UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Máster Universitario de Ingeniería de Sistemas Automáticos y Electrónica Industrial.

TRABAJO FINAL DE MÁTER

Estudio e implementación de un controlador para un robot tipo Segway y de los algoritmos que lo capacitan para el seguimiento de trayectorias desconocidas.

Alumna: Paola Andrea Lora Thola

16 de marzo de 2015

Directora: Rita María Planas Dangla.
Escuela Técnica Superior de Ingenierías Industrial
y Aeronáutica de Terrassa

UNIVERSIDAD POLITÉCNICA DE CATALUNYA (UPC)

Índice:

Índice de figuras:

Índice de tablas:

1. INTRODUCCIÓN.

El proyecto recae en el estudio e implementación de un controlador para un robot móvil de morfología inestable (*Segway*). El controlador deberá permitir la estabilización del robot durante todo su movimiento. Así mismo, el proyecto comprende la programación del robot para el seguimiento de trayectorias previamente desconocidas, utilizando datos sensoriales. Las trayectorias deberán poder contemplar desniveles así como acciones a realizar durante el recorrido.

1.1. Objetivos y abasto del Trabajo final de Máster

Este proyecto aborda uno de los problemas clásicos de la teoría de control, el péndulo invertido, en este caso sobre dos ruedas; para llevarlo a cabo se ha utilizado la plataforma Lego Mindstorms, en concreto la serie NXT 2.0 junto con un sensor giroscópico y otro de color de la marca Hitechnic para configurar un robot de tipo *Segway*.

Los objetivos del proyecto son:

- Construir con los elementos del kit Lego Mindstorms NXT 2.0 un robot del tipo *Segway*.
- Desarrollar e implementar algoritmos de control que permitan mantener el robot en equilibrio durante todo el recorrido. Para mantener el robot en equilibrio se utilizará un sensor giroscópico, que indicará la inclinación actual del robot.
- Desarrollar e implementar algoritmos de control que permitan al robot seguir una trayectoria desconocida. Este objetivo se realizará gracias a un sensor de color, es decir que la trayectoria será opto guiada, la trayectoria se indicará en forma de línea negra marcada en el suelo.
- La trayectoria a seguir deberá comprender rectas, curvas y desniveles con pendientes de subida y bajada.
- Las acciones a realizar podrán ser la de parar de seguir la trayectoria durante un momento sin perder el equilibrio, si se encuentra con un objeto delante o parar si encuentra en la trayectoria un color específico.
- El robot debe soportar perturbaciones externas moderadas sin perder el equilibrio.

La estrategia a seguir es la siguiente:

- Elaborar un modelo matemático del prototipo del robot.
- Con ayuda del programa *Simulink* de *Matlab* se realizarán las simulaciones necesarias para diseñar un controlador que nos permita mantener el robot en equilibrio.
- Se realizará un programa con la acción de control proporcionada por el controlador, con el programa *Bricx Command Center* (*BricxCC*).
- Una vez se consigue que el robot se mantenga en equilibrio, se procede a realizar el control del seguimiento de trayectoria.
- Posteriormente se añade las acciones a realizar durante el recorrido.

1.1.1. Entorno de trabajo.

En este apartado se explicará de forma gráfica el entorno de trabajo del actual trabajo. En la figura 1 se encuentra la parte teórica y de simulación del entorno de trabajo. Un robot tipo *Segway* no deja de ser un péndulo invertido sobre dos ruedas, este hecho se utiliza para la realización del modelo matemático del robot. Una vez se obtiene el modelo matemático, se procede a la simulación con el software *Matlab y Simulink*. Con la simulación del modelo matemático, se puede obtener un controlador que cumpla con las especificaciones que requiere el sistema para mantenerse en equilibrio.

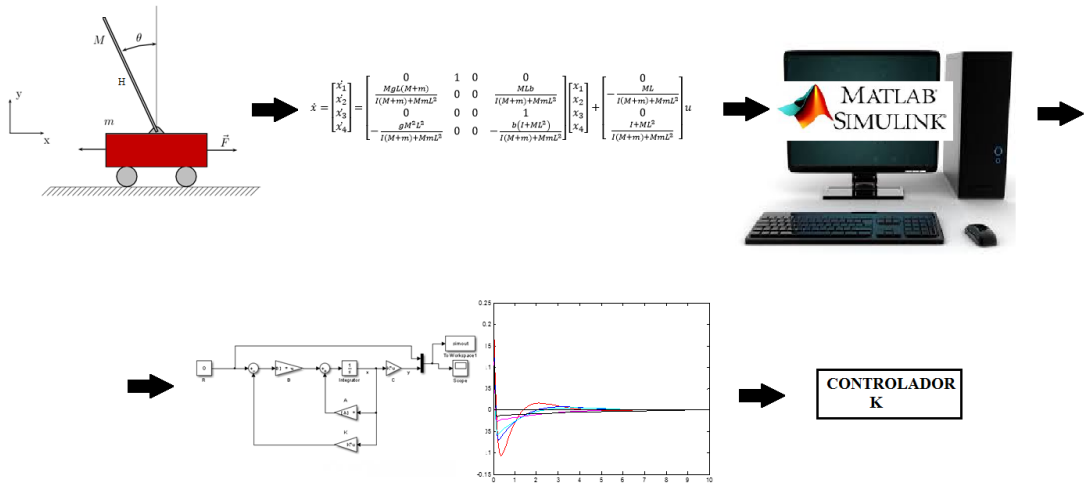


Figura 1: Entorno de trabajo I.

En la figura 2 se encuentra el entorno de trabajo con el sistema real. En esta imagen se ve que a partir del controlador encontrado en la parte teórica y de simulación, se desarrolla un programa con ayuda del software BricxCC, que posteriormente se carga en el *Brick* del robot a través de un cable USB. Este programa desarrollado incorpora otros aspectos que no se reflejan en este apartado pero que se explicarán más adelante. Este programa permitirá que el robot realice las acciones marcadas como objetivos en el apartado anterior.

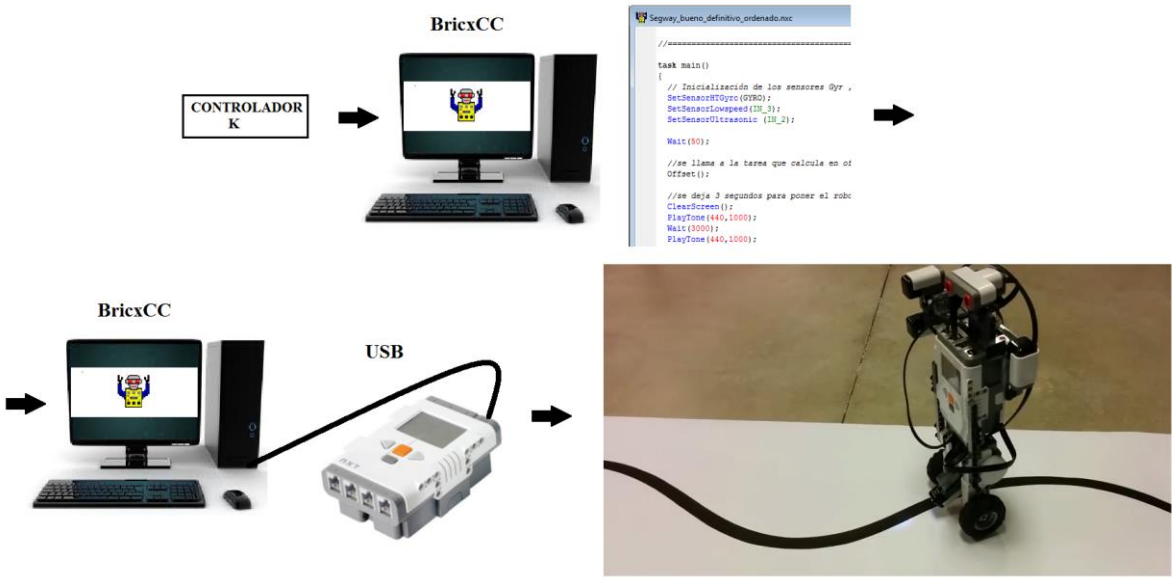


Figura 2: Entorno de trabajo II.

1.1.2. Lego Mindstorms.

Lego Mindstorms es un kit de robótica educativa creado por Lego junto con National Instruments, este kit permite la construcción de diferentes configuraciones de robots. Hasta el momento a lo largo de su historia se han creado 3 generaciones de bloques que son los siguientes:

Primera generación: RCX.

Puesto a la venta en el año 2001, se realizaron 3 versiones 1.0, 1.5 y 2.0; sus características más destacadas son las siguientes:

- Dispone de 3 puertos de entrada y 3 de salida.
- Memoria ROM de 16 Kb, una memoria RAM externa de 32 kb.
- Pantalla LCD.
- Aporta hasta 9V en los puertos de salida.
- La principal desventaja es que no permiten ejecutar varias instrucciones a la vez.

Figura 3: Lego Mindstorms RCX.

Segunda generación: NXT.

Fue en el año 2006 que salió a la venta su primera versión. Las características generales de los NXT son las siguientes:

- Memoria Flash de 256 Kb y 64 Kb de RAM externa.
- 4 puertos de entradas y 3 de salida.
- La versión 2.0 contienen interfaz USB y Bluetooth.
- Servo motores que permiten la detección de giros de la rueda.
- Aporta hasta 9V en los puertos de salida.
- Permite ejecutar varias instrucciones a la vez.

Figura 4: Lego Mindstorms NXT.

Tercera generación: EV3.

La última generación llegó en el año 2013. Características:

- 4 puertos de entrada y 4 de salida.
- Un puerto mini USB para PC, un puerto de host USB (para agregar un conector WiFi y establecer conexiones en cadena), un puerto para tarjetas Micro SD
- Receptor Bluetooth y wifi.
- Servo motores.
- Permite ejecutar varias instrucciones a la vez.

Figura 5: Lego Mindstorms EV3.

1.2. Requerimientos del trabajo final de máster.

La realización del presente proyecto requiere de los siguientes elementos:

- Kit Lego Mindstorms 2.0

Figura 6: Kit Lego Mindstorms 2.0.

- Sensor color: se ha utilizado el sensor de color V2 de Hitechnic.

Figura 7: Sensor Color V2 de Hitechnic.

- Sensor gyroscopico: sensor Gyro de Hitechnic

Figura 8: *Sensor Gyro de Hitechnic.*

- Sensor ultrasónico:

Figura 9: sensor ultrasónico.

1.2.1. Lego Mindstorms NXT 2.0.

La plataforma Lego Mindstorms NXT 2.0 está formado por una unidad central procesadora (llamada bloque o *Brick*), sensores, actuadores y comunicaciones.

Figura 10: NXT 2.0 con sensores y actuadores.

La unidad central está constituida por un microprocesador principal de 32 bits que incluye 256kb de memoria Flash y 64 kb de RAM externa, y un microprocesador secundario de 8 bits con 4 kb de memoria Flash y 512 bytes de RAM externa. Contiene cuatro puertos de entradas analógicas o digitales y tres puertos de salida que están capacitados para obtener la señal de entrada de los encoders de los motores.

Los sensores que se pueden conectar al bloque pueden ser oficiales o no oficiales:

Oficiales: luz, contacto, color y ultrasonidos.

No oficiales: giroscópico, temperatura, aceleración, infrarrojo, magnético, sonido, barométrico, proximidad, ángulo de rotación y brújula. Muchos de estos sensores son fabricados por Hitechnic.

La comunicación entre la unidad central y los sensores/actuadores se realiza mediante conexión de un cable 6-wire, con el protocolo I2C. La comunicación con otros elementos como el PC u otra unidad central se realiza mediante los interfaces de comunicación USB 2.0 y/o Bluetooth.

1.3. Justificación y necesidad del trabajo final de máster

Básicamente la justificación y necesidad de este trabajo surge de querer llevar a la práctica conocimientos adquiridos en máster.

El método de aprendizaje en asignaturas de control se centra en la enseñanza de los conceptos teóricos, seguidos de prácticas de simulación con software especializado, este proyecto supone llevar a la práctica la implementación de uno de los retos clásicos de control, como lo es el péndulo invertido, planteando una solución para el mismo.

Una de las motivaciones principales es la de combinar los conocimientos de teoría de control y de robótica obtenidos en el máster, para desarrollar e implementar un controlador capaz de cumplir con las especificaciones del proyecto.

2. ANTECEDENTES.

Segway es el primer dispositivo de transporte personal con auto balanceo controlado por ordenador, se trata de un vehículo ligero de dos ruedas que se mantiene en equilibrio en todo momento El usuario se debe balancear en la dirección que desea tomar.

Figura 11: Segway PT (personal transport).

Este sistema *Segway* puede simplificarse a un péndulo invertido sobre dos ruedas accionadas por dos servomotores DC independientes. El sistema es inestable y no lineal que requiere un sistema de control para mantenerlo en equilibrio incluso con perturbaciones externas o internas, como las aceleraciones de los motores que pueden desequilibrarlo.

2.1. *Segway* con Lego Mindstorms.

Dentro de la plataforma Lego Mindstorms se han realizado diferentes proyectos con el objetivo de mantener un robot en equilibrio. A continuación se resumen algunos:

- *LegWay*: fue desarrollado con el *Brick* RCX, el programa fue escrito con *BrickOS* (*legOS*). Para mantener el equilibrio se utilizó un sensor de proximidad, por lo que si la distancia disminuye los motores de mueven hacia delante y si aumenta se mueven hacia atrás.

Figura 12: LegWay

- NXT Two Wheels balancing robot tricks & keys: en este caso se ha utilizado la plataforma NXT con el sensor giroscópico de Hitechnic.

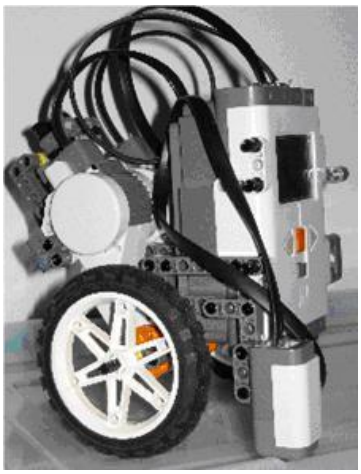
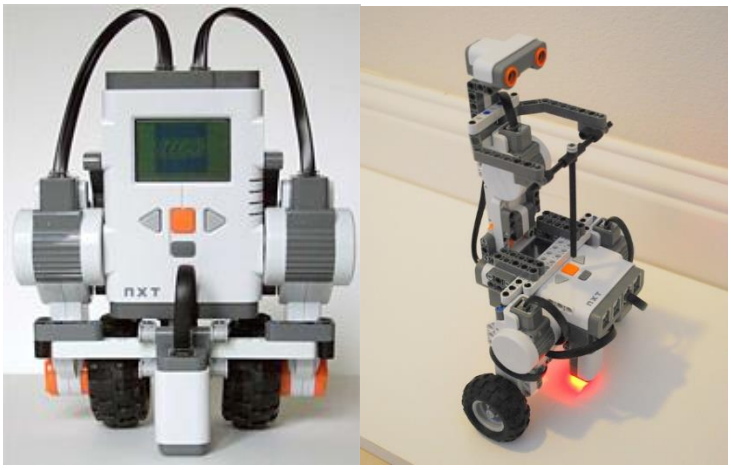


Figura 13: NXT Two Wheels balancing robot tricks & keys

- *NXTway* o NXT *Segway* with Rider: *Segway* con la plataforma NXT con sensor de luz.



