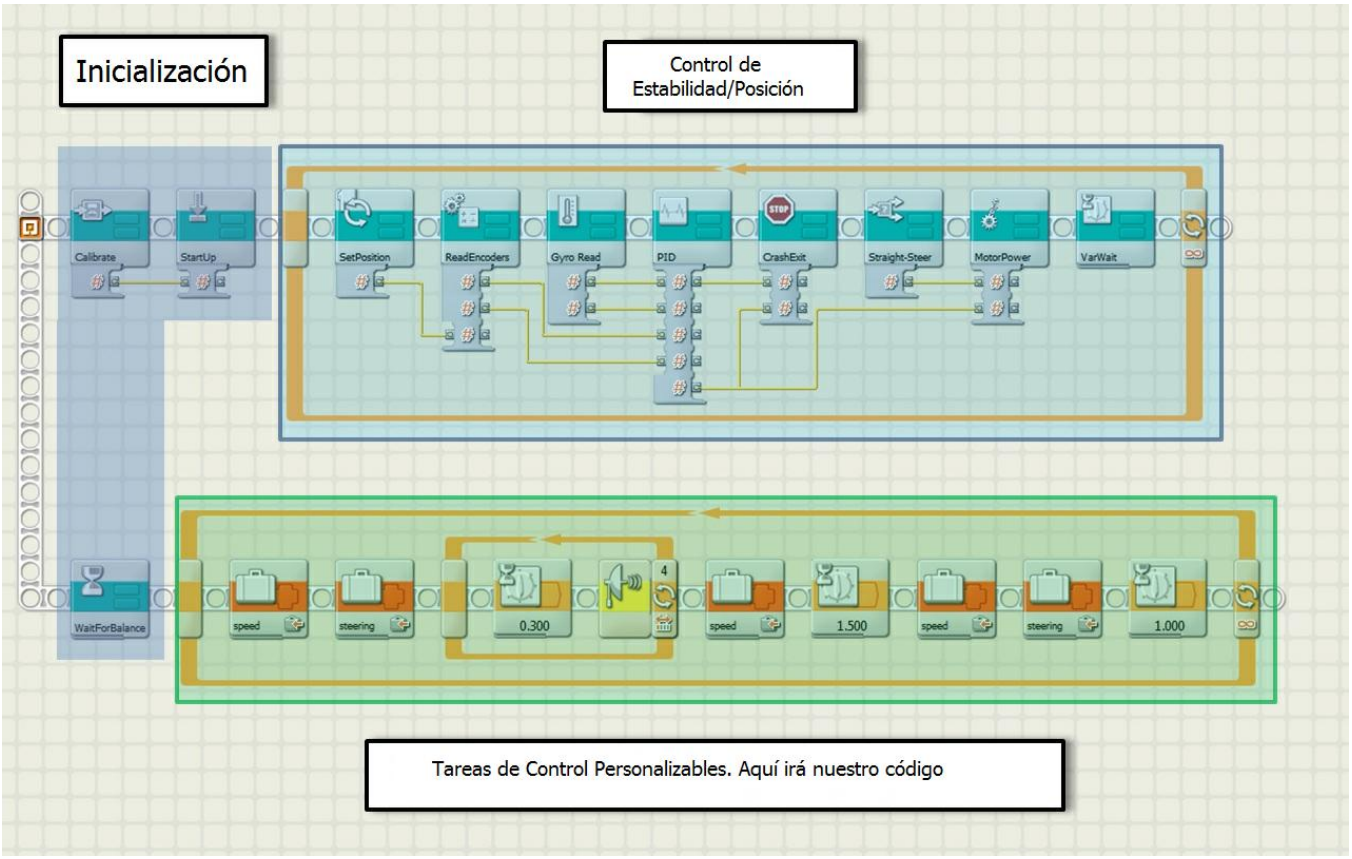


Figura 4.3. Esquema del código NXT-G

Como vemos, el código tiene 3 secciones de código:

- La primera sección es la de I**nicialización**. En ella, se hacen las tareas previas necesarias para la ejecución de los algoritmos de control.

- La segunda sección es la **de Control de Estabilidad y Posición**. En ella, el algoritmo de control PID se encargará de realizar las acciones necesarias para que nuestro robot mantenga la estabilidad; y además, de realizar una actuación que recupere el robot a su posición inicial ante

perturbaciones. Este controlador tiene como núcleo un controlador PID discreto.

- Por último, la tercera sección es donde colocaremos el **código de personalización**, es decir, los algoritmos que queramos que nuestro robot ejecute en los tiempos muertos del algoritmo de control de estabilidad y posición. Será aquí donde más adelante implementaremos el algoritmo de monitorización de los buzones NXT, para el intercambio de mensajes entre el NXT y matlab.

Por el hecho de usar la planta que nos propone el autor, el controlador PID está ajustado a la planta de forma contrastada. En el caso de usar una planta ligeramente diferente, a priori el controlador PID debería ser capaz de realizar un control de la estabilidad suficientemente bueno como para evitar que caiga.

Hemos optado por utilizar la planta que nos propone el autor ya que, como comentaba, este controlador ha sido ampliamente probado y depurado, y realiza un control de la estabilidad sin casi desplazamientos en eje de actuación. Esto es deseable para nuestro problema, ya que nuestro robot no tiene demasiado espacio para corregir, y un exceso de actuación se traduciría en un gran desplazamiento, lo que podría llegar a provocar un choque con alguna de las paredes, lo que provocaría que nuestro robot caería al suelo.

Tras el montaje, al ejecutar el algoritmo podemos ver un control de la estabilidad y de la posición suficientemente bueno como para poder dar por concluida esta parte. Ahora hablaremos de la forma en la que intercambiaremos información entre el robot NXT, y nuestro equipo sobre el que estaremos corriendo Matlab.

4.2 Intercambio de Información Matlab-NXT: RWTH

Una vez determinado y comprobado el control de la estabilidad y la posición. Buscaremos la forma de poder comunicarnos con el robot NXT. La solución implementada hace uso de las funciones RWTH de Matlab.

Lo que haremos será:

1. Estableceremos una comunicación por Bluetooth o USB con el robot NXT.

2. Una vez la comunicación esté establecida, mandaremos al robot NXT ejecutar el algoritmo de control de estabilidad y posición (control.rxe).

3. Dicho algoritmo se dedicará a monitorizar uno de los buzones NXT, donde esperará mensajes.
    a. Cuando llegue un mensaje con el texto “Adelante”, ejecutará el código de avanzar
    b. Cuando llegue un mensaje con el texto “Atras”, ejecutará el código de retroceder
    c. Cuando llegue un mensaje con el texto “Giro90”, ejecutará el código girar en sentido Horario
    d. Cuando llegue un mensaje con el texto “Giro-90”, ejecutará el código de girar en sentido Anti Horario.

4. Cuando queramos que el robot ejecute una acción, desde Matlab mandaremos un mensaje al buzón objetivo a través de Bluetooth o USB con el texto de la acción que queremos que realice nuestro robot.

5. Cuando queramos obtener la distancia que está midiendo el sensor de ultrasonidos, ejecutaremos el algoritmo de lectura; el cual utiliza la conexión establecida para leer el valor directamente del propio robot NXT.

Ahora vamos a pasar a comentar como se ha implementado lo comentado. En primer lugar, hablaremos del código personalizado introducido al robot AnyWay.

La forma de implementar el código de las 4 acciones (Adelante, Atrás, Giro de 90º horario y Giro de 90º Anti Horario) se ha hecho mediante un bloque Switch anidado en un bloque Case. Cuando llega un nuevo mensaje a dicho buzón, se extrae el mensaje y la instrucción Case ejecuta la acción correspondiente.

La acción de **moverse hacia adelante** realiza un pequeño desplazamiento hacia adelante. Lo que haremos en nuestro control del movimiento en el seno del escenario será comprobar la distancia hasta el muro que tiene enfrente. De esta forma sabremos cuanto debemos movernos para haber hecho la transición de estado; haciendo el envío de tantos mensajes de movernos hacia adelante como nos sea necesario.

La acción de **moverse hacia atrás** tiene un funcionamiento análogo al de moverse hacia adelante.

Por las características del robot (Modelo diferencial) podemos saber cuántos grados ha girado el robot, y cuanto se ha desplazado respecto a su eje central, sabiendo el valor del encoder de las ruedas, ya que la posición y el ángulo del robot pueden obtenerse teniendo como único dato el valor de los encoder del robot.

Por ello, las **acciones de girar 90º en sentido horario y la de girar 90º en sentido anti horario** utilizan los datos de los encoder rotacionales de cada uno de los motores (solidario a cada una de las ruedas) para saber cuanto ha girado el robot. Lo que se hace es obtener el valor del encoder antes de empezar a realizar el giro, para a continuación empezar a girar. Cuando el valor del encoder llegue al valor correspondiente a un giro de 90º, dejaremos de girar.

En Matlab, deberemos inicializar una comunicación con el bloque NXT, mediante el uso del código:

```
NXT3.MAC = '00165313177B';
NXT3.connectionMode = 'any';
NXT3.bluetoothIni = 'bluetooth.ini';
handle = COM_OpenNXTEx(NXT3.connectionMode,NXT3.MAC,NXT3.bluetoothIni);
```

La función COM_OpenNXTEx, mediante el uso del parámetro ‘connectionMode’ puesto en Any, intentará en primer lugar una comunicación Bluetooth. En el caso de fallar, intentará realizar la comunicación mediante el uso del cable USB.

Una vez la comunicación sea exitosa, tendremos un Handle de la comunicación actual. Mandaremos que ejecute el código NXT-G mediante el uso del la función NXT_StartProgram, dándole como parámetro el Handle de la comunicación establecida:

```
NXT_StartProgram('Control.rxe',handle);
```

Una vez dicho código se esté ejecutando, podremos mandar mensajes a un buzón NXT mediante el uso de la función NXT_MessageWrite:

```
NXT_MessageWrite(“Texto”, 1, handle);
```

Por último, cuando deseemos leer el valor de la distancia medida por el sensor de ultrasonidos, podemos realizarlo ejecutando el siguiente código:

```
OpenUltrasonic(SENSOR_4,'snapshot',handle);
distancia = GetUltrasonic(SENSOR_4,handle);
CloseSensor(SENSOR_4,handle);
```

Donde distancia será tipo double con la lectura instantánea de distancia en cm.

Cuando hablemos de las funciones de aprendizaje por refuerzo, expandiremos este punto.

4.3 Aprendizaje por Confirmación

Con todo lo expuesto, ya somos capaces de establecer una comunicación bidireccional entre el robot NXT y Matlab, pudiendo darle ordenes de movimiento al robot, mientras mantiene su posición y estabilidad. Ahora vamos a hablar acerca del algoritmo de aprendizaje por confirmación.

En primer lugar, hay que tener en cuenta que la forma de conseguir el aprendizaje por confirmación consiste en usar un algoritmo de aprendizaje por refuerzo, al que vamos modificando los valores de su función de recompensa para conseguir, con esta estrategia, poder influir en su comportamiento proporcionando ejemplos al robot. Por ello, a lo largo del capítulo (y del resto de capítulos) es muy común que hablemos indistintamente de aprendizaje por refuerzo y de aprendizaje por confirmación.

Los algoritmos de aprendizaje por confirmación se han implementado en Matlab. El sistema almacena la posición y orientación del robot, y en base a ello, decide la secuencia de acciones que debe ejecutar el robot para hacer la transición de un estado a otro, haciendo uso de las funciones RWTH. Estas secuencias de acciones y su implementación se trataran en la sección siguiente, asumiendo en este punto que somos capaces de hacer dichas acciones.

En primer lugar, hemos creado **2 modos de funcionamiento** del algoritmo de aprendizaje: uno **real**, donde debemos establecer la comunicación con el robot, y ejecutaremos las acciones que nos ordene nuestra política; y otro **simulado**, donde utilizamos la representación del mundo del algoritmo de aprendizaje por refuerzo, y asumimos que las acciones de movimiento y giro se hacen de forma automática.

Para conseguir, en primer lugar, el aprendizaje por ejemplos hemos tenido que implementar de la siguiente forma los elementos que componen el sistema de aprendizaje por refuerzo:

- **El Modelo del mundo**: El mundo sobre el cual trabajara nuestro algoritmo de aprendizaje por refuerzo está definido por **9 estados**. En cada estado se permiten **cuatro acciones: Ir al Norte, al Sur, al Este y al Oeste.** Cuando el robot está en una posición en la que la acción que ha decidido realizar no es posible (por no haber ningún estado, o ser una pared), la transición que se produce es hacia el mismo estado (es decir, el robot no se movería). Por las características del estado central, ninguna acción provocará una transición real a ese estado, al ser un estado no visitable.

- **La política**: La política implementada seleccionará aquella acción de entre las cuatro posibles, que tenga un valor Q(S,A) mayor.

- **La función de recompensas:** La función de recompensas implementada asigna de forma estática a cada estado una recompensa. Las recompensas asignadas pueden verse en la figura inferior. El estado superior derecha será nuestra meta, y será un estado terminal.

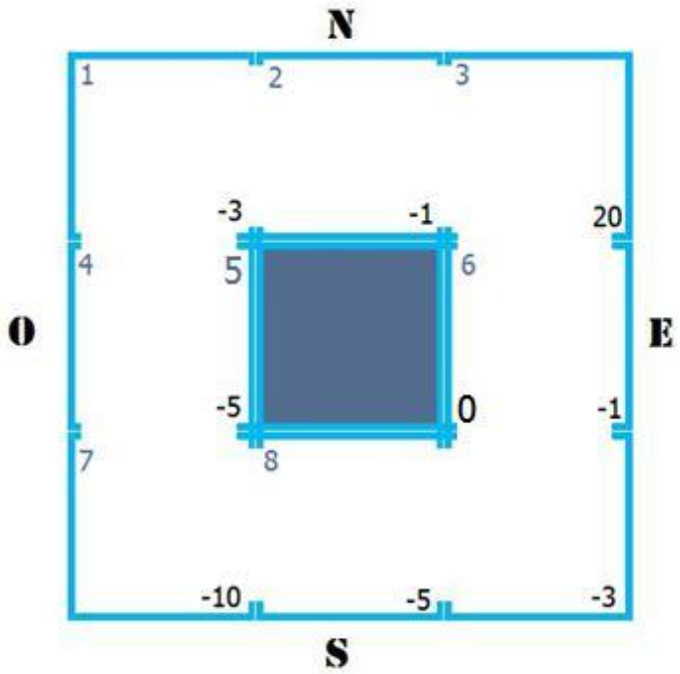


Figura 4.4. Recompensas asociadas a cada estado (negro), y numero de estado (azul).

- **La función de Valor:** La función de valor utilizada deriva del algoritmo SARSA (el cual ya hemos comentado en el estado del arte). Inicialmente, todos los pares Q(S,A) están inicializados a 0.

Un detalle interesante a hacer notar es el hecho de tener recompensas negativas en los estados intermedios. Esto debemos hacerlo al tener posibles transiciones de un estado a sí mismo, ya que nuestro agente podría llegar a quedarse en dicho estado recibiendo recompensas, al haber aprendido que la transición a ese estado, le proporciona una recompensa muy grande a largo plazo. Las recompensas que hemos elegido buscan llevar al agente hasta la meta con un menor tiempo de exploración. En el capítulo de pruebas haremos un experimento para demostrar que el robot es capaz de aprender la ruta incluso con una configuración de recompensas que no aporten información extra del entorno.