(a) (b)

Figura 14: (a) NXTway, (b) NXT Segway with Rider

- NXTway-G: *Segway* con plataforma NXT y sensor giroscopico.

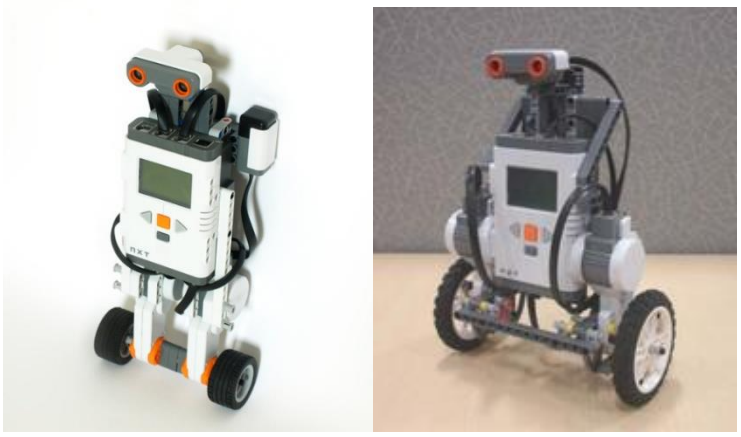


Figura 15: NXTway-G.

2.2. Soluciones alternativas.

La plataforma lego permite la construcción de múltiples robots, incluidos robots tipo *Segway*. Como se ha visto en el apartado anterior existen varias posibilidades de implementación de *Segway*, se ha optado por seleccionar una configuración existente y no una completamente nueva; ya que de esta manera nos da ciertas garantías de éxito sobre el presente trabajo y nos ofrece también la posibilidad de encontrar referencias, posibles dificultades, comentarios, etc. En este apartado se dará a conocer cuál es la configuración seleccionada y las razones por las que se selecciona.

El robot NXT Two Wheels balancing robot tricks & keys de la figura 13 no se selecciona debido a que es una configuración un poco aparatosa, no lleva incorporado el sensor

ultrasónico, no lleva incorporado el sensor de color y que la posición del sensor giroscópico que casi está rozando el suelo, no se considera una posición acertada para este sensor. Aunque sería posible incluir los sensores necesarios existen mejores configuraciones.

NXTway junto con NXT *Segway* with Rider son dos configuraciones con las que el equilibrio se realiza mediante un sensor de luz, figura 14. Con el sensor giroscópico se puede obtener un equilibrio más robusto que con el sensor de luz, es por esta razón por las que se descartan estas dos configuraciones. Otra razón por la que se descarta es que uno de los objetivos es también el seguimiento de trayectorias y para ello se requiere un sensor de color, no se considera razonable utilizar un sensor de luz junto a un sensor de color delante del robot.

Otra configuración tipo *Segway* es la de un péndulo invertido sobre una rueda, se puede ver una imagen de esta configuración en la figura 16. Esta configuración es descartada a causa de la imposibilidad de control sobre los giros del robot, ya que es esencial tener un control sobre el giro para el seguimiento de la trayectoria.

Figura 16: Péndulo invertido con una rueda.

Robot NXTway-G, esta es la configuración seleccionada para la realización de este proyecto. En la figura 15 se puede ver la forma que tiene esta opción, dentro de esta configuración existen diversas modificaciones, en la figura 17 se ecuentra la configuración seleccionada con la posición de los sensores. Se ha seleccionado ya que utiliza un sensor giroscópico para el control del equilibrio, lleva incorporado el sensor ultrasonido y de color.

Figura 17: Configuración Segway seleccionada.

3. PLANIFICACIÓN Y PROGRAMACIÓN.

En este apartado se encuentra la planificación del trabajo final de máster, para ello se ha realizado un diagrama de Gantt. En la tabla 1 se encuentran las tareas junto con la fechas de inicio y final. En la figura 18 se encuentra el diagrama de Gantt.

Tarea	Fecha inicio	Duración	Fecha fin
Búsqueda modelos de *Segway*	25/07/2014	10	04/08/2014
Búsqueda modelos matemáticos	05/08/2014	20	25/08/2014
Simulación modelos matemáticos	26/08/2014	20	15/09/2014
Diseño de controladores	16/09/2014	4	20/09/2014
Información seguimiento de líneas	21/09/2014	8	29/09/2014
Búsqueda antecedentes de Lego Mindstorms	30/09/2014	9	09/10/2014
Búsqueda proyectos similares	10/10/2014	10	20/10/2014
Memoria	21/10/2014	145	15/03/2015
Búsqueda información sensores	22/10/2014	3	25/10/2014
Calibración sensor Gyro	26/10/2014	8	03/11/2014
Entorno de programación	04/11/2014	7	11/11/2014
Selección del entorno de programación	12/11/2014	6	18/11/2014
Búsqueda programas de ejemplo	19/11/2014	11	30/11/2014
Programación de equilibrio	01/12/2014	15	16/12/2014
Programación seguimiento de línea	17/12/2014	10	27/12/2014
Programación acciones a realizar	28/12/2014	8	05/01/2015
Montaje del robot	06/01/2015	3	09/01/2015
Pruebas de calibración del sensor Gyro con el robot	10/01/2015	5	15/01/2015
Pruebas de los controladores diseñados con el robot	16/01/2015	15	31/01/2015
Selección y ajuste del controlador de equilibrio con el robot	01/02/2015	5	06/02/2015
Pruebas y ajuste del controlador de seguimiento de líneas	07/02/2015	8	15/02/2015
Pruebas acciones a realizar	16/02/2015	5	21/02/2015
Programa final	22/02/2015	10	04/03/2015
Pruebas con el programa final	05/03/2015	10	15/03/2015
Entrega	16/03/2015	1	17/03/2015

Tabla 1: Planificación del TFM.

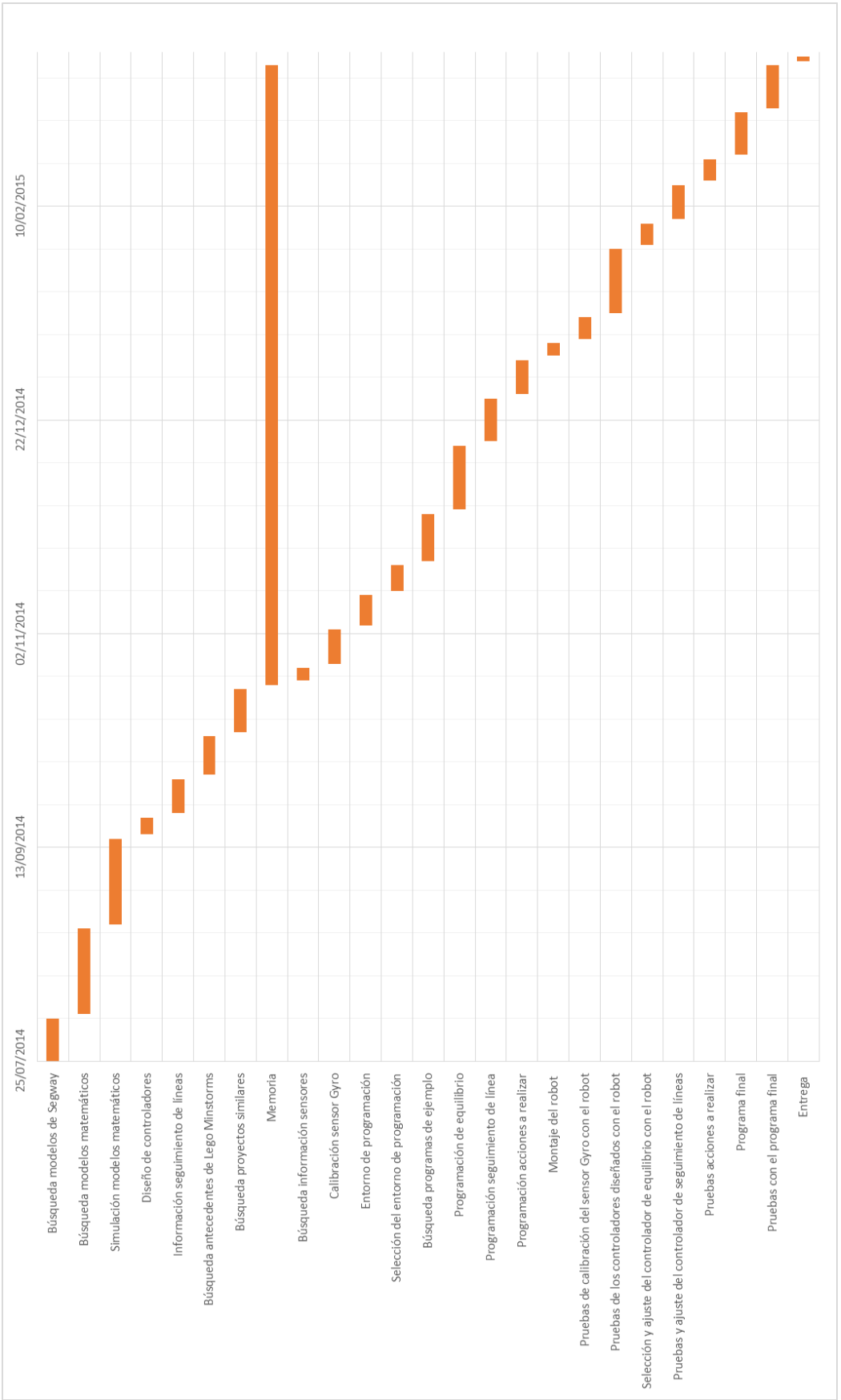


Figura 18: Diagrama de Gantt.

4. CONSTRUCCIÓN DEL ROBOT *SEGWAY*.

Para la construcción del robot *Segway* se ha tenido disposición del kit Lego Mindstorms 2.0, un sensor giroscópico y un sensor de color. En este apartado se describirá cada uno de los elementos que se han utilizado para la construcción de robot.

Las vistas frontal y lateral del robot construido se puede ver en la siguiente figura:

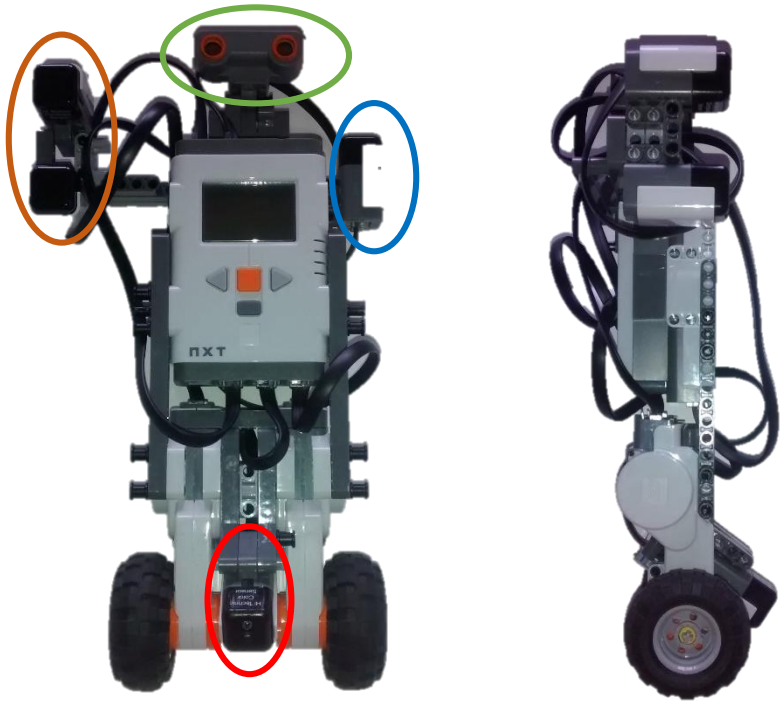


Figura 19: Robot Segway construido.

En la figura 19 se encuentra el sensor giroscópico remarcado en color azul, el sensor de color en color rojo, el sensor de ultrasonido en color verde y en color naranja dos sensores: uno es un acelerómetro y el otro una brújula. Los últimos dos sensores se han integrado debido a que la morfología de *Segway* seleccionada para este trabajo los incorpora, aunque no serán utilizados, se incorporan con el fin de respetar el modelo. Otra utilidad de la incorporación de estos sensores, es que ya se tendrán incorporados su peso y posición en el controlador desarrollado, esto será de utilidad en el caso de futuras aplicaciones que requieran de estos sensores.

4.1. Elementos utilizados.

4.1.1. Lego NXT *Brick* (CPU & I/O).

El *Brick* permite la conexión de sensores y actuadores, pudiendo conectar hasta 4 sensores y 3 actuadores. Dispone de una pantalla LCD frontal de 100x64 pixeles, con botones para interactuar con el usuario. La comunicación se puede hacer vía USB o Bluetooth.

4.1.2. Sensor de color V2.

El sensor de color se utilizará para realizar el seguimiento de la trayectoria.

El sensor de color V2 de Hitechnic utiliza un solo LED de color blanco que ilumina la superficie del objeto, se analiza el componente de color de la luz que se refleja de la superficie y se calcula el número correspondiente. El sensor obtiene 100 valores por segundo.

Según el fabricante, la relación entre los valores que devuelve el sensor y los colores se muestra en la siguiente figura:

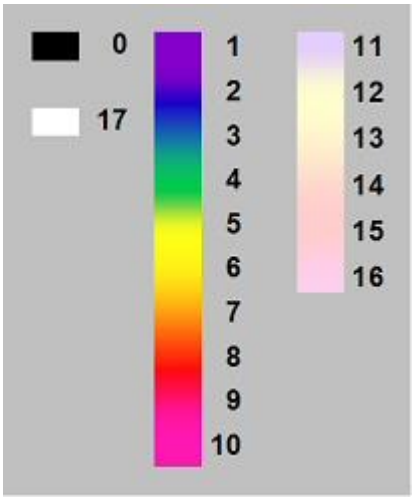


Figura 20: Relación valor-color.

Para un correcto funcionamiento del sensor el fabricante indica que no debe estar demasiado cerca de la superficie de la cual se quiere saber el color, también debe estar a cierto ángulo de ésta. La posición adecuada en la que debe utilizarse el sensor es la indicada en la siguiente imagen:

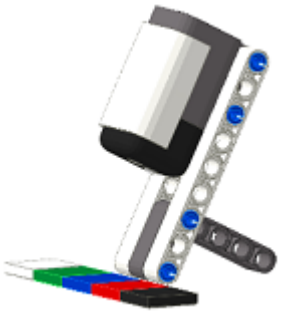


Figura 21: Posición adecuada del sensor de color V2.

En la figura 19 se puede ver que el sensor de color se ha colocado de manera correcta, con el ángulo y distacia al objeto correcto.

Otro aspecto a tener en cuenta a la hora de utilizar el sensor de color es la configuración para la frecuencia eléctrica a la cual opera el pais en el que nos encontremos, en nuestro caso son 50Hz. Para ello se ha de cargar el archivo *SetTo50Hz.rxe* en la unidad central o *Brick*, el archivo lo podemos encontrar en la web del fabricante [1].

4.1.3.Sensor giroscópico.