Ahora hablaremos de cómo se ha implementado el aprendizaje por confirmación sobre el algoritmo de aprendizaje por refuerzo. En primer lugar, antes de empezar 1 episodio (es decir, una sesión de movimiento del robot desde su posición actual al estado terminal), podemos elegir entre dos opciones:

- **Episodio de Exploración**: Este modo de funcionamiento ejecuta el algoritmo de aprendizaje por refuerzo tal y como está definido. Es decir, el robot explorará el escenario, eligiendo la siguiente acción en función de su política (la cual, como ya hemos dicho, depende de los valores de su función de valor Q(S,A)). Por ello, los valores de los Q(S,A) se irán actualizando con su exploración/explotación.

- **Episodio de Entrenamiento**: Este modo de funcionamiento nos pedirá en todo momento cual es la acción que desde nuestra posición privilegiada, creemos que el robot debería ejecutar. El robot, una vez le hayamos dicho al algoritmo lo que creemos que debe hacer, ejecutará la acción; y desde su nuevo estado en el entorno, nos pedirá de nuevo nuestra opinión. Cuando le demos nuestro ejemplo de acción correcta para ese estado, el valor

Q(S,A) para dicho estado-acción se verán ligeramente incrementado, mientras que el resto de acciones para dicho estado se verán ligeramente penalizadas.

De esta forma, utilizando los episodios de Entrenamiento, somos capaces de enseñar al robot, mediante ejemplos (tal y como haríamos en el caso de un aprendizaje supervisado) cuales son las acciones que consideramos mejor. En el caso de estar equivocados (o de directamente estar engañándole con nuestros ejemplos), el robot será capaz de, con su exploración (y su actualización de los valores de Q(S,A durante la misma), deshacer lo que le hemos enseñado.

A priori se pueden intercalar episodios de exploración con episodios de entrenamiento. Aunque, los mejores resultados se obtienen cuando ejecutamos inicialmente varios episodios de entrenamiento, como ya veremos en la parte de pruebas.

La forma de modificar los valores de Q(S,A) de tal forma que podamos influir en su comportamiento, sin sesgar el aprendizaje a sido unas de las tareas más complejas del presente proyecto. Inicialmente se pensó en variar únicamente el Q(S,A) del estado en que me encuentro, reduciendo el resto de acciones, y potenciando la acción que he realizado. Pero el algoritmo SARSA utiliza para decidir la siguiente acción el valor Q(S,A) del siguiente estado. Por ello se ha optado por reducir los valores Q(S,A) de los estados vecinos al estado en el que me encontraba antes de realizar la transición. De esta forma, potenciamos dicho camino, sin sesgar el aprendizaje futuro del aprendizaje por refuerzo.

```
%Potenciamos la acción que le hemos dicho que es correcta, y rebajamos el
resto.
for t=1:(size(actionlist,2))
      Q(sp,t)=(Q(sp,t)-2);
      if(Q(sp,t)<=-10) Q(sp,t)=-10;end
end

Q(sp,ap)=Q(sp,ap)+3;
if(Q(sp,t)>=30) Q(sp,t)=30;end
```

Algoritmo 3: Ejemplo de función de potenciación

4.4 Control de Movimiento y Posición en el Entorno

Vamos a hablar del último punto desarrollado: el control del robot en el seno del escenario donde habita el agente.

Por las características de todo lo que hemos desarrollado anteriormente, la forma de hacer este control se ha basado en implementar una serie de funciones que al llamarlas, envíen una secuencia de órdenes al robot a través de las funciones RWTH. Cuando el algoritmo de aprendizaje por refuerzo necesita hacer una acción, llama a estas funciones, las cuales se encargarán de realizar las acciones realmente en el robot. En el caso de estar ejecutando el algoritmo de forma simulada (es decir, sin robot), el algoritmo de aprendizaje por refuerzo omitirá llamar a estas funciones.

En esencia, tendríamos 4 funciones: la función giro_mas_90, la función giro_menos_90, la función adelante, y la función atrás. Cada una de estas funciones enviará un mensaje al buzón del robot NXT. Un ejemplo de dicho código puede verse en el Algoritmo 4. En el caso de adelante, la función enviará varios mensajes de adelante o de atrás en función de la distancia hasta la pared, parando de enviar mensajes cuando la distancia sea la correcta.

```
while(orientacion~=action)
%Mientras no este mirando en la direccion correcta, girare.
%-------------------------------------------------------------
      if(orientacion==1) %Estoy mirando hacia el norte
            if(action==2 || action==3)
                  if(real==1)
                  %Giro -90
                  NXT_MessageWrite('Giro-90', 1, handle);
                  pause(5)
                  end
            orientacion=orientacion+1;
            end
```

Algoritmo 4: Ejemplo de función de desplazamiento

Para lograr este cálculo, el algoritmo determinará cuándo debe moverse en función de la posición en la que inicialmente está. Luego calculará el error que actualmente tiene el robot en su posición. Esto lo podemos lograr al saber exactamente como de grande es cada estado (siendo cuadrados de lado x lado).

El centro de cualquier estado estará a $(N * Lado) + (Lado/2)$, es decir, N lados completos, más la mitad de uno de esos lados. Restaremos el número de lados completos, y restaremos también Lado/2. El resultado nos dirá como de lejos estamos, y hacia qué lado. De esta forma, a la posición final de nuestro robot le tendremos que restar o sumar ese pequeño error. Dicho código puede verse en el Algoritmo 5.

```
dist_inicial=(d_ini+d_ini2)/2;
pause(5);
%Con esto obtengo cuando de lejos estoy de donde debería estar.
%En primer lugar, quito los estados completos que tengo por delante
%En segundo lugar, donde debería estar (el centro está a lado/2)
error=mod(dist_inicial,lado)-(lado/2);
%Un error >0, significa que debería moverme ese poco más.
%Un error <0, significa que debería moverme ese poco menos.
```

Algoritmo 5: Cálculo del error

Capitulo 5: Experimentos y Resultados

Tras el análisis del problema y su posterior desarrollo vamos a hacer las pruebas funcionales del sistema diseñado. En primer lugar vamos a presentar las diferentes pruebas que vamos a realizar.

5.1 Descripción de Experimentos

Vamos a pasar a exponer cada uno de los experimentos y pruebas que realizaremos para probar los distintos controladores del robot:

- **Prueba de Estabilidad y Control de Posición**: En esta prueba pondremos el robot en vertical (es decir, en su posición de equilibrio) y empezaremos a empujarlo en repetidas ocasiones. Tras estas perturbaciones, comprobaremos como de lejos queda de la posición inicial. El robot debería no desviarse más de un par de centímetros, y sobre todo, ser capaz de mantener la estabilidad.

- **Prueba de Control de Navegación:** Con esta prueba comprobaremos una parte del correcto funcionamiento de la comunicación bidireccional; y del control de movimiento y posición por el entorno. Haremos al robot transitar de un estado a otro para comprobar como de lejos se queda el robot del centro del siguiente estado, probando varios errores iniciales respecto a la celda inicial.

- **Pruebas de aceleración del aprendizaje mediante aprendizaje por confirmación:** Esta última serie de pruebas busca demostrar que hemos sido capaces de acelerar el aprendizaje del robot mediante el uso de una serie de Episodios de entrenamiento previos a permitir al robot empezar a explorar; siendo capaces de darle nuestras preferencias sobre el camino que consideramos que debería tomar de entre los posibles.

5.2 Resultados y Discusión de Experimentos

Vamos a ir exponiendo los diferentes experimentos realizados, y los resultados obtenidos en los mismos.

5.2.1 Prueba de Estabilidad y Control de Posición

Este primer experimento busca encontrar la desviación en el eje de actuación del robot para no perder el equilibrio, cuando es empujado en repetidas ocasiones. Este escenario es mucho peor que el que se le podría dar al robot dentro del escenario, donde el robot no será, a priori, molestado por ningún agente externo. Por ello, si esta prueba resulta satisfactoria podremos concluir que el robot será capaz de controlar su estabilidad y posición tras actuar para corregir su caída.

Para las pruebas se ha usado la manta que acompaña al kit lego Mindstorms 2.0, la cual trae una zona escalada en centímetros para este tipo de pruebas. La prueba realizada pasa por empujar al robot con el dedo cada vez que el robot ha vuelto a la posición inicial (es decir, la que el robot considera que es su posición inicial). Cada vez empujamos el robot con mayor fuerza, siempre empujándole con el dedo sobre la pantalla del bloque NXT.

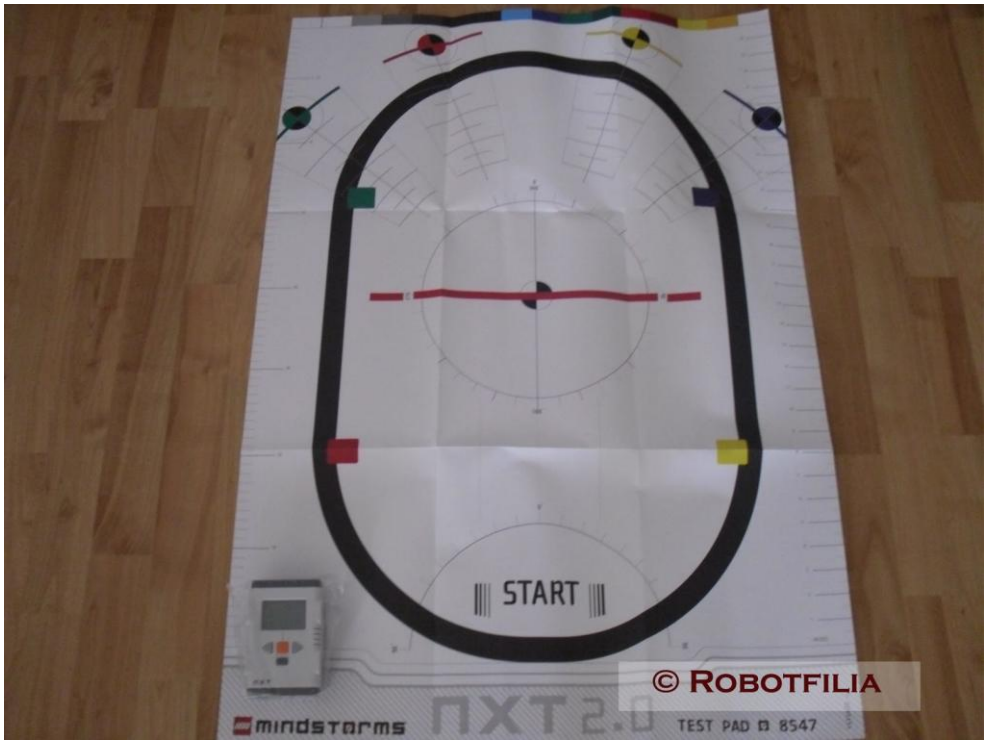


Figura 5.1. Sabana de pruebas

Los resultados obtenidos, tras 2 minutos de empujar al robot, han sido de 2,5 cm hacia atrás. Realmente es un desplazamiento muy pequeño, casi despreciable, teniendo en cuenta el error que introduce el sensor de ultrasonidos (recordemos que tenemos una incertidumbre en el peor de los casos de ±3cm sólo por usar este sensor), y que posteriormente nos provocará errores de un orden similar. También debemos recordar que el algoritmo que controla el desplazamiento al cambiar de un estado a otro debe ser capaz de corregir los pequeños errores en la posición (tanto los reales, como en este caso, como los introducidos al medir mal la distancia el sensor de ultrasonidos).

Por todo esto, este resultado es más que satisfactorio. Se ha conseguido empujar al robot con bastante fuerza, sin que cayera al ser capaz de corregir la desviación, pero además sin perder su posición inicial. De esta prueba también se desprende que el robot, tras estar un tiempo sin recibir ninguna orden de movimiento, es capaz de mantener su posición en el espacio.

5.2.2 Prueba de Control de Navegación

Esta prueba busca demostrar que funciona correctamente la comunicación bidireccional, y el control de movimiento y posición por el entorno. La forma de probarlo ha sido llevar el robot desde su posición inicial, a la posición final; usando un episodio de entrenamiento.

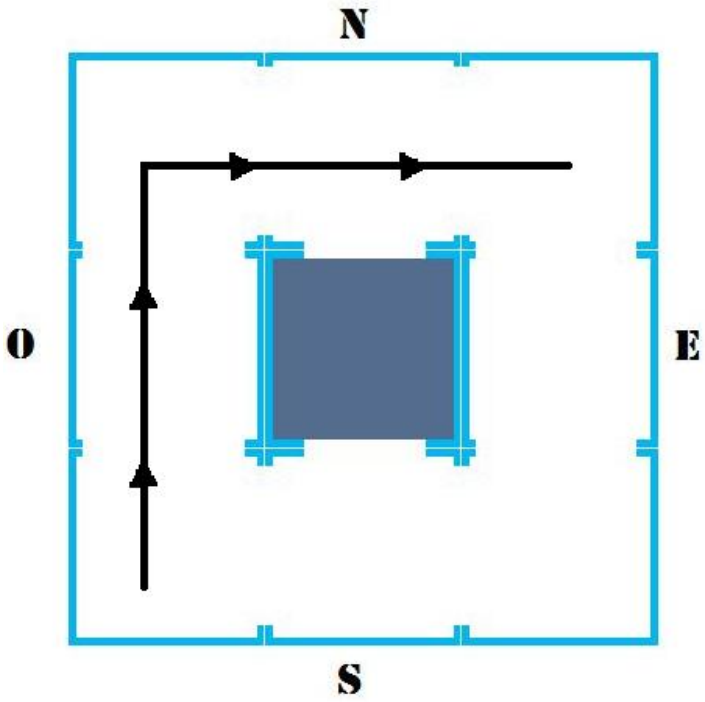


Figura 5.2. Camino recorrido en prueba de Navegación

Por tanto, desde el interfaz de Matlab, le diremos al robot que debe ejecutar las acciones NORTE1-NORTE2-ESTE1-ESTE2. El lado de los estados se ha fijado en 30cm, y el entorno se ha construido con pilas de libros (por sencillez).

Los resultados en cada acción son los siguientes:

- **Paso 1 (NORTE1):** En esta transición se corrige el error que hemos introducido al poner el robot más o menos en el centro. Además, notamos que el robot cuando empieza su ejecución tiende a irse ligeramente hacia atrás (alrededor de los 3cm). Cuando hace la transición, el robot queda a 2 cm del centro del siguiente estado.

- **Paso 2 (NORTE2)**: En esta transición observamos que se ha quedado a 3 cm del centro del estado.

- **Paso 3 (ESTE1)**: Este estado implica un giro de 90º. Al realizar el giro observamos que el mismo es lo suficientemente preciso. Si bien, también hemos observado que si la batería comienza a descargarse, los giros pueden no ser del todo precisos, lo cual afectará a nuestra navegación. Tras esto, el robot se mueve hacia adelante, quedándose a 4 centímetros del centro del estado.

- **Paso 4 (ESTE2)**: Una vez más, el robot se mueve hacia adelante, dejando un error final de 2 centímetros.

Reducir el error que cometemos al intentar ponernos en el centro del estado es importante, ya que un error acumulado podría hacernos creer que estamos en un estado, y no estar en el mismo realmente. Esto provocaría que el robot cayera.

5.2.2 Pruebas de Aceleración del Aprendizaje, mediante el uso de Aprendizaje por Confirmación.

Esta sección busca demostrar que hemos conseguido lo que nos proponíamos inicialmente con el proyecto; crear un sistema de aprendizaje por confirmación, que nos permita aprovecharnos de nuestro conocimiento del mundo donde está nuestro agente, para facilitarle su tarea de aprendizaje, y así, reducir el número de pasos y de episodios de exploración que requiere el agente para encontrar la solución al problema. En las tablas que encontraremos a continuación, marcaremos en verde los episodios en los que el robot ha seguido nuestras premisas, al ser un episodio de entrenamiento. El resto de episodios serán de exploración, marcando en rojo el primer episodio de exploración del robot que ya es la solución optima.