El sensor giroscópico es esencial en el desarrollo de este proyecto, ya que es el que nos indica la velocidad angular que tiene el robot, por lo que es indispensable para que el robot se mantenga en equilibrio.

Figura 22: Giróscopo clásico.

En la figura 22 se puede ver un ejemplo de giróscopo clásico. Este dispositivo es esencialmente un cuerpo con simetría de rotación que gira alrededor de su eje de simetría. Si es sometido a un momento de fuerza que tiende a cambiar la orientación del eje de rotación su conducta es cambiar de orientación en una dirección perpendicular a la dirección “intuitiva”. Este tipo de sensores debido a su coste, tamaño y mantenimiento no sería viable utilizarlos en este tipo de aplicaciones.

El sensor giroscópico utilizado es un sensor de inclinación de silicio. Contiene una placa de silicio que vibra con corriente electrostática que mueve las partículas de silicio, cuando es sometido a una inclinación genera una fuerza que es medida por los circuitos internos del mismo. Esta fuerza es proporcional a la inclinación a la que es sometido.

El sensor giroscópico de Hitechnic detecta la rotación en un solo eje y devuelve un valor en grados por segundo, por lo tanto nos indica la velocidad angular. También devuelve el sentido de la rotación, por lo que su rango es de +/- 360 °/s. Para este proyecto es suficiente la detección de rotación de un solo eje, ya que queremos conocer la inclinación del robot.

La velocidad angular se lee unas 300 veces por segundo. El eje en el que se toman las medidas se muestra en las figuras 23 y 24.

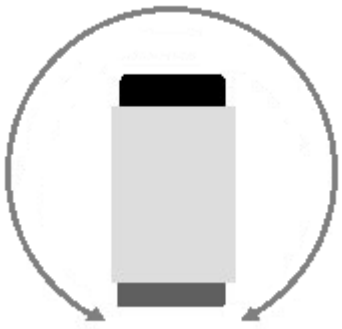


Figura 23: Eje de medida del sensor giroscópico I.

Figura 24: Eje de medida del sensor giroscópico II.

4.1.4.Sensor ultrasónico.

Este sensor permite al robot detectar obstáculos y medir la distancia a la que se encuentran. Su rango de medida es de 4 a 255 cm y tiene una precisión de +/- 3cm. Este sensor tiene el mismo principio de funcionamiento que un radar o un sonar, dispone de un emisor y un receptor, el emisor emite una onda de sonido a un frecuencia muy alta, inaudible al oído humano, cuando el sonido rebota en el obstáculo y vuelve al receptor la distancia al objeto se calcula en función del tiempo que tarda en volver el sonido.

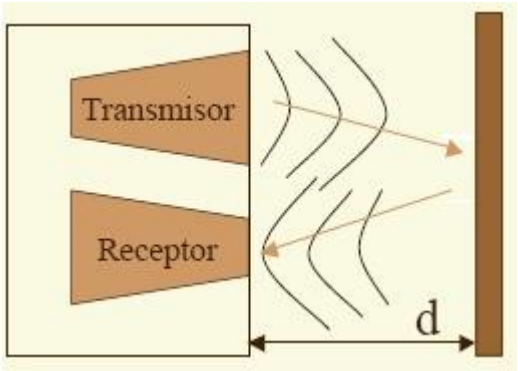


Figura 25: Funcionamiento sensor ultrasónico.

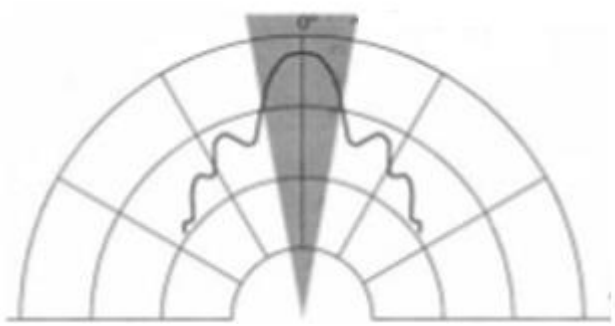


Figura 26: Rango de visión del sensor de ultrasonido.

La periferia de visión del sensor no se limita a objetos que estén justo por delante de él, en la figura 26 se puede ver el rango de visión del sensor. Marcado en color gris se encuentra el cono de visión, se ha de tener en cuenta que el sensor sólo devuelve un valor de distancia. Esto supone que sólo dará información de la distancia a la cual está el objeto y no en qué dirección se encuentra.

4.1.5. Motores/actuadores.

Los motores utilizados en realidad son servomotores, éstos aparte de ser unos motores eléctricos convencionales disponen de sensores de posición, los sensores nos indicarán la posición angular, concretamente se trata de encoders incrementales de 360 pasos por vuelta; esto supone una resolución de 1°/paso. Esta resolución supone que no podremos detectar movimientos menores de un grado, como se comprobará más adelante, es suficiente para mantener el equilibrio del robot. El motor también dispone de un tren de engranajes con los que aumentar su par.

Figura 27: Servomotor y su tren de engranajes.

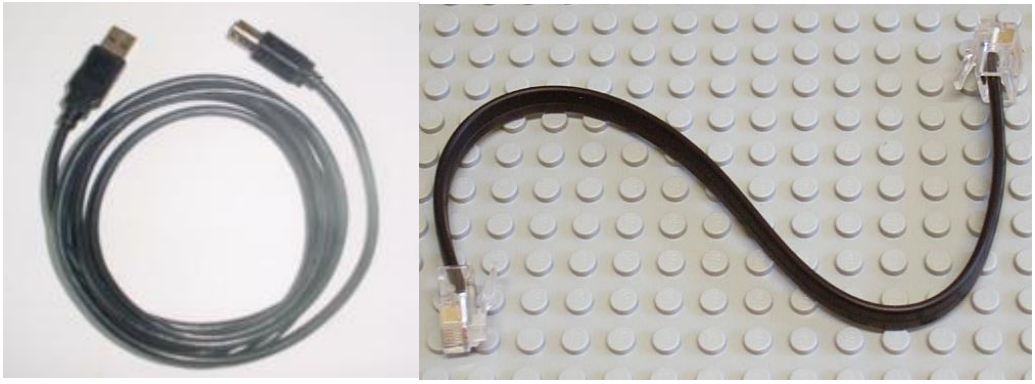
4.1.6. Piezas de construcción.

Las diferentes piezas que se han utilizado en la construcción del robot son las piezas de lego incluidas en el kit de Lego Mindstorms.

Figura 28: Piezas de construcción de Lego.

4.1.7. Elementos de Comunicación.

La comunicación entre el *Brick* y el PC se realizan a través del cable USB (figura 29.a) y la comunicación de los sensores y actuadores con el *Brick* se realizan con el cable que se puede ver en la figura 29.b.



(a) (b)

Figura 29: (a) Cable conexión Brick-PC, (b) cable conexión sensores/actuadores.

5. MODELO MATEMÁTICO DEL SISTEMA.

El péndulo invertido es un problema típico de teoría de control para el cual se han obtenido diferentes modelos matemáticos, y por lo tanto diferentes soluciones. En este apartado se describirá tanto el modelo utilizado como una alternativa no utilizada.

Para llegar a realizar el control de un sistema es muy útil disponer de un modelo matemático del mismo que nos ayude a diseñar un controlador que permita mantener el robot en equilibrio. Es por medio de la simulación que el modelo matemático puede ayudarnos, en concreto las simulaciones se realizarán con el programa *Simulink*. Como se ha dicho anteriormente el robot *Segway* no deja de ser un péndulo invertido sobre dos ruedas, en el apartado 5.2 se desarrollará el modelo matemático del sistema utilizado.

Aunque cabe destacar que se llegaron a realizar diferentes simulaciones para otros modelos matemáticos, en el siguiente apartado se describirán brevemente los controladores más destacados diseñados, pero que no se utilizan en el programa final, por diversas razones, como se explica a continuación.

5.1. Modelos alternativos.

Como se ha dicho anteriormente se disponen de diferentes modelos matemáticos para el péndulo invertido, en este caso se ha utilizado el modelo que se puede encontrar en [2]. En este apartado se mostrará el modelo matemático y los controladores desarrollados.

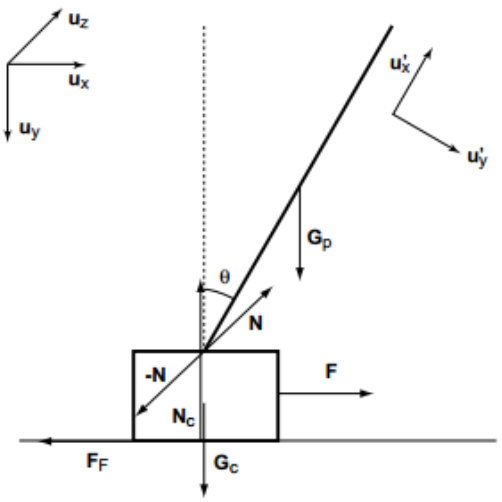


Figura 30: Sistema péndulo invertido alternativo.

En la figura 30 se encuentra el esquema del modelo del sistema, con el que se obtienen las siguientes ecuaciones:

$$\ddot{\theta} = \frac{gsin\theta + cos\theta\left(\frac{-F - ML\dot{\theta}^2 sin\theta}{M+m}\right)}{L\left(\frac{4}{3} - \frac{Mcos^2\theta}{M+m}\right)} \tag{1}$$

$$\ddot{x} = \frac{F + ML\left(\dot{\theta}^2 sin\theta - \ddot{\theta}cos\theta\right)}{M+m} \tag{2}$$

Donde *Θ* es el ángulo del péndulo y *x* es la posición.

Con el modelo anterior y con ayuda del software *Simulink* de *Matlab* se realizó un controlador *Fuzzy*, el sistema en diagrama de bloques del sistema con el controlador que se obtiene es el siguiente:

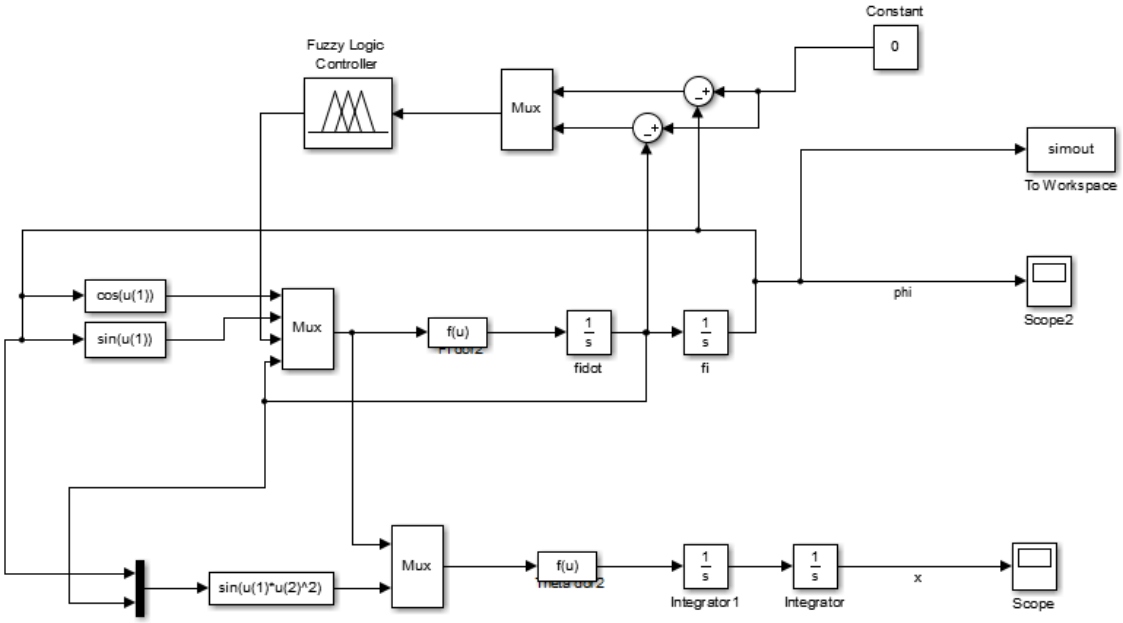


Figura 31: Diagrama de bloques del sistema alternativo.

Los valores que se sustituyen en las ecuaciones (1) y (2) se encuentran en la tabla 2.

Un control *Fuzzy* o de lógica difusa es un controlador ampliamente utilizado. Se denomina difuso debido a que su lógica no se basa en el todo o nada, sino que contempla que los

conceptos pueden ser verdaderos o parcialmente verdaderos. La gran ventaja que tiene este tipo de controladores es que la solución a un problema puede convertirse en términos fácilmente entendibles por un humano, de esta manera la experiencia humana puede ser útil para el diseño del controlador.

El controlador Fuzzy se basa en unas funciones de *fuzzificación* y unas reglas; las funciones pueden ser de diferentes tipos y las reglas del tipo *IF*, las funciones se aplican a las entradas y a las salidas. Como entradas al controlador se utiliza el ángulo del péndulo (phi) y la velocidad angular (phidot), y como salida se obtiene la acción de control. En la figura 32 se encuentran las funciones *Fuzzy* de las entradas, en la figura 33 las de salida y en la 34 las reglas.

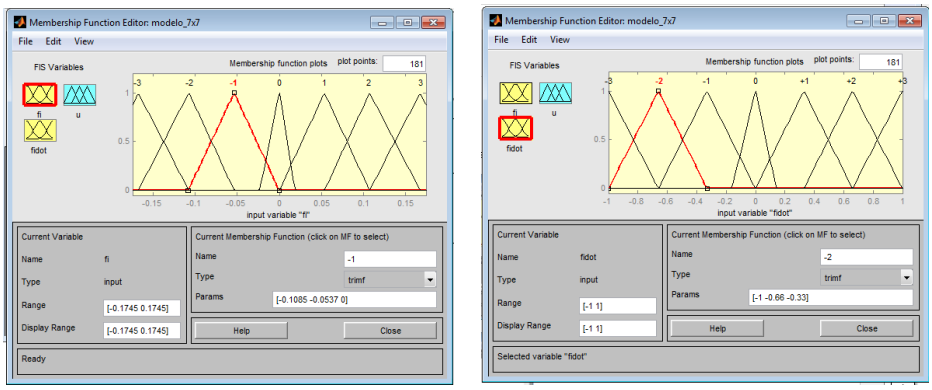


Figura 32: Funciones Fuzzificación entradas.

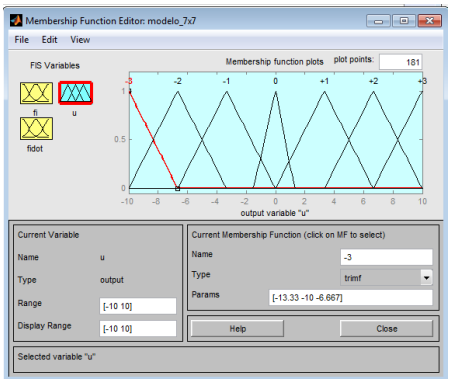


Figura 33: Funciones Fuzzificación salidas.

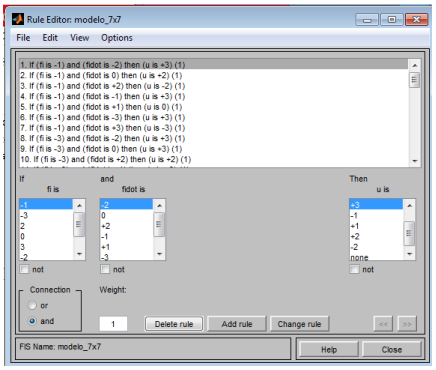


Figura 34: Reglas del controlador Fuzzy.