El **primer conjunto de pruebas** está destinado a demostrar que el robot es capaz de encontrar una solución en menor número de pasos/episodios de exploración, si previamente le hemos dado nosotros ejemplos de acciones correctas en cada estado. La ruta que pediremos que haga es la misma que en la prueba del control de navegación.

En el caso de que no hiciéramos **ningún episodio de entrenamiento**, obtendríamos el mismo resultado que en el caso de un aprendizaje por refuerzo. Los resultados son los siguientes:

Episodio	Numero de Pasos	Recompensa
1	8	3
2	6	7
3	20	-70
4	14	-38
5	4	11
6	6	7
7	4	13

Tabla 5.1. Episodios hasta convergencia (0 Entrenam.)

Como vemos, han hecho falta 7 episodios de exploración. El total de pasos hasta realizar por primera vez la solución optima (en el episodio 7) han sido 62.

En el caso de realizar **un primer episodio de entrenamiento**, obtenemos el siguiente resultado:

Episodio	Numero de Pasos	Recompensa
1	4	11
2	6	7
3	20	-70
4	4	11
5	5	10
6	4	11
7	4	11

Tabla 5.2. Episodios hasta convergencia (1 Entrenam.)

Como vemos, ha hecho falta un episodio de entrenamiento, más 5 episodios de exploración. El total de pasos hasta realizar por primera vez la solución óptima (en el episodio 6) han sido 43.

En el caso de realizar **dos episodios de entrenamiento**, obtenemos el siguiente resultado:

Episodio	Numero de Pasos	Recompensa
1	4	11
2	4	11
3	6	7
4	20	-70
5	4	11
6	4	11
7	4	11

Tabla 5.3. Episodios hasta convergencia (2 Entrenam.)

Como vemos, han hecho falta dos episodios de entrenamiento, más 3 episodios de exploración. El total de pasos hasta realizar por primera vez la solución óptima (en el episodio 5) han sido 38.

En el caso de realizar tres **episodios de entrenamiento**, obtenemos el siguiente resultado:

Episodio	Numero de Pasos	Recompensa
1	4	11
2	4	11
3	4	11
4	6	7
5	8	-29
6	4	11
7	4	11

Tabla 5.4. Episodios hasta convergencia (3 Entrenam.)

Como vemos, han hecho falta tres episodios de entrenamiento, más 3 episodios de exploración. El total de pasos hasta realizar por primera vez la solución óptima (en el episodio 6) han sido 30.

En el caso de realizar cuatro **episodios de entrenamiento**, obtenemos el siguiente resultado:

Episodio	Numero de Pasos	Recompensa
1	4	11
2	4	11
3	4	11
4	4	11
5	4	11
6	4	11
7	4	11

Tabla 5.5. Episodios hasta convergencia (4 Entrenam.)

Como vemos, con los cuatro episodios de entrenamiento, hemos conseguido enseñarle la solución correcta. El total de pasos hasta realizar por primera vez la solución óptima es de 20.

La siguiente grafica representa como, con cada episodio de entrenamiento, el número de pasos hasta encontrar la solución correcta disminuye. Hay que tener en cuenta que contamos los pasos dados para su entrenamiento como pasos en búsqueda de la solución óptima:

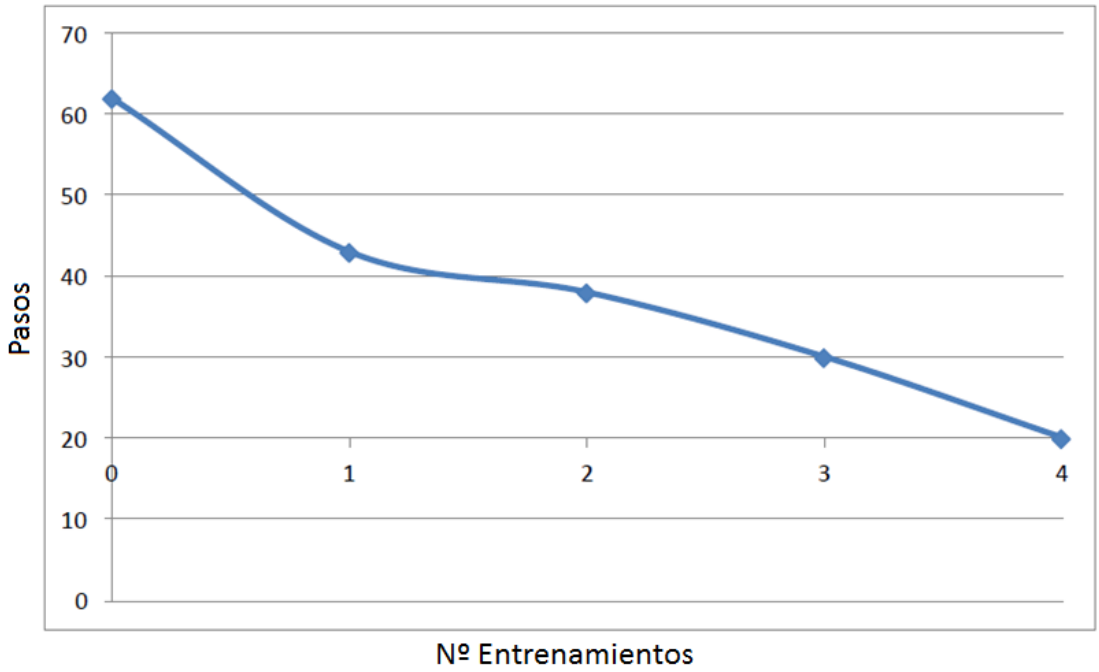


Figura 5.3. Gráfica de aceleración del aprendizaje 1

Para 5 o más episodios de entrenamiento, obviamente el robot no aprende más rápido, y dichos episodios son innecesarios. Como vemos el primer episodio de entrenamiento es el que más acelera el aprendizaje, seguido del cuarto (y último) episodio de entrenamiento.

Ahora vamos a hacer una prueba con otra función de recompensas. En esta prueba, todos los estados tienen una recompensa de -1, salvo el estado final que tiene una recompensa de 0.

En este ejemplo, no le damos información adicional de cómo de útil es un estado, sino que el agente aprenderá la utilidad del estado mediante la exploración. Esto provoca que el agente necesite más pasos que en el caso anterior. Tras hacer los cambios en las funciones para adaptarlo a las nuevas recompensas, obtenemos la siguiente aceleración del aprendizaje:

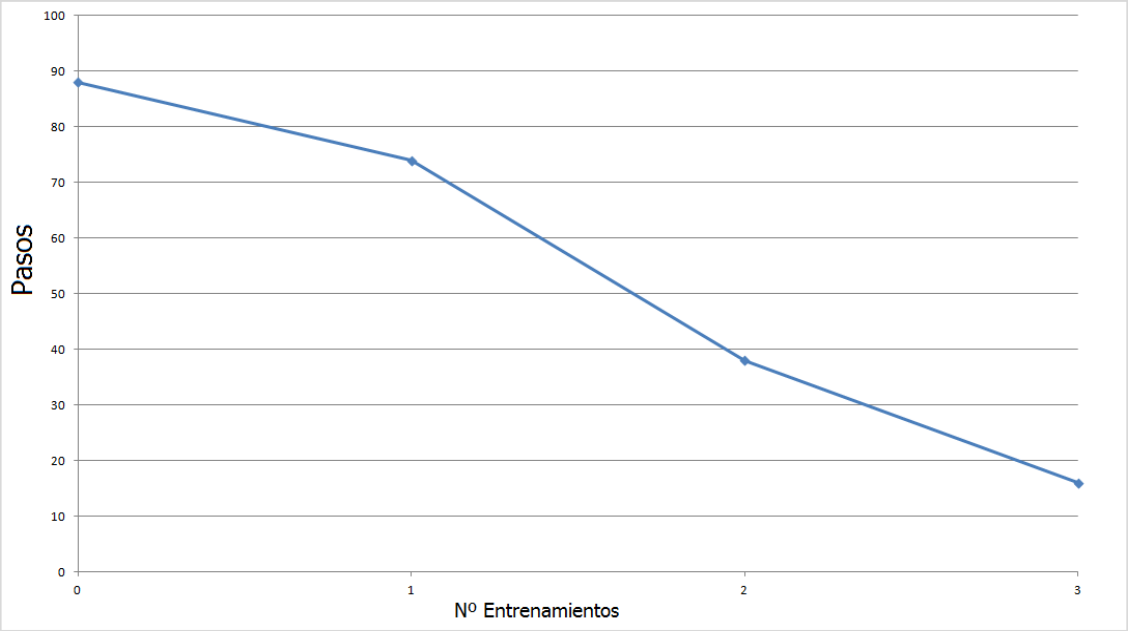


Figura 5.4. Gráfica de aceleración del aprendizaje 2

El **segundo conjunto de pruebas** está destinado a demostrar que el robot es capaz de seguir la ruta que nosotros elijamos, siempre que le demos un número suficiente de ejemplos. Para ello, **partiendo de la situación de haber aprendido la otra ruta posible mediante exploración pura**, buscaremos a partir de cuantos episodios de entrenamiento el robot cambia su ruta hacia el estado terminal a la que le sugerimos nosotros.

Por ello, se ha ido incrementando el número de episodios de entrenamiento, hasta que hemos conseguido modificar su ruta. **El número de episodios de entrenamiento que hemos necesitado para hacerle cambiar de ruta ha sido de 1 episodio**. Más episodios únicamente reforzarán que el robot elija ese camino; si bien esto es algo que el agente hace automáticamente al realizar la explotación de la ruta sugerida. La razón de solo necesitar 1 episodio es que ambas rutas son igual de buenas para nuestro robot.

De la misma forma, vamos a estudiar cuantos episodios de **entrenamientos iniciales necesitamos para influir en la ruta final** que seguirá el robot. Tras las pruebas, hemos conseguido modificar, de forma que nos aseguremos totalmente de que siga la ruta deseada, tras 3 episodios de entrenamiento. Al hacer esto, la ruta descartada nunca será visitada (al tener un bajo Q(S, A)), y nuestro robot seguirá la ruta que deseábamos. Si deseáramos cambiar de ruta, deberíamos, mediante 3 episodios de entrenamiento, enseñarle la otra ruta.

En el caso de mentirle, instruyéndole con acciones que no son las correctas (como hacer acciones que me lleven a repetir el mismo estado una y otra vez), cuando empiece su exploración tomará como base lo enseñado por nosotros. Hay que tener en cuenta que incluso cuando estamos haciendo un episodio de entrenamiento, el robot va actualizando los valores Q(S, A).

Esto le da, cuando empieza a explorar, algo de información de la utilidad de los estados, al margen de lo que el instructor opinaba como correcto. Así, el robot ha necesitado de menos episodios de exploración (y con menos pasos) que si no hubiéramos hecho algún episodio de entrenamiento, incluso siendo este parcialmente incorrecto. El grado de “ayuda” al robot depende de cómo de equivocadas fueran las acciones, y de las acciones en concreto que tomemos. Es por ello que no se ha puesto un ejemplo representativo en esta sección.

Capitulo 6: Conclusiones y Trabajo Futuro

Llegados a este punto, podemos afirmar que hemos logrado completar con éxito los objetivos que perseguíamos con el proyecto. Vamos a repasar el grado de consecución de los mismos:

- **Algoritmos de control de la estabilidad del robot**: Buscábamos controlar la estabilidad del robot de tal forma que el robot mantuviera la estabilidad bajo las condiciones de escenario donde iba a moverse. Este problema ha podido ser resuelto de forma complemente satisfactoria mediante el uso del modelo AnyWay; como la prueba de estabilidad nos ha demostrado.

- **Algoritmos de control de posición y movimiento en el seno del entorno:** Queríamos poder mover el robot a cualquier posición del entorno, para hacer posible la navegación del robot. Este objetivo ha sido resuelto mediante la combinación de los controladores de movimiento y posición; como la prueba de control de navegación ha demostrado.

- **Algoritmo de aprendizaje por confirmación**: Queríamos una implementación basada en aprendizaje por refuerzo; y demostrar que mediante el uso de ejemplos, el robot es capaz de acelerar su aprendizaje. Este objetivo ha sido resuelto mediante la implementación en MATLAB de los algoritmos de aprendizaje por refuerzo basado en SARSA; y hemos podido demostrar que existe una aceleración del aprendizaje cuando damos ejemplos al algoritmo durante las pruebas.

Si bien, aunque los resultados obtenidos han sido muy satisfactorios en lo que respecta a los objetivos que perseguíamos con el proyecto, existen una serie de puntos que podrían desarrollarse con mayor profundidad, y los cuales no se han seguido desarrollando por salirse de nuestros objetivos. Algunos de estos puntos son:

- **Mejora de la precisión en los desplazamientos:** Por las características del sistema NXT, no podemos esperar una completa precisión en nuestras actuaciones. Una primera fuente de imprecisiones son los sensores, los cuales tienen una baja precisión como ya hemos comentado. En nuestro caso, estas imprecisiones vienen del sensor de ultrasonidos y de los encoder de los servomotores; los cuales pueden no dar siempre una información veraz del entorno.

- **Implementación y pruebas con diferentes recompensas:** En este proyecto hemos realizado la implementación del algoritmo de aprendizaje por confirmación sobre un algoritmo de aprendizaje por refuerzo basado en SARSA con una función de recompensas estática (basada en la distancia al estado terminal), y en base a esta función de recompensas se han realizado las pruebas. Realizar las pruebas con distintas funciones de recompensa podría ser interesante, ya que puede desprender mayor información sobre la forma de obtener la solución óptima a partir del entrenamiento.

Además, podría ser interesante realizar una implementación del control de estabilidad utilizando como solución un controlador no basado en NXT-G; para no tener que utilizar los buzones NXT como elemento intermedio en la comunicación.

Bibliografía

[ANYWAYW] http://robotsquare.com/2012/03/13/tutorial-segway-with-nxt-g/

[APRAUT] Pajares, G, Jesus M, de la cruz “Aprendizaje Automático. Un enfoque práctico”, Capitulo 12, RA-MA

[CEAANY]http://www.ceautomatica.es/sites/default/files/upload/955/files/VIII_Simposio_2012/Normativa_Concurso_Control_Inteligente_2Ed_2012.pdf

[EMBEDCONT] Berthilsson, S, Danmark, A, Hammarqvist, U, Nygren, H, Savin, V. “Embebedded Control Systems LegoWay“ http://www.it.uu.se/edu/course/homepage/styrsystem/vt09/Nyheter/Grupper/g5_Final_Report.pdf

[HITECH] http://www.hitechnic.com

[INTPEND] http://eprints.ucm.es/16096/1/memoriaPFC.pdf

[JAEW01]http://plaza.ufl.edu/yiz21cn/refer/Stock%20price%20prediction%20using%20reinforcement%20learning.pdf

[LENGCOMP] http://www.teamhassenplug.org/NXT/NXTSoftware.html

[MATLNXT] http://www.mathworks.com/matlabcentral/fileexchange/35206-simulink-support-package-for-lego-mindstorms-nxt-hardware/content/lego/legodemos/html/publish_lego_selfbalance.html

[NILS01] J. Nilsson, Nils. “Inteligencia Artificial. Una nueva síntesis” Mc GrawHill, 2001

[NXTARDU] http://digitaldiner.blogspot.com.es/2012/02/two-wheeled-self-balancing-robot.html

[NXTGPR] Floyd, J. “Lego Mindstorms 2.0 NXT-G Programing Guide”.