El ángulo Θ que se obtiene como respuesta al controlador se encuentra en la figura 35.

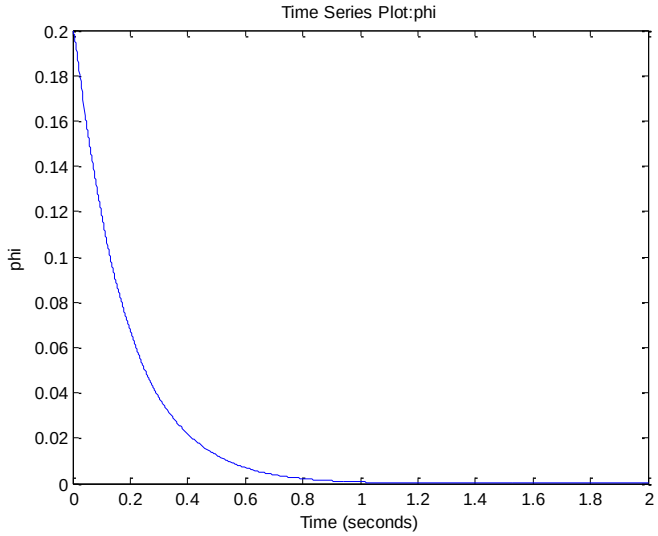


Figura 35: Respuesta al controlador Fuzzy.

Como se puede ver en la figura anterior, se obtiene una respuesta a un ángulo inicial de 0.2 rad (11.4°), la acción de control consigue llevar el ángulo a cero pero tarda mucho en reaccionar, cerca de 5 s en estabilizarse. Esta respuesta en el sistema real no será suficiente.

Esta opción se descarta debido a que tarda mucho en estabilizarse y a que con el software utilizado para desarrollar el programa (BricxCC) no disponía de funciones con las que implementar el controlador fácilmente.

Con este mismo modelo matemático se diseñó también un controlador PID con el mismo software, para ello se calculó la función de transferencia para la inclinación del péndulo Θ:

$$\frac{\ddot{\theta}}{F} = \frac{1}{s^2 L\left(\frac{4}{3}(M+m) - M\right) - g(M+m)} \tag{3}$$

$$\frac{\ddot{\theta}}{F} = \frac{1}{s^2 0.034896 - 5.08676} \tag{4}$$

Con la función de transferencia (3) y los valores de la tabla 2 se obtiene la ecuación (4).

Llevando la función de transferencia a diagrama de bloques y si se le añade un controlador PID se obtiene lo siguiente:

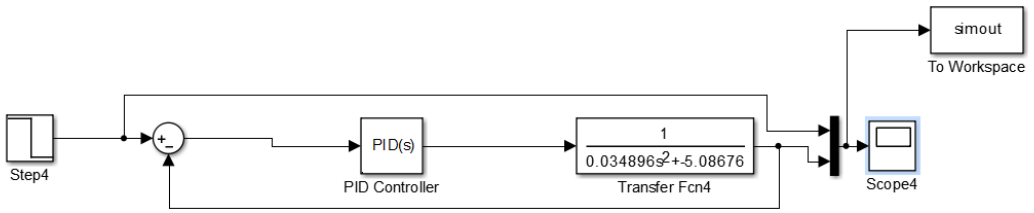


Figura 36: Diagrama de bloques control PID.

Con ayuda de la herramienta de *Matlab Tune* que se encuentra en el bloque PID de *Simulink*, se encuentra el controlador. En la siguiente figura se encuentra la ventana que corresponde con el bloque PID, en ella se encuentran los valores para el controlador, y se marca con un flecha en rojo la herramienta *Tune* con la que han obtenido.

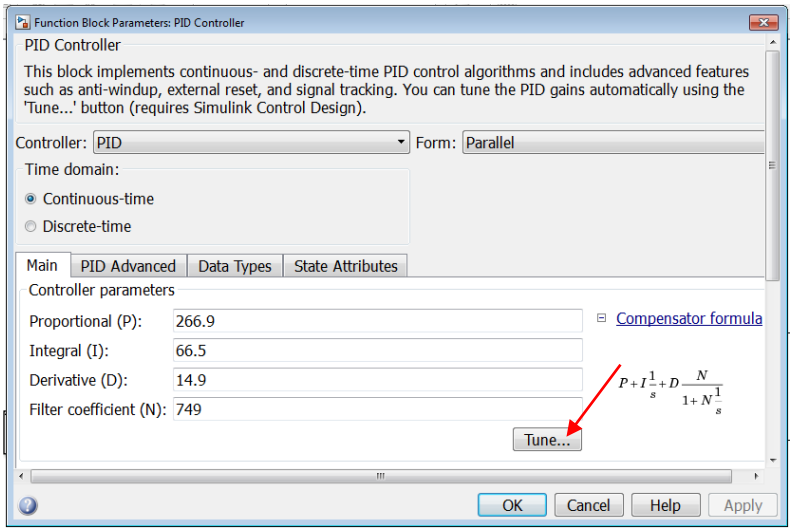


Figura 37: Controlador PID.

En la siguiente figura se encuentra la respuesta del anterior controlador ante un escalón que va desde 0.2 rad (11.4°) a 0°.

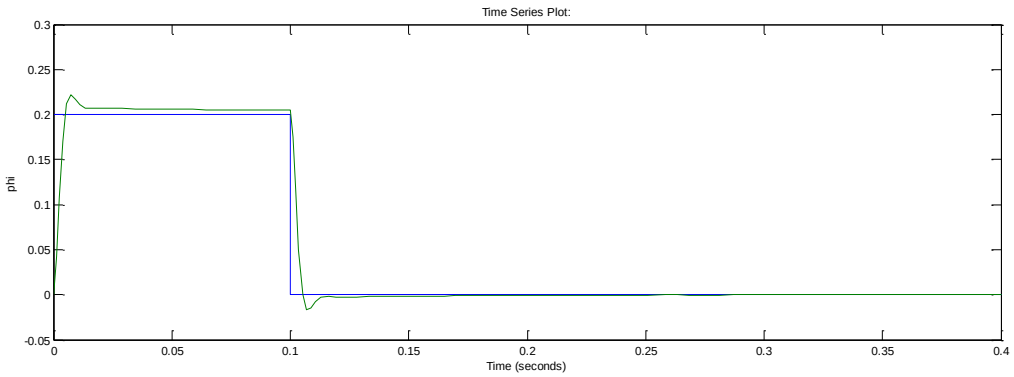


Figura 38: Respuesta controlador PID.

Como se ve en la respuesta anterior, el controlador es muy rápido y tiene muy poco sobre pico.

Este controlador se llegó a implementar en el sistema real pero no se obtuvieron buenos resultados, Se intentó sintonizar manualmente el PID con el sistema real, se probaron muchos valores para el controlador pero en ningún caso se consiguió un buen equilibrio para el robot. Dado que no se consigue el objetivo de mantener el robot en equilibrio, se descarta el controlador PID.

5.2. Péndulo invertido sobre dos ruedas.

El modelo matemático utilizado se puede encontrar en [3]. El desarrollo del modelo matemático se ha realizado teniendo en cuenta las leyes físicas a las que está sometido el péndulo invertido sobre dos ruedas. En la figura 39 se muestra el esquema del péndulo invertido sobre el que se ha basado el modelo, en ella se puede ver un péndulo sobre un carro, en nuestro caso la masa del carro es la masa de las ruedas. En la tabla 2 se pueden encontrar todos los parámetros del sistema con sus respectivas descripciones y unidades.

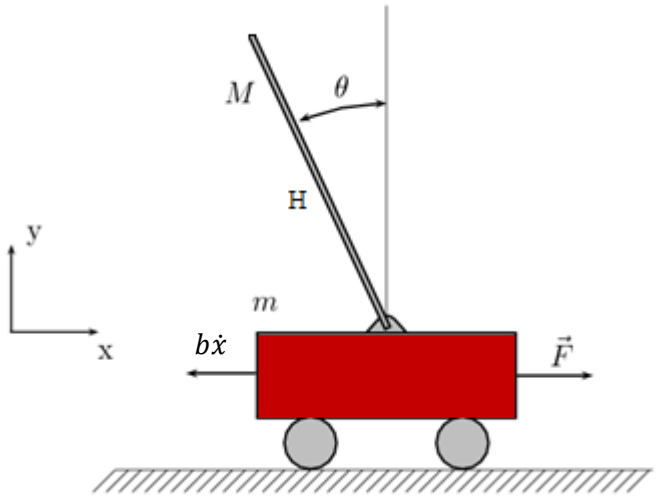


Figura 39: Esquema péndulo invertido.

g	Aceleración de la gravedad	9.81 m/s^2
M	Masa del cuerpo del péndulo	0.45 kg
m	Masa de las ruedas	0.03 kg
b	Fricción	0.1 N/m/s
H	Altura del cuerpo del péndulo	0.29 m
L	Distancia del centro de masas al eje de las ruedas	$\frac{H}{2} = 0.145\ m$
I	Inercia del péndulo	$\frac{ML^2}{3} kg\ m^2$
θ	Ángulo de inclinación del cuerpo del péndulo	
x	Posición del péndulo	
F	Fuerza aplicada	

Tabla 2: Parámetros físicos del sistema.

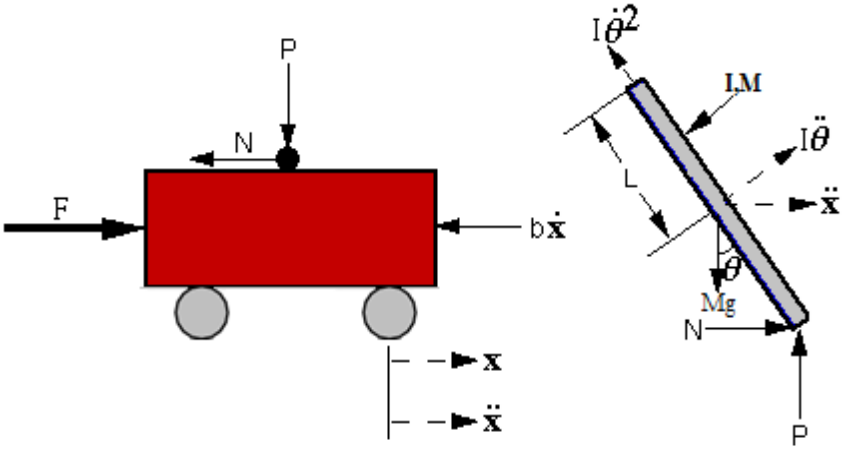


Figura 40: Análisis de las fuerzas del sistema

En la figura 40 se encuentra la descomposición de fuerzas del sistema, de aquí podemos extraer las ecuaciones del sistema.

La primera ecuación la obtenemos de la suma de fuerzas horizontales del carro, que en nuestro caso corresponde solo a las ruedas, es la siguiente:

$$m\ddot{x} + b\dot{x} + N = F \tag{5}$$

La segunda ecuación la obtenemos de la suma de fuerzas horizontales del péndulo, donde obtenemos la ecuación de la fuerza N:

$$N = M\ddot{x} + ML\ddot{\theta}cos\theta - ML\dot{\theta}^2 sin\theta \tag{6}$$

Si sustituimos la ecuación (6) en (5), obtenemos la primera ecuación del movimiento del sistema:

$$(M+m)\ddot{x} + b\dot{x} + ML\ddot{\theta}cos\theta - ML\dot{\theta}^2 sin\theta = F \tag{7}$$

La tercera ecuación se obtiene de la suma de fuerzas perpendiculares al péndulo

$$PLsin\theta + NLcos\theta - MgLsin\theta = ML\ddot{\theta} + ML\ddot{x}cos\theta \tag{8}$$

La suma de momentos del centro de masas del péndulo es la siguiente:

$$-PLsin\theta - NLcos\theta = I\ddot{\theta} \tag{9}$$

Combinando las ecuaciones (8) y (9) obtenemos la segunda ecuación dinámica del sistema:

$$(I + ML^2)\ddot{\theta} + MgLsin\theta = -ML\ddot{x}cos\theta \tag{10}$$

Las ecuaciones del sistema (7) y (10) son no lineales, dado que para que el robot se mantenga en equilibrio el ángulo θ se debe mantener cercano a cero, podemos linealizar las ecuaciones en este punto para facilitar el diseño del control.

Dado que tendremos pequeñas variaciones de θ, podemos asumir que:

$$sin\theta \approx \theta \tag{11}$$

$$cos\theta \approx 1 \tag{12}$$

$$\dot{\theta}^2 \approx 0 \tag{13}$$

Las ecuaciones linealizadas que se obtienen son las siguientes

$$(M+m)\ddot{x} + b\dot{x} + ML\ddot{\theta} = F \tag{14}$$

$$(I+ML^2)\ddot{\theta} - MgL\theta = -ML\ddot{x} \tag{15}$$

Despejando $\ddot{x}$ y $\ddot{\theta}$ obtenemos:

$$\ddot{x} = \frac{-ML}{(M+m)}\ddot{\theta} - \frac{b}{(M+m)}\dot{x} + \frac{1}{(M+m)}F \tag{16}$$

$$\ddot{\theta} = -\frac{ML}{(I+ML^2)}\ddot{x} + \frac{MgL}{(I+ML^2)}\theta \tag{17}$$

Sustituyendo (16) en (15) y (17) en (14) se obtienen las siguientes ecuaciones:

$$\ddot{\theta} = \frac{MgL(M+m)}{I(M+m)+MmL^2}\theta + \frac{MLb}{I(M+m)+MmL^2}\dot{x} - \frac{ML}{I(M+m)+MmL^2}F \tag{18}$$

$$\ddot{x} = -\frac{gM^2L^2}{I(M+m)+MmL^2}\theta - \frac{b(I+ML^2)}{I(M+m)+MmL^2}\dot{x} + \frac{I+ML^2}{I(M+m)+MmL^2}F \tag{19}$$

5.3. Espacio de estados del péndulo invertido.

Con las ecuaciones (18) y (19) se realizará el modelo en espacio de estados del sistema. Para ello se ha de tener en cuenta que la acción de control u=F, y las variables de estado son $x_1 = \theta, x_2 = \dot{\theta}, x_3 = x, x_4 = \dot{x}$; y las salidas son $y_1 = x, y_2 = \theta$.

Las cuatro ecuaciones de estado son las siguientes:

$$\dot{x_1} = x_2 \tag{20}$$

$$\dot{x_2} = \frac{MgL(M+m)}{I(M+m)+MmL^2}x_1 + \frac{MLb}{I(M+m)+MmL^2}x_4 - \frac{ML}{I(M+m)+MmL^2}u \tag{21}$$

$$\dot{x_3} = x_4 \tag{22}$$

$$\dot{x_4} = -\frac{gM^2L^2}{I(M+m)+MmL^2}x_1 - \frac{b(I+ML^2)}{I(M+m)+MmL^2}x_4 + \frac{I+ML^2}{I(M+m)+MmL^2}u \tag{23}$$

La representación general en espacio de estados es la siguiente:

$$x = Ax + Bu \tag{24}$$

$$y = Cx + Du \tag{25}$$

En nuestro caso las ecuaciones de estado del sistema son las siguientes:

$$\dot{x} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{MgL(M+m)}{I(M+m)+MmL^2} & 0 & 0 & \frac{MLb}{I(M+m)+MmL^2} \\ 0 & 0 & 0 & 1 \\ -\frac{gM^2L^2}{I(M+m)+MmL^2} & 0 & 0 & -\frac{b(I+ML^2)}{I(M+m)+MmL^2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{ML}{I(M+m)+MmL^2} \\ 0 \\ \frac{I+ML^2}{I(M+m)+MmL^2} \end{bmatrix} u \tag{26}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u \tag{27}$$

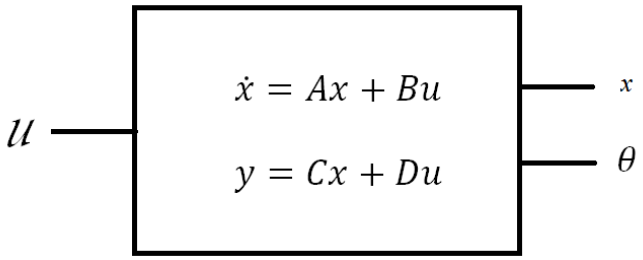


Figura 41: Entradas y salidas del modelo en espacio de estados

En la figura 41 podemos ver las entradas y salidas del modelo matemático en espacio de estados, en ella se encuentra la acción de control *u* como única entrada y las salidas son la posición y el ángulo de inclinación del péndulo invertido. La acción de control *u* será posteriormente la potencia que se le aplicará a los motores.

5.4. Simulación del sistema.

Para realizar la simulación del sistema se ha utilizado el programa Simulink, ya que resulta muy útil simulando sistemas dinámicos ante entradas como impulsos, escalón, rampas o condiciones iniciales no nulas. Sustituyendo los parámetros de la tabla 2 en la ecuación (26) se obtienen los siguientes valores:

$$\dot{x} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 170.745 & 0 & 0 & 3.629 \\ 0 & 0 & 0 & 1 \\ -23.229 & 0 & 0 & -0.7016 \end{bmatrix}}^{A} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \overbrace{\begin{bmatrix} 0 \\ -36.29 \\ 0 \\ 7.016 \end{bmatrix}}^{B} u \qquad (28)$$

$$y = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{C} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

Para plasmar las anteriores ecuaciones que se obtienen en espacio de estados, en diagrama de bloques, se utiliza el esquema de la figura 42, donde los valores para las matrices A, B y C se encuentran en la ecuación 28. Esta figura corresponde al modelo matemático sin control. Como forma de comprobación de que el modelo desarrollado en el apartado se asemeja al sistema real, se simula este sistema. La respuesta debería ser inestable ante una condición inicial diferente de cero, dado que no se está aplicando ninguna medida de control modelo, de igual manera que si el robot de dejará sin ninguna acción, el robot caería.

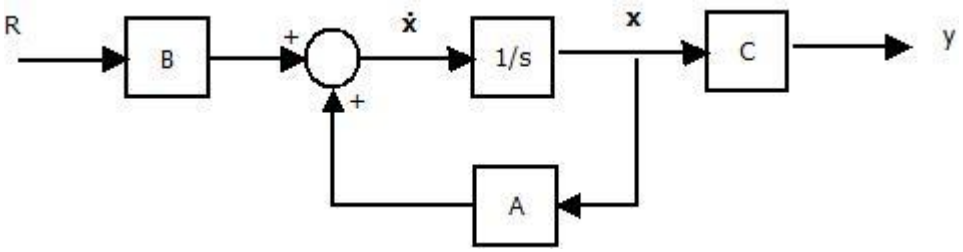


Figura 42: Diagrama de bloques de las ecuaciones en espacio de estados.

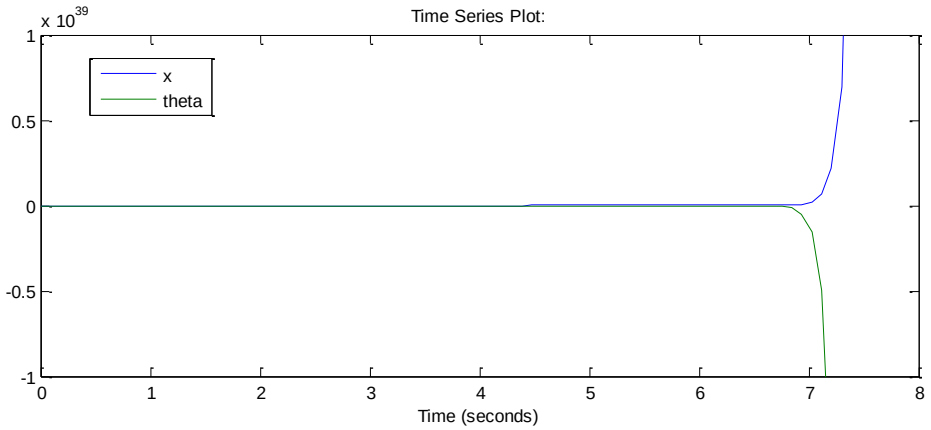


Figura 43: Respuesta del sistema sin control.

En la figura 43 se encuentra la respuesta del sistema sin acción de control ante unas condiciones iniciales de 0.2 rad, 11.46°, en ella se puede ver como ante estas condiciones iniciales el sistema se hace inestable, la posición tiende hacia el infinito y el ángulo hacia menos infinito. En el sistema real si se aplica un ángulo inicial sin ninguna acción de control el robot se caerá. Por lo que se comprueba que el modelo se acerca a la realidad del péndulo invertido.

Los polos del sistema son: 0, 12.826, -13.32 y -0.207. Es por el polo positivo 12.826 que se puede confirmar que el sistema es inestable.

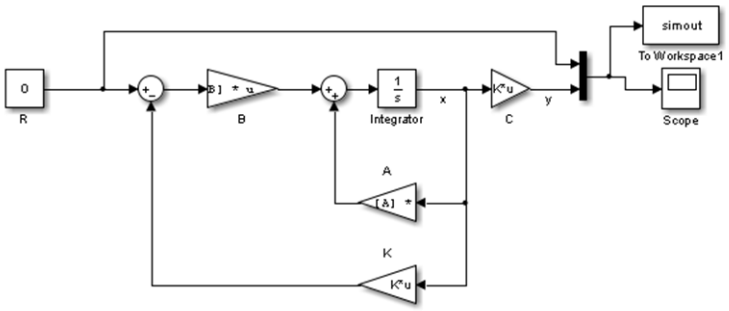


Figura 44: Diagrama de bloques en Simulink.

En la figura 44 se encuentra el diagrama de bloques en lazo cerrado del sistema, con el controlador.

Dado que lo que nos interesa es equilibrar el robot, nos centraremos en encontrar un controlador que estabilice el ángulo de inclinación del péndulo (Θ). El controlador K que se utilizará es un vector que consta de 4 valores, un valor por cada variable de estado, este tipo de control se denomina LQR (*Linear-Quadratic Regulator*).

5.4.1. Diseño control LQR.

Se le denomina control óptimo LQR, esta técnica forma parte de una de las ramas de control automático más importantes en el desarrollo de las estrategias modernas de control y es una de las más utilizadas hoy en día.

Con esta técnica se obtendrán los valores K_i para nuestro controlador, para ello se explicará brevemente en que consiste el control LQR.

Teniendo el sistema dinámico a controlar en espacio de estados, se determina una medida de la calidad del desempeño del sistema o criterio de desempeño, se le denomina J y viene determinada por la siguiente ecuación [4]:

$$J = \int_0^\infty [x^T(t)Qx(t) + u^T(t)Rx(t)]dt \tag{29}$$

Donde las matrices Q y R son matrices de peso reales, simétricas constantes y definidas positivas. Las Ecuaciones Q y R tienen la siguiente forma:

$$Q = \begin{bmatrix} q_{11} & 0 & \cdots & 0 \\ 0 & q_{22} & 0 & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & \ddots \end{bmatrix}, \; R = \begin{bmatrix} r_{11} & 0 & \cdots & 0 \\ 0 & r_{22} & 0 & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & \ddots \end{bmatrix} \tag{30}$$

El control óptimo resulta minimizando la ecuación (29) y viene dado por la ley lineal de realimentación de estado (31) donde K se define con la ecuación (32) y P es la única solución definida positiva de la matriz Ecuación Algebraica de Riccati (EAR) (33).

$$u(t) = -Kx(t) \tag{31}$$

$$K = R^{-1}B^TP \tag{32}$$

$$A^TP + PA - PBR^{-1}B^TP + Q = 0 \tag{33}$$

Para el diseño del controlador se ha recurrido a la función de Matlab *lqr*, lo que hace esta función es minimizar la ecuación (29) con los parámetros de entrada A, B, Q y R, devolviendo las matrices K y P. El parámetro P se utiliza para calcular el controlado K, la función *lqr* lo utiliza internamente, pero no es necesario que conozcamos su valor para obtener el controlador, aunque es posible conocerlo, las matrices A y B se han encontrado en los apartados anteriores, las matrices Q y R depende de las especificaciones que se requieran. El valor R frecuentemente tiene un valor igual a uno, y Q se irá variando hasta conseguir los resultados deseados. A continuación se muestra el código utilizado:

```
Q= [1 0 0 0; 0 0 0 0; 0 0 1 0; 0 0 0 0];

R=1;

K = lqr(A, B, Q, R);
```

Con lo que se obtiene el siguiente controlador:

```
K =  -11.2563   -1.0057   -1.0000   -1.2617
```

En la figura 45 se encuentra la respuesta al anterior controlador, en color rojo se puede ver la respuesta de theta ante una condición inicial de 0.2 rad (11.4°), y en color verde el valor consigna que es cero. La respuesta que se obtiene no es aceptable ya que el tiempo de establecimiento es excesivo, cerca de 1.1 s, y también el sobre pico es muy elevado; el robot requiere de un tiempo de respuesta menor de 0.5 s ya que si es superior el robot se caerá.

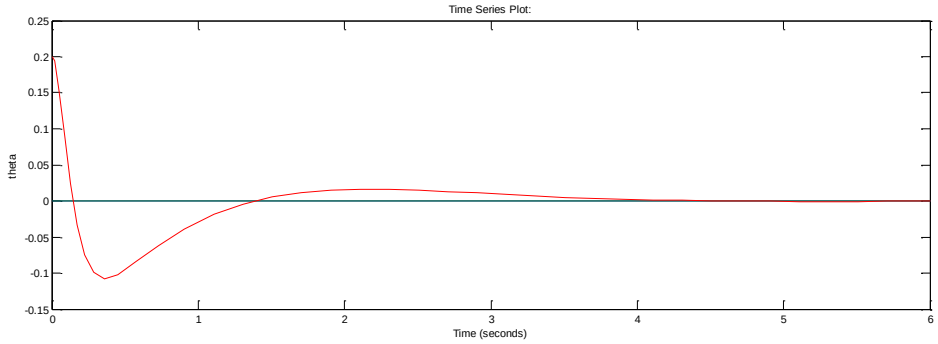


Figura 45: Respuesta controlador K= -11.25 -1.0057 -1 -1.26.

Con el objetivo de minimizar el tiempo de establecimiento y el sobre pico se cambia los valores de la matriz Q de tal manera que se obtienen diferentes valores para el controlador, como se ve en la figura 46. En esta figura se puede ver cómo se va disminuyendo tanto el sobre pico como el tiempo de establecimiento.

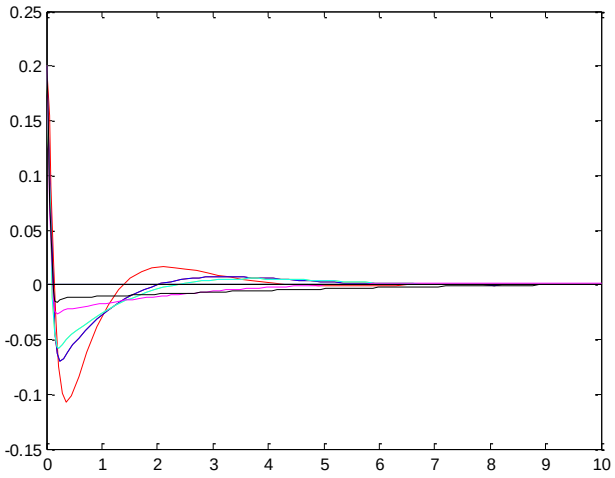


Figura 46: Respuesta para diferentes controladores.

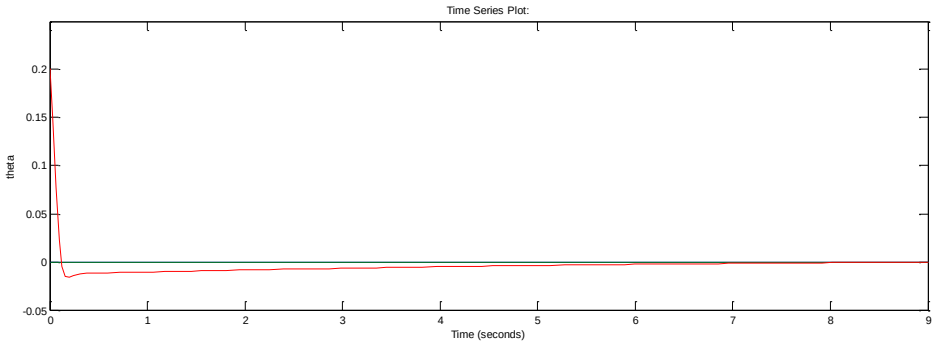


Figura 47: Respuesta controlador K= -25.6476 -1.3224 -0.1000 -0.7695.

En la figura 47 se encuentra la respuesta para el controlador con el que se obtuvieron mejores resultados, el controlador es **K= -25.6476 -1.3224 -0.1000 -0.7695**, con él se obtiene un tiempo de establecimiento al 2%, menor de 0.5 s, y un sobre pico de 0.015 rad (0.86°). Debido a que estas características son las buscadas, se opta por este controlador.

A modo de comprobación de la estabilidad del sistema se calculan los polos del sistema en lazo cerrado. Polos del sistema:

-21.4884 +16.8502i

-21.4884 -16.8502i

-0.1580 + 0.1505i

-0.1580 - 0.1505i

Todos los polos son negativos, por lo que se puede confirmar que el sistema es estable.

6. SEGUIMIENTO DE TRAYECTORIA.

El sensor que se utiliza para el seguimiento de trayectorias es el sensor de color de Hitechnic, en este apartado se explicará el método que se ha utilizado.

En la figura 48 se encuentra un esquema de la vista superior del robot con la posición de las ruedas y del sensor de color pintado en rojo. El objetivo es encontrar una acción de control que indique el giro que han de dar las ruedas para seguir el contorno izquierdo de la línea pintada en el suelo.