[NXTINTR] http://www.ro-botica.com/img/NXT/pdf/NXT%20Guia%20Rapida-ES.pdf

[NXTMAIL] http://www.cs.uleth.ca/~benkoczi/3720/data/NXT_Bluetooth_handout-jeremy.pdf

[NXTTWO] http://nxttwowheels.blogspot.com.es/2011/01/nxt-two-wheels-keys.html

[OGATA02] Ogata, K. “Ingenieria de Control Moderna”. Prentice Hall 2002

[PIDCONTR] https://sites.google.com/site/controlandelectronics/understanding-pid

[PIDEXAMP] http://www.modelingandcontrol.com/Industrial-Applications-of-PID-Control.pdf

[RUSS04] S. Russel, P. Norvig. “Artificial Intelligence: A Modern Approach, Prentice-Hall”, 2ª edición, 2004.

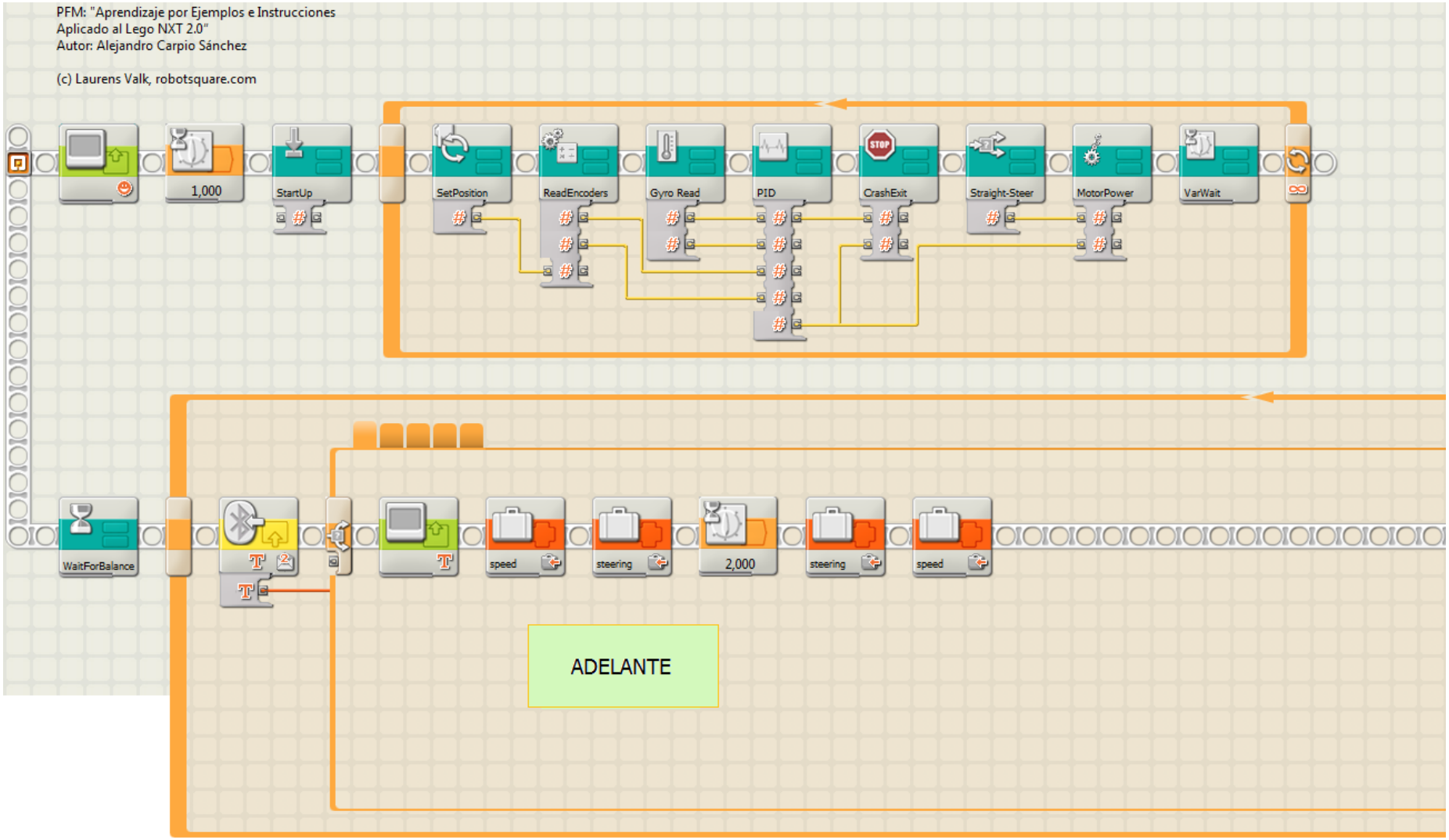
[RWTHW] http://www.mindstorms.rwth-aachen.de/

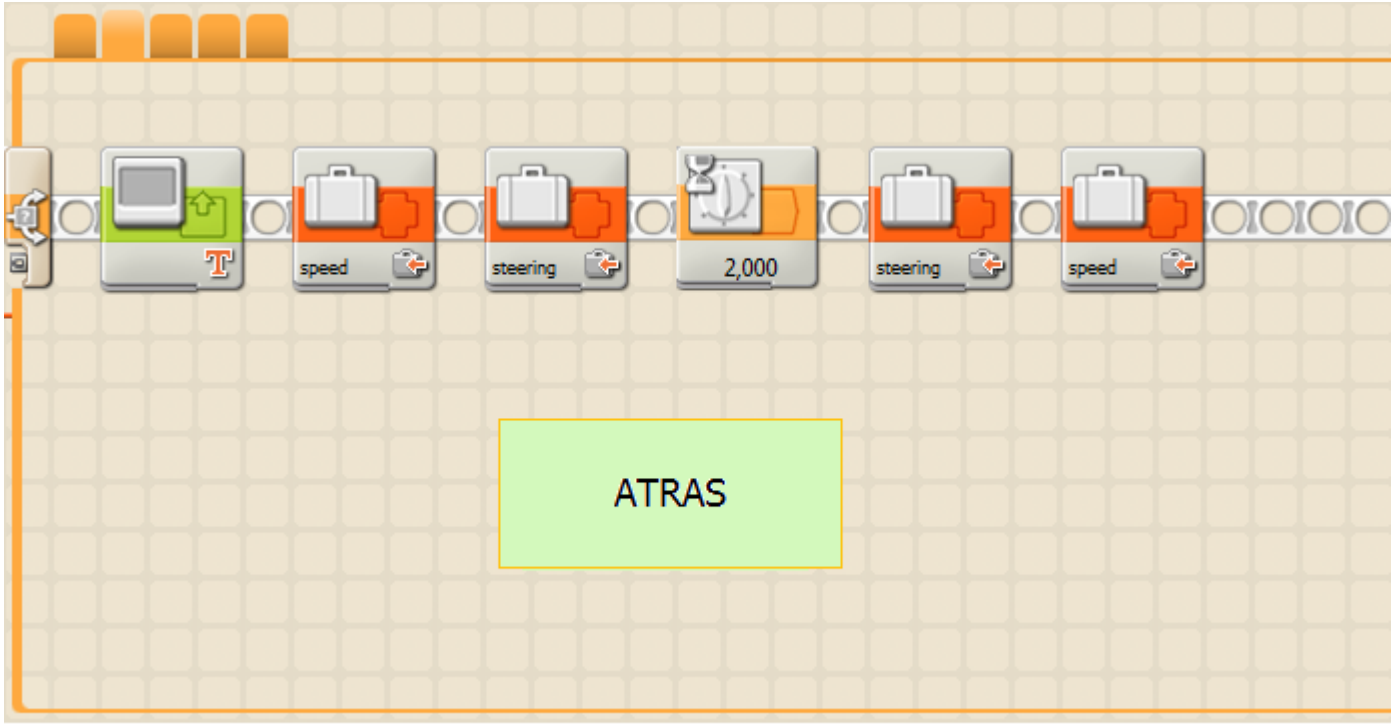
[SELFVIDEO] http://www.youtube.com/watch?v=Tc93WCm2uPU