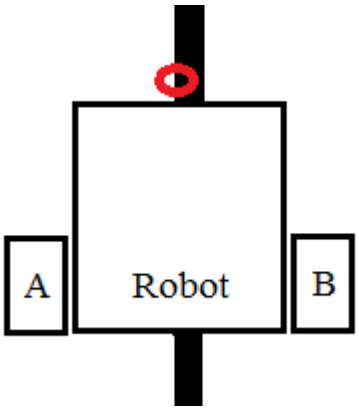


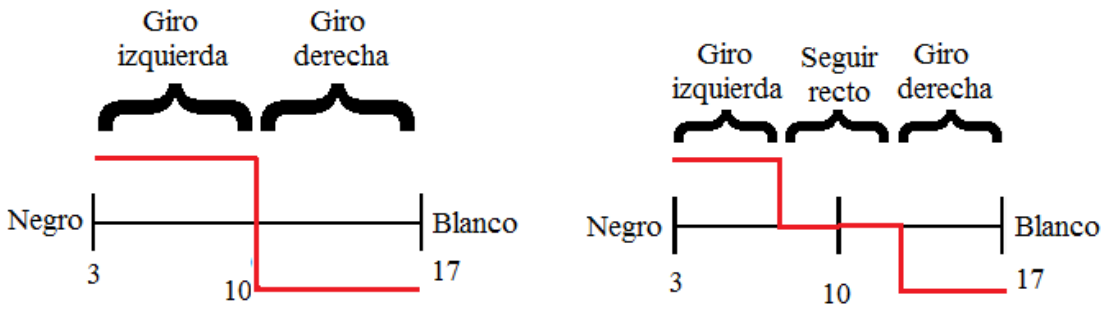
Figura 48: Esquemas robot + seguimiento de trayectoria.

Para mayor facilidad se ha decidido poner el suelo de color blanco con una línea negra que es la que indica la trayectoria a seguir. Los seguidores de línea se suelen realizar con dos configuraciones: uno o dos sensores de luz o color, con la configuración de dos sensores el objetivo es que la línea se mantenga entre los dos sensores, y con un sensor el objetivo es que siga uno de los bordes de la línea, también se puede implementar con más de dos sensores. En general cuanto más sensores mejor seguimiento se la línea, ya que con un solo sensor se puede seguir la línea de manera precisa, el robot se ha dotado con un sensor.

La causa por la que se sigue el borde de la línea es que de esta manera se sabe que cuando el sensor se encuentra en una zona de color blanco ha de girar a la derecha, y cuando se encuentra en una zona de color negro el robot ha de girar a la izquierda, si se sigue el borde izquierdo de la línea. De esta manera cuando el robot se encuentre justo en el borde de la línea seguirá recto.

Los dos colores que se tienen en cuenta a la hora de seguir la línea son blanco y negro, cuando el sensor se encuentra delante de una superficie de color blanco devuelve un valor de 17 y cuando es de color negro un valor de 3. En la figura 49 a) se pueden ver los giros

que debe dar el robot en función del color de la superficie donde se encuentre, el valor medio entre blanco y negro que debería ser el valor del borde de la línea será de 10. Cuando el valor que devuelve el sensor sea mayor que 10 el robot debe girar a la derecha ya que se encontrará en una superficie blanca, y cuando es menor de 10 girar a la izquierda porque se encontrará en la línea negra. Con la figura 49 a) se obtendrá un seguidor de líneas que realizará la trayectoria en zigzag, con la aproximación de la figura 49 b) se ve un rango cercano al borde de la línea en el que el robot seguirá recto, en este caso el robot realizará tres acciones: girar a la derecha, a la izquierda o seguir recto.



a) B)

Figura 49: Giros del robot según el color.

Con el modelo anterior se conseguirá que el robot de giros bruscos, ya que se quiere un seguimiento suave de la línea se aproximan los giros a una línea, como se puede ver en la figura 50, en este caso en un eje se encuentra en error y en el otro el giro. De esta manera se consigue dar giros brucos cuando se está lejos del borde y giros suaves cuando se encuentra cerca.

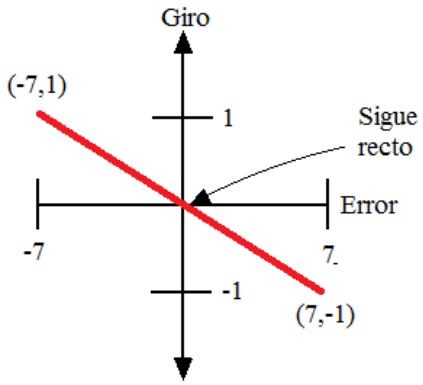


Figura 50: Aproximación lineal seguimiento de trayectorias.

La ecuación de la recta del modelo de la figura 50 se puede se puede conocer de la siguiente manera:

$$y = mx + b \tag{34}$$

El eje *y* corresponde con el giro y *x* con el error, dado que la recta pasa por el centro *b* es igual a cero. El valor de la pendiente *m* es -0.14 que se puede encontrar con los dos puntos conocidos. La ecuación queda de la siguiente manera:

$$Giro = -0.14\ error\ ,\ \ Giro = K\ error \tag{35}$$

De la ecuación de la recta anterior podemos obtener un control proporcional, donde K corresponde con la constante proporcional K_p.

Para obtener el valor *error* se debe obtener el valor actual del sensor de color, a este valor se le ha de restar un *offset* para obtenerlo. El *offset* viene determinado por la distancia entre el color blanco o negro al valor medio entre ellos. Es decir, si blanco es 17, negro es 3 y su valor medio es 10, entonces el *offset* será 7, la distancia de 3 a 10 o de 17 a 10.

Un aspecto que se debe tener en cuenta a la hora de ejecutar el programa es que al sensor de color le afecta la luminosidad ambiental, por lo que los valores que se obtienen para un color podrían variar si cambian las condiciones de luminosidad.

Por lo que antes de ejecutar el programa se debe comprobar el valor que se obtiene del sensor ante los colores blancos y negros, y si se obtienen valores diferentes a los valores anteriores, se debe calcular de nuevo el offset de color. Se ha creado un programa para obtener y guardar los valores de color, a continuación se muestra el programa:

```
byte fileHandle;        // Inicialización de las variables necesarias para guardar los datos
short bytesWritten;
float actual_position;          // Inicialización de la variable que contendrá el color

task main ()
{
SetSensorLowspeed(IN_3);            // Inicialización del sensor de color
Off(OUT_AB);                        // Se paran los motores
```

```
    DeleteFile("Datos.txt");                // se crea un archivo donde se guardarán los
datos
    CreateFile("Datos.txt", 50000, fileHandle);
    while(true)
    {
    actual_position=SensorHTColorNum (IN_3);        // Se obtiene el valor de color del
sensor
    NumOut(0, LCD_LINE1, actual_position);          // se muestra por la pantalla del
brick

    string col = NumToStr(actual_position);         // Se guardan los valores en un
                                                    // documento de texto.
    string write = StrCat(col);
    WriteLnString(fileHandle,write, bytesWritten);
    Wait(10);
    }
    }
```

6.1. Controlador PID.

Para obtener mejores resultados en el seguimiento de la trayectoria se utilizará un controlador PID. En este apartado se dará una breve descripción de este controlador.

Para diseñar el control de trayectoria no se dispone de modelo matemático, por lo que no podemos utilizar un control LQR, como en el control de equilibrio. En este caso se ha optado por un controlador PID, ya que se puede sintonizar manualmente relativamente fácil.

Se trata de un control proporcional-integral-derivativo, es una técnica muy común utilizada para una amplia variedad de robots, vehículos, maquinaria, etc. Un controlador PID calcula el error, que es la diferencia entre el valor medido y el valor deseado o *setpoint*, el objetivo del controlador es minimizar este error ajustando una variable manipulable.

La fórmula del controlador es la siguiente:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \quad (36)$$

Donde:

K_p: Ganancia proporcional

K_i: Ganancia integral

K_d: Ganancia derivativa

e: Error: diferencia entre valor actual y *setpoint*

Los valores K_p, K_i y K_d son los valores a sintonizar. La parte proporcional es el producto entre el error y la constante proporcional K_p, con esta parte se intenta que en estado estacionario el error sea aproximadamente cero. En muchos casos la parte proporcional no es suficiente para eliminar el error en estacionario, por lo que es necesario una parte integral. La parte integral es el producto entre la integral del error y la constante integral K_i, que disminuye y elimina el error en estado estacionario. La acción derivativa consiste en multiplicar la derivada del error por la contante derivativa K_d, esta acción evita que el error se incremente corrigiéndolo con la misma velocidad que se produce.

En la figura 51 se puede ver el diagrama de bloques del controlador PID del proyecto actual, donde la planta del sistema es el robot, la acción de control o salida del controlador PID es el giro de las ruedas, y la salida la obtenemos del sensor de color.

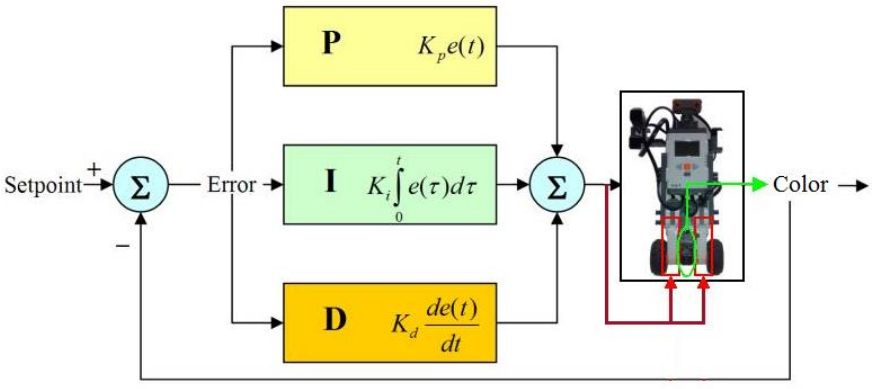


Figura 51: Diagrama de bloques PID.

Existen diferentes técnicas de sintonización de un PID, pero generalmente requieren de un modelo de la planta. Dado que no se dispone del modelo se ha optado por la sintonización manual de las constantes del controlador, para ello se han seguido los siguientes pasos:

- Se inicializan los valores K_i y K_d a cero.
- Se incrementa K_p hasta tener oscilaciones constantes, con lo que se encuentra el valor crítico, se fija K_p a la mitad de este valor.
- Se incrementa K_i para reducir el error en estacionario, teniendo en cuenta que un valor muy elevado de esta constante puede provocar inestabilidad.
- Se incrementa K_d hasta obtener un lazo lo suficientemente rápido ante interferencias.

Para la sintonización manual también se han tenido en cuenta los efectos de los parámetros PID de la tabla 3.

Efectos de los parámetros				
Parámetro	**Tiempo de subida**	**Sobre pico**	**Tiempo de establecimiento**	**Error en permanente**
K_p	Reduce	Aumenta	Poco cambio	Reduce
K_i	Reduce	Aumenta	Aumenta	Elimina
K_d	Indefinido	Reduce	Reduce	Nada

Tabla 3: Efecto de los parámetros PID.

Teniendo en cuanta los aspectos anteriores se obtuvo el siguiente controlador:

$$\boldsymbol{K_p = 0.9,\ K_i = 1.1\ y\ K_d = 0.001} \tag{37}$$

7. DESARROLLO DEL PROGRAMA.

7.1. Entorno de programación.

Existen diferentes software con los que se puede programar el *Brick* NXT 2.0, en este apartado se enumerará algunos de ellos.

- Lego Mindstorms NXT: Lego junto con *National Instruments* han creado este software, es un programa muy intuitivo de programación por bloques.

Figura 52: Software Lego mindstorms NXT.

- *Matlab* y *Labview*: estos programas disponen de diversas *toolbox* con los que se puede programar el NXT 2.0.

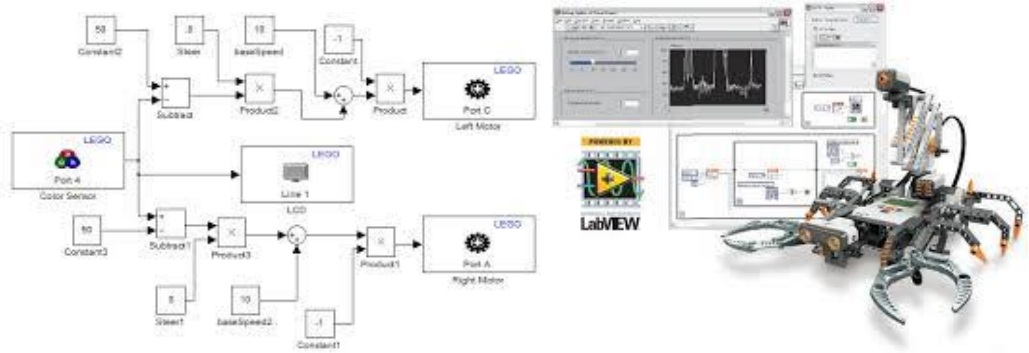


Figura 53: Software Matlab y LabView.

- Bricx Command Center: Su lenguaje de programación es similar al lenguaje C, NXC *not exactly* C, este programa es el seleccionado para la realización del programa de este proyecto. Se ha seleccionado este programa debido a que este entorno

ofrece funciones implementadas para la lectura de los diferentes sensores y actuadores, también dispone de visualización de datos a tiempo real. Se ha seleccionado este software debido a los conocimientos previos en programación C.

Bricx Command Center es un programa con el que se pueden programar tareas que se ejecuten paralelamente

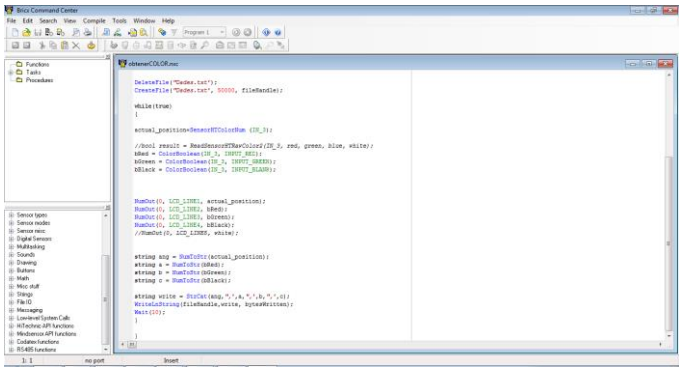


Figura 54: Software Bricx Command Center.

- Existen otros programas con lenguajes similares a C como RobotC.

7.2. Programa desarrollado.

En este apartado se explicara el programa desarrollado y los aspectos que se han tenido en cuenta. El programa se introduce en el *Brick* del robot y éste mediante los valores que obtiene de los sensores, envía a los motores la acción de control que corresponda en cada momento, para mantener el robot en equilibrio y le capacita para el seguimiento de la trayectoria. El programa al completo se puede encontrar en la Anexo A de este documento.

Para la realización del programa se ha tomado como ejemplo el programa de hitechnic, que se puede encontrar en su web [5].

7.2.1. Derivas del sensor giroscópico.

Uno de los aspectos muy importantes a tener en cuenta son las derivas que tiene el sensor Gyro, por su funcionamiento este sensor lleva intrínsecas unas derivas que pueden depender de diversos factores tales como la fabricación del propio sensor, el nivel de voltaje de alimentación del *Brick* o a las vibraciones a las que se somete durante su funcionamiento.

Dado que es imprescindible obtener un dato fidedigno de este sensor es necesario contrarrestar estas derivas. Para ello se calcula el *offset* que tiene el sensor, este valor se calcula al inicio del programa y se va actualizando durante la ejecución del programa, ya que va variando. El *offset* es el valor que facilita del sensor cuando su velocidad angular es cero.

Para calcular el offset inicial se posiciona el robot en una posición estable sin vibraciones ni movimiento, se toman 100 medidas y se calcula el valor medio de estas medidas. Para ello se ha utilizado el siguiente código:

```
for (i=0; i<100; i++) {
    Gyro_Valor = SensorHTGyro(GYRO);
    Gyro_Suma += Gyro_Valor;
    Wait(5);
}
Gyro_Offset = Gyro_Suma / 100;
```

En el código anterior se puede ver un bucle *for* que se ejecuta 100 veces, en cada ejecución se adquiere un valor del sensor Gyro y se suma al anterior, ya que el sensor proporciona datos aproximadamente cada 3,33 ms se esperan 5 ms antes de solicitar la siguiente medida. Al finalizar el bucle se divide la suma de los valores por el número de muestras y se obtiene el valor del *offset* inicial.

Como se ha dicho el *offset* va variando durante la ejecución del programa, por lo que se realiza una actualización de su valor en cada iteración, para ello se actualiza en función de la última lectura obtenida del sensor y del *offset* anterior, dando mayor peso al valor anterior del *offset* como se ve en el siguiente fragmento de código:

```
Gyro_Actual = SensorHTGyro(GYRO);     // se obtiene el valor actual del sensor
Gyro_Offset = 0.0005 * Gyro_Actual + (1-0.0005) * Gyro_Offset;   //actualización offset
```

Se han comprobado diferentes técnicas y valores para los pesos anteriores, para la actualización del *offset*, pero se obtuvieron mejores resultados con el código anterior.

7.2.2. Acciones a realizar durante el recorrido.

Uno de los objetivos del proyecto es el de realizar acciones durante el recorrido, las acciones propuestas son parar el robot si se encuentra con algún objeto delante o parar si se encuentra con algún color específico, que no sean color blanco y negro, durante el seguimiento de la línea.

Mediante el sensor ultrasonido se ha conseguido que el robot tenga la capacidad de detectar objetos que se encuentran en su campo de visión. En la ejecución de las pruebas se ha colocado la mano delante del sensor ultrasonido. Como se verá más adelante, el código desarrollado contempla la acción de parar ante la detección de un objeto en un rango seleccionado. El rango se puede configurar de la manera que el usuario desee, teniendo en cuenta que el rango de visión del sensor es de 4 a 255 cm y que tiene una precisión de +/- 3 cm.

El objeto no tiene que encontrarse necesariamente justo delante del sensor, ya que el sensor tiene visión lateral, en la figura 55 se puede ver el rango del sensor ultrasonido desde una vista superior, en la imagen se puede ver un objeto en color rojo a unos 10°. El cono de visión del sensor que está pintado en color gris contempla este ángulo, el sensor detectaría el objeto. Como se ve en la imagen cuanto mayor es el ángulo menor precisión se tendrá. Por lo que no se recomienda posicionar el objeto a más de +/- 10°.

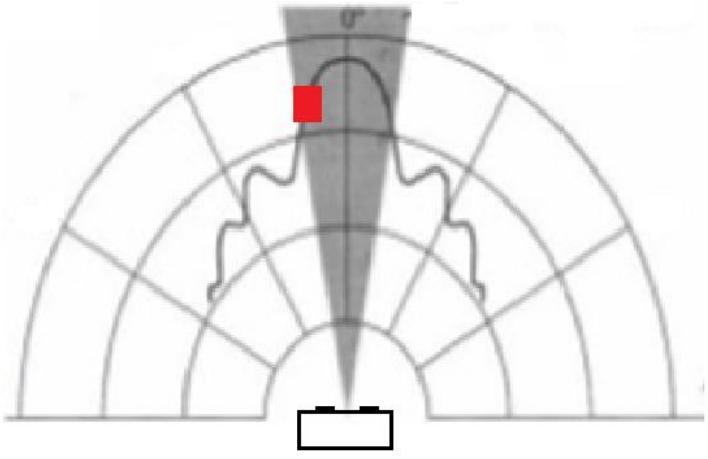


Figura 55: Rango sensor ultrasonido.

Dado que el robot utiliza el sensor de color para el seguimiento de la trayectoria, se ha considerado utilizar este sensor para detectar un color diferente al blanco o negro, ya que se utilizan para el seguimiento de la trayectoria. La acción a realizar sería la de parar cuando se detecta un color establecido, pero se han tenido algunos problemas. El problema es que los valores que se obtiene para un color suele variar aunque el robot no se mueva, en la figura 56 se puede ver un ejemplo de los valores que proporciona el sensor ante un color amarillo, como se puede ver los valores que proporciona el sensor no son constantes. La figura corresponde a la recogida de datos durante 7 segundos aproximadamente, el problema es que en muchos momentos se obtienen valores que corresponden al blanco y al negro (17 y 3). Esta misma situación se tiene con todos los colores comprobados, incluidos el blanco y el negro. En la figura 57 se puede ver los valores que se obtienen ante un color blanco.

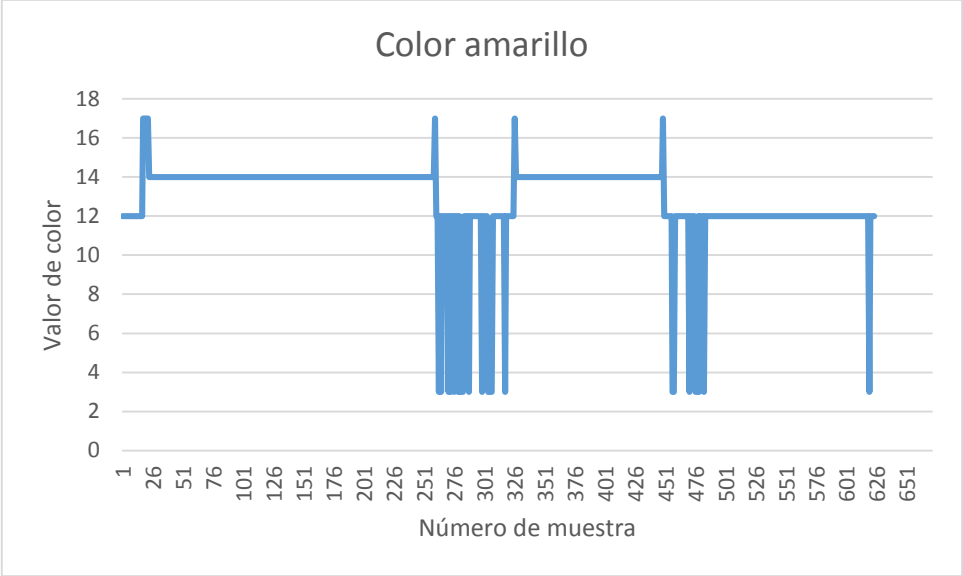


Figura 56: Valores de color para el color amarillo.

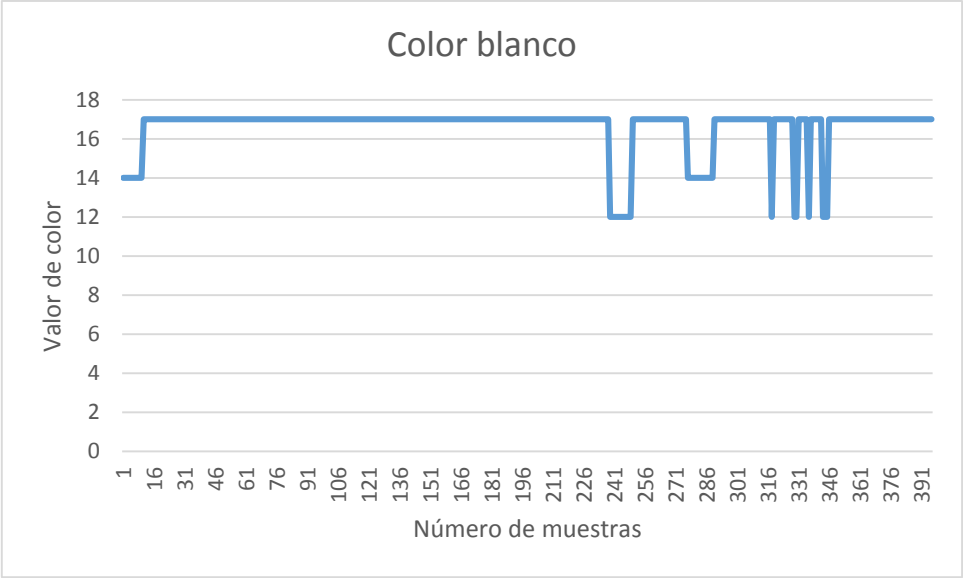


Figura 57: Valores de color para el color blanco.

Esta misma situación se repite con todos los colores probados: rojo, verde, azul y magenta; con estos últimos colores se obtienen valores muy cercanos al negro (3), por lo que se confunden con él. Esta situación crea confusión, ya que hace que en el caso de que se quiera que el robot pare ante un color amarillo, se ha de introducir en el programa que cuando encuentre un valor entre 12 y 14, significa que tiene un color amarillo delante y por lo tanto debe parar; a lo largo de la ejecución del programa aunque no se encuentre con este color, el sensor obtiene repetidamente estos valores, lo que hace que el robot se pare muchas veces, esto genera inestabilidad y hace que el robot se desvíe de la línea negra y por lo tanto se "pierde".

Se realizaron muchas pruebas con diferentes colores pero con el mismo resultado, el robot acaba por perder la línea negra y por lo tanto no es capaz de realizar el seguimiento de la trayectoria.

Aunque sería posible desarrollar filtros con los que sería posible solucionar este problema, esto supondría realizar un estudio con diferentes filtros, implementarlos y comprobarlos, en definitiva supondría tiempo; ya que en realidad la acción fundamental es la de parar el robot y ya se ha conseguido con el sensor ultrasonido, se ha decidido no introducir como acción a realizar durante el recorrido que el robot se pare ante un color específico.

7.2.3. Estructura del programa.

El programa se divide en tareas o subrutinas para mejor estructuración y comprensión del mismo, de las cuales algunas se ejecutan simultáneamente. En este apartado se describirá las tareas que se utilizan en el programa así como las variables y constantes que necesitan.

En la figura 58 se encuentra el diagrama de bloques con las subrutinas del programa. Como se puede ver se inicializan las variables y constantes globales, seguidamente se ejecuta la subrutina *main,* dentro de esta tarea se llama a la tarea *offset* que es la encargada de calcular el *offset* inicial. Una vez se ejecuta esta tarea la subrutina *main* finaliza, a continuación se ejecutan tres tareas simultáneamente que son *Equilibrio*, *ParaOadelante* y *Ultrasonido*. Dentro de la tarea Equilibrio se llaman a las tareas *CalcIntervalo, DatosGyro, DatosMotor* y *ControlTrayectoria*.

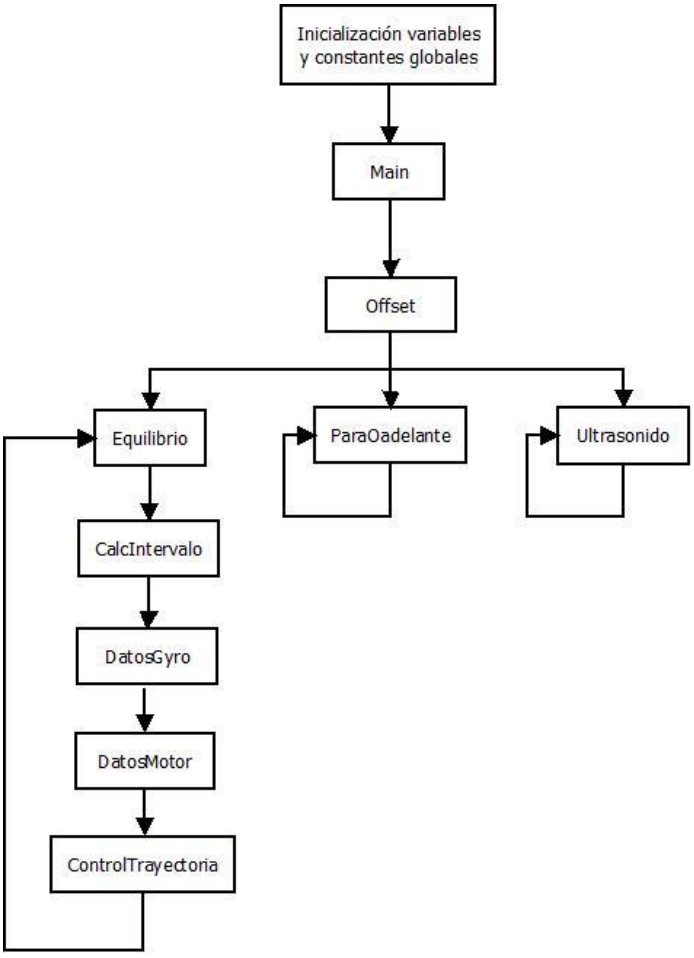


Figura 58: Diagrama de bloques del programa.

A continuación se describirá las variables y constantes globales del programa y cada una de las subrutinas que se encuentran en el programa.

7.2.3.1. Variables y constantes globales.

En las tablas 4 y 5 se encuentra cada una de las constantes y variables globales que se utilizan en el programa junto con una breve descripción.

Constantes	**Descripción**
KGYROANG 26.5	K1 constante control LQR para equilibrio
KGYROVEL 1.56	K2 constante control LQR para equilibrio
KPOS 0.15	K3 constante control LQR para equilibrio
KVEL 0.13	K4 constante control LQR para equilibrio
Kp 0.9	Kp constante control PID para seguimiento
Ki 1.1	Ki constante control PID para seguimiento
Kd 0.001	Kd constante control PID para seguimiento
CONTROL_VEL 600.0	Velocidad máxima para los motores

Tabla 4: Constantes del programa.

Variables	
Sum_motores = 0	Suma velocidad, indica si se desea parar o ir hacia delante.
tCalcStart = 0	Primer periodo de ejecución del programa
tInterval = 0	Periodo de ejecución actual
Gyro_Offset	Offset del sensor giroscópico actual
Gyro_Angle_Global = 0	Ángulo del robot actual
actual_position	Dato sensor de color actual
actual_position_old	Dato sensor de color anterior
proportional	Parte proporcional del control PID
integral	Parte integral del control PID
derivativo	Parte derivativa del control PID
previous_error_color=0.0	Error anterior de color
error_color=0.0	Error actual de color
offset_color=7.5	Offset de color
output	Acción de control del control PID
Motor_Posicion = 0	Posición actual de los motores
Motor_Rot_Sum = 0	Suma actual de la rotación de los motores
Motor_Rot_SumPrev = 0	Suma anterior de la rotación de los motores
Motor_Rot_D3 = 0	Delta 3 de rotación para cálculo de velocidad de los motores
Motor_Rot_D2 = 0	Delta 2 de rotación para cálculo de velocidad de los motores
Motor_Rot_D1 = 0	Delta 1 de rotación para cálculo de velocidad de los motores
Motor_Rot_D = 0	Delta 0 de rotación para cálculo de velocidad de los motores
Motor_Rot_Izq = 0	Rotaciones del motor izquierdo
Motor_Rot_Der = 0	Rotaciones del motor derecho
power	Señal de control del control LQR de equilibrio
powerLeft	Señal de control introducida al motor izquierdo
powerRight	Señal de control introducida al motor derecho
LEFT = 0	Cantidad de giro a la derecha
RIGHT = 0	Cantidad de giro a la izquierda
distancia = 0	Distancia obtenida del sensor de ultrasonido
detectado_obs=0	indica si el sensor de ultrasonido detecta un objeto

Tabla 5: Variables del programa.