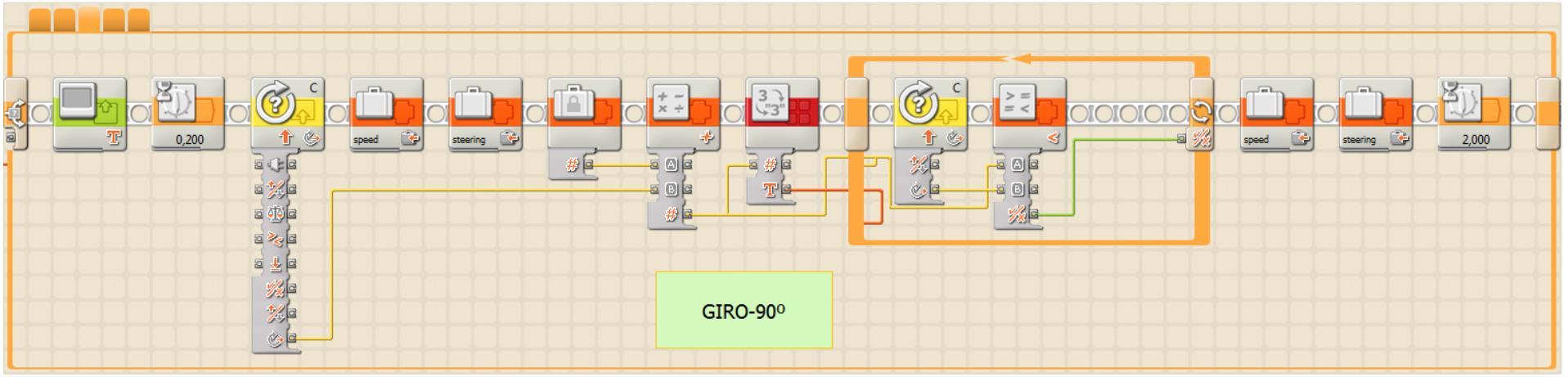
[SUTT98] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning, MIT Press, Cambridge, MA, 1998.

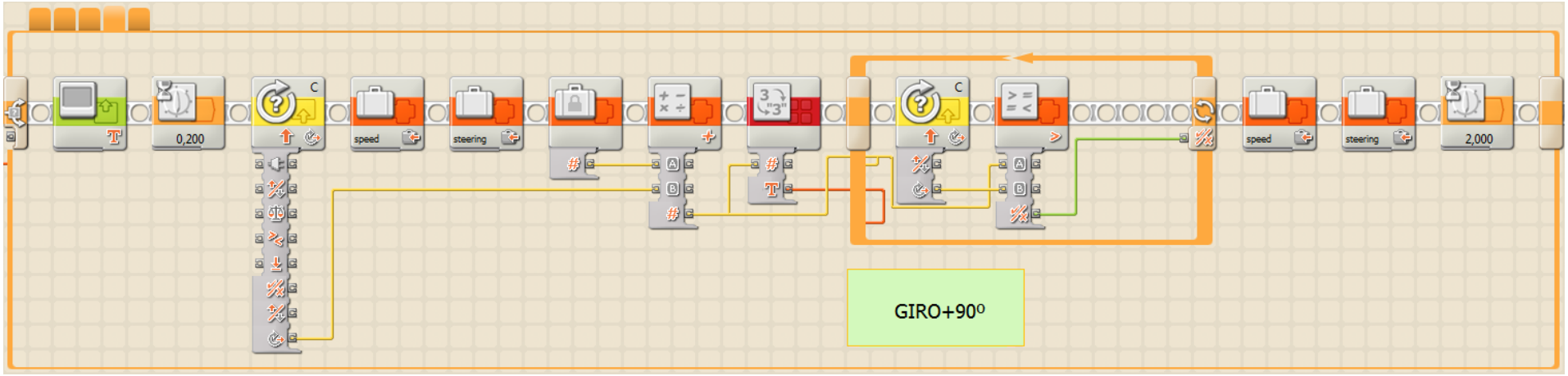
[Xiao06] http://cdn.intechopen.com/pdfs/686/InTech-Reinforcement_learning_based_supervisory_control_strategy_for_a_rotary_kiln_process.pdf

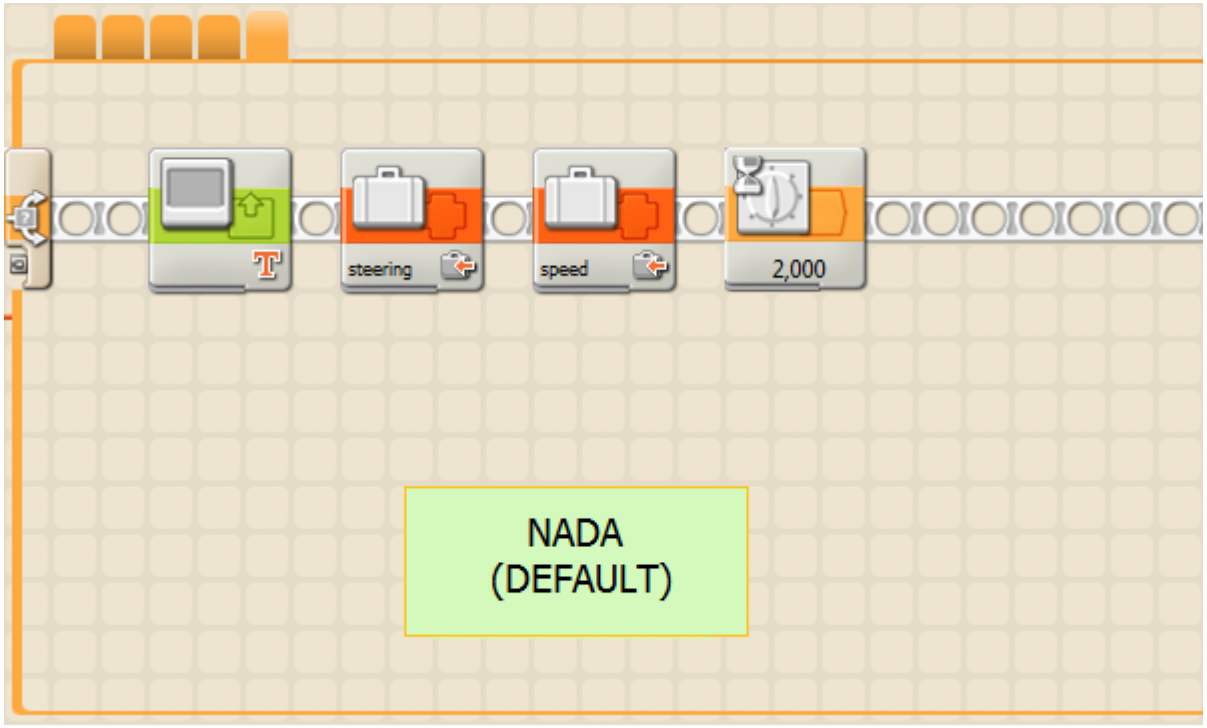
ANEXO A:
Diagrama de Bloques Control Estabilidad (ANYWAY)


PFM: "Aprendizaje por Ejemplos e Instrucciones
Aplicado al Lego NXT 2.0"
Autor: Alejandro Carpio Sánchez
(c) Laurens Valk, robotsquare.com
1,000
StartUp
SetPosition
ReadEncoders
Gyro Read
PID
CrashExit
Straight-Steer
MotorPower
VarWait
WaitForBalance
speed
steering
2,000
steering
speed
ADELANTE


speed
steering
2,000
steering
speed
ATRAS


0,200
speed
steering
speed
steering
2,000
GIRO-90º


0,200
speed
steering
speed
steering
2,000
GIRO+90º


steering
speed
2,000
NADA
(DEFAULT)