7.2.3.2. Subrutinas del programa.

En este apartado se explicará la función de cada una de las subrutinas que se encuentran en el programa.

***Main*:**

Al iniciar el programa se debe poner el robot en posición estable, ya que se debe calcular el offset del sensor giroscópico. La tarea *main* es siempre la primera en ejecutarse, es aquí donde se inicializan los sensores y se llama a la función *Offset.* Una vez se ejecuta *Offset* vuelve a la función *main* e indica con dos pitidos separados por tres segundos que se debe posicionar el robot en vertical para iniciar el recorrido. Al finalizar el *main* se ejecutan tres funciones simultáneamente: *Equilibrio*, *ParaOadelante* y *Ultrasonido*.

Offset:

Como se ha explicado en el apartado 7.2.1. de este documento es necesario calcular el offset inicial del sensor giroscópico, es en esta función donde se calcula.

Equilibrio:

Es en esta función donde se calcula la acción de control que se debe dar a los motores para mantener el robot en equilibrio y que realice el seguimiento de la línea. En esta función se inicializan las variables locales siguientes:

gyroSpeed, velocidad angular del robot.

gyroAngle, ángulo del robot.

Motor_Velocidad, velocidad del motor

cLoop, indica el número de veces que se ha ejecuta el bucle infinito de esta función.

Después se inicializan los contadores de rotación de los motores, es un dato que se utilizará después. A continuación se inicia un bucle infinito donde se llama a la función *CalIntervalo*, esta función actualizará el valor del periodo que tarda cada iteración del bucle; a continuación se ejecuta la función *DatosGyro*, donde se actualizará el valor para la velocidad angular y el ángulo del robot; seguidamente se ejecuta la función *DatosMotor*, donde se actualizan los valores de velocidad y posición del motor. A continuación para indicar si se desea ir hacia delante o parar se le añade a la posición del motor la variable Sum_motores, esta variable se actualiza en la función ParaOadelante.

Seguidamente se ejecuta el controlador LQR que calculará la acción de control que mantendrá el robot en equilibrio.

Se probaron inicialmente los valores del controlador que se obtuvieron con la simulación del modelo matemático, y para mejor funcionamiento del equilibrio se cambió el signo y sus valores ligeramente, a continuación se ven los valores iniciales y los utilizados.

Controlador inicial: K= -25.65 -1.32 -0.1 -0.7695

Controlador utilizado: K= 26.5 1.56 0.15 0.13

Se observa que ha sido necesario un cambio de signo del controlador, esto podría ser debido a las suposiciones que se realizan en la linealización del modelo matemático, se recuerda las suposiciones a continuación:

$$sin\theta \approx \theta \tag{11}$$

$$cos\theta \approx 1 \tag{12}$$

$$\dot{\theta}^2 \approx 0 \tag{13}$$

Como se ha conocido en el modelo matemático [6], estas suposiciones también podrían haberse tomado como:

$$sin\theta = \sin(\pi + \phi) \approx -\phi \tag{38}$$

$$cos\theta = \cos(\pi + \phi) \approx -1 \tag{39}$$

$$\dot{\theta}^2 = \dot{\phi}^2 \approx 0 \tag{40}$$

Donde ϕ corresponde a una pequeña variación de θ, como se puede ver sólo los signos cambiarían de haber hecho esta suposición en la linealización desarrollada para este trabajo, esta podría ser la causa de la necesidad del cambio de signo en el sistema real.

Según el modelo matemático los cuatro valores del controlador multiplican respectivamente al ángulo, velocidad angular, posición y velocidad del péndulo; para la implementación del controlador se ha aproximado la posición y velocidad del péndulo por la posición y velocidad de los motores. Por lo tanto el controlador queda de la siguiente manera:

```
power = KGYROANG * gyroAngle+
        KGYROVEL * gyroSpeed +
        KPOS     * Motor_Posicion +
        KVEL     * Motor_Velocidad;
```

Una vez tenemos la acción de control que mantendrá el robot en equilibrio se le añade la acción de control que hace que hace posible el seguimiento de la línea. Para ello se ejecuta la función *ControlTrayectoria*.

Finalmente en la función *Equilibrio* se transmite la acción de control a los motores.

CalIntervalo:

Esta función lo que hace es aproximar el tiempo que tarda en ejecutarse cada iteración del bucle de la función *Equilibrio*. Este valor se utiliza para el cálculo del ángulo del péndulo, velocidad del motor y en el controlador PID. Este periodo se calcula a partir del tiempo transcurrido desde que se empieza el bucle dividido por el número de veces que se ha ejecutado el bucle. Para el primer valor se utiliza un valor aproximado. Seguidamente se encuentra el código de la función:

```
// La primera iteración se pone un valor fijo como intervalo
 if (cLoop == 0) {
   tInterval = 0.0055;
   tCalcStart = CurrentTick();
 } else {

// Cálculo del intervalo a partir del valor medio.
   tInterval = (CurrentTick() - tCalcStart)/(cLoop*1000);
 }
```

DatosGyro:

En esta función se actualizan los valores del *offset* del sensor giroscópico, velocidad angular y ángulo del robot. Para ello se obtiene el valor del sensor giroscópico, se actualiza el valor del *offset* como se explicó en el apartado 7.2.1. la velocidad angular viene dada por el valor del sensor menos el *offset* actual. En ángulo se obtiene con el sumatorio de la multiplicación la velocidad angular por el valor del periodo actual, a continuación se encuentra el fragmento del código con el que se obtiene lo anterior:

```
Gyro_Actual = SensorHTGyro(GYRO);
Gyro_Offset = 0.0005 * Gyro_Actual + (1-0.0005) * Gyro_Offset;
gyroSpeed = Gyro_Actual - Gyro_Offset ;
Gyro_Angle_Global += gyroSpeed*tInterval;
gyroAngle = Gyro_Angle_Global;
```

***DatosMotor*:**

De esta función se obtiene la velocidad y posición del motor. Se suman las rotaciones de los dos motores, con este valor y el valor de la suma anterior se obtiene la primera delta, este valor se va acumulando y se obtiene la posición del motor. La velocidad del motor se obtiene de dividir el valor medio de las últimas cuatro deltas por el periodo actual.

A continuación se encuentra el fragmento de código de esta función:

```
// Datos rotación de los motores.
 Motor_Rot_Izq = MotorRotationCount(LEFT_MOTOR);
 Motor_Rot_Der = MotorRotationCount(RIGHT_MOTOR);

 // Se calcula la suma de la rotación de los motores.
 Motor_Rot_SumPrev = Motor_Rot_Sum;
 Motor_Rot_Sum = Motor_Rot_Izq + Motor_Rot_Der;

 //Calculo de Motor_Posicion a partir de la suma de la rotación de los motores.
 Motor_Rot_D = Motor_Rot_Sum - Motor_Rot_SumPrev;
 Motor_Posicion += Motor_Rot_D;

 // Se calcula Motor_Velocidad a partir del promedio de las ultimas 4 deltas.

Motor_Velocidad=(Motor_Rot_D+Motor_Rot_D1+Motor_Rot_D2+Motor_Rot_D3)/
                        (4*tInterval);
```

```
// Se guardan los valores de las deltas.

Motor_Rot_D3 = Motor_Rot_D2;

Motor_Rot_D2 = Motor_Rot_D1;

Motor_Rot_D1 = Motor_Rot_D;
```

ControlTrayectoria:

Lo primero que ejecuta esta función es actualizar el valor de color que se obtiene del sensor, a continuación se le resta el offset de color para obtener el valor del error de color. La parte proporcional del controlador se calcula multiplicando el error por el término Kp, la parte integral se calcula multiplicando el término Ki por la parte integral del error y la integral con la multiplicación de Kd por la parte derivativa.

La discretización de la parte integral se calcula haciendo el sumatorio del error multiplicado por el periodo o dt, la parte derivativa se calcula restando el valor de error actual y el anterior dividido por dt. La acción de control se obtiene sumando los tres términos.

A continuación se encuentra el fragmento de código correspondiente al controlador:

```
proportional = Kp * error_color;

integral = integral + error_color*tInterval;

derivativo = (error_color - previous_error_color)/ tInterval;

output = proportional + Ki * integral + Kd * derivativo;

previous_error_color = error_color;
```

Una vez se obtiene la acción de giro se aplica a la acción de equilibrio, de la siguiente manera:

```
powerLeft = power - output;

powerRight = power + output;
```

power es la acción de control de equilibrio y output es el giro.

El valor que se aplica a los motores queda de la manera anterior sino se ha detectado un obstáculo delante del robot, en caso de encontrar un obstáculo queda de la siguiente manera:

```
powerLeft = power;

powerRight = power
```

Ya que cuando se detecta un obstáculo sólo se quiere que se mantenga en equilibrio y no gire.

Por último en esta función se aplica una limitación de potencia, ya que los motores tienen una limitación de +/- 100. Por ello si la acción de control resultante es mayor de +100 o menor que -100 se limita a +100 o -100 respectivamente.

ParaOadelante:

Esta función se ejecuta después del *main*, en ella se realizan dos acciones, o bien se dirige el robot hacia delante o se para. Para ello se actualiza el valor de la variable Sum_motores en función si se encuentra o no un objeto delante del robot. Las dos variables LEFT y RIGHT se encargan de dirigir el robot, si se encuentra un objeto delante del robot estas variables serán igual a cero, para parar el robot. Por el contrario si no hay objeto se quiere que el robot siga hacia delante, entonces toman valor de -50. Con estos valores también se regula la velocidad del robot, si aumentamos sus valores en valor absoluto, la velocidad aumenta y viceversa; y toman un valor negativo cuando se quiere ir hacia delante. A continuación se ve el fragmento de código que se ejecuta cuando no hay objeto delante.

```
if (detectado_obs==0){

    // Sigue si no se detecta objeto

    LEFT = -50;

    RIGHT = -50;

    do

    {

    Sum_motores = (LEFT + RIGHT) * CONTROL_VEL / 200.0;

    }while (detectado_obs==0)

}
```

Si hay objeto el valor de Sum_motores es cero.

Ultrasonido:

Es en esta función que se actualiza la variable que indica si hay objeto delante del robot o no. Es aquí donde se obtiene el valor del sensor ultrasonido, si el valor devuelto se encuentra a una distancia comprendida entre 4 y 40 cm, la variable que indica si hay objeto: *detectado_obs* tomará un valor igual a uno, de lo contrario se iguala a cero.

El rango de detección del objeto es configurable, se ha seleccionado el rango de medida de 4 a 40 cm. El rango de medida del sensor es de 4 a 255 cm pero se considera que 40cm son suficientes. A continuación se encuentra el código que realiza la función *Ultrasonido*.

```
while (true)
{
    //lectura del sensor ultrasonido
    distancia = SensorUS(IN_2);
    // Si se detecta un objeto a un distancia entre 4 y 40cm detectado_obs= 1
    if (distancia > 4)
    {
     if (distancia < 40){detectado_obs= 1; }
     }
     if (distancia < 4)  {
     detectado_obs= 0;}
     if (distancia > 40)  {
     detectado_obs= 0;}
    }
```

8. PRUEBAS CON EL SISTEMA REAL.

En este apartado se explicarán las diferentes pruebas que se han realizado, así como como los aspectos a tener en cuenta de las trayectorias y el manual de usuario.

8.1. Trayectorias.

En cuanto a los aspectos a tener en cuenta para la realización de las trayectorias se detallarán en este apartado. Se debe de disponer de una superficie de color blanco con una línea negra que marca la trayectoria a seguir, la línea negra debe ser de 1.5 a 2 cm de ancho y puede contener rectas y curvas, se recomienda que no contenga esquinas. En la figura 59 se puede ver un ejemplo de trayectoria.

Figura 59: Ejemplo de trayectoria.

En el caso de que la trayectoria obtenga esquinas el robot debería dar un giro bastante brusco comparado con lo que tiene que dar con las curvas, para ello debe circular a muy baja velocidad, y aun en este caso llegaría a perder la trayectoria. Es por esta razón que se recomienda que la trayectoria con contenga esquinas.

El robot para realizar la trayectoria necesita “ver” el color del suelo, para ello hace uso del sensor de color V2, este sensor necesita estar inclinado con respecto a la superficie de la que se quiere conocer el color, en la figura 60 se tiene la posición correcta, esta posición se mantiene mientras la trayectoria no tenga pendientes. A la hora de subir o bajar pendientes la inclinación del sensor con respecto a la superficie cambia, como se puede ver en las figuras 61 y 62. Esto supone que se llegue a obtener valores inválidos y por lo tanto el robot se llega a “perder”. Esta condición limita la pendiente que el robot es capaz de subir sin “perderse”. Para solucionarlo se ha intentado cambiar de posición el sensor

sin resultados positivos, ya que se llegaba a “perder” incluso en superficies sin inclinación. La pendiente de subida que podía soportar sin perderse es de 10° y 4° de bajada.

Figura 60: Esquema del robot sin pendiente.

Figura 61: Esquema del robot bajando pendientes.

Figura 62: Esquema del robot subiendo pendientes.

8.2. Manual de usuario.

En este apartado se explicarán los pasos y aspectos a tener en cuenta a la hora de hacer uso de este programa.

Una vez se dispone del kit Lego Mindstorms NXT 2.0, para la construcción del robot se deben de seguir los pasos indicados en el Anexo C de este documento, teniendo en cuenta la posición y conexión de los sensores que se indicará a continuación con ayuda de la imagen 63, los sensores se deben conectar en la parte inferior del Brick, y los actuadores en la parte superior.

- 1. Sensor ultrasonido: se debe conectar el la entrada 2 del *Brick*.
- 2. Sensor giroscópico, Gyro de Hitechnic: se debe conectar el la entrada 4 del *Brick*.
- 3. Sensor brújula: este sensor en realidad no se utiliza en este proyecto, se ha incorporado para futuras mejoras. Por lo que su conexión no es necesaria.
- 4. Sensor acelerómetro: como el sensor anterior, este tampoco se utiliza.
- 5. Sensor de color V2 de Hitechnic: conexión en la entrada 3 del *Brick*.
- 6. Actuador: el motor derecho se conecta en la salida A.
- 7. Actuador: el motor izquierdo se conecta en la salida B.

Figura 63: Sensores y actuadores robot.

El programa con el que se ha programado es Bricx Command Center, para disponer del programa se puede acceder su web [7]. Una vez se tiene el BricxCC instalado se puede cargar al Brick, con ayuda del cable USB, el programa desarrollado en este proyecto, para ello se puede recurrir a la ayuda que dispone el programa *bricxCC*.

Para el seguimiento de trayectoria es necesario tener un *offset* de color, puesto que los valores que se obtienen del sensor de color puede variar con diferentes condiciones de luminosidad, de debe comprobar qué valor tiene el offset, para ello se puede recurrir al apartado 6 de este documento.

Una vez se comprueba el valor del offset actual de color, se procede a la ejecución del programa cargado anteriormente; para ello se busca en el Brick el programa, antes de ejecutar el programa de debe poner el robot en una posición estable y sin vibraciones, la mejor posición es horizontal sobre el suelo o mesa. Seguidamente se puede ejecutar el programa con la opción RUN, a continuación aparecerá en la pantalla la palabra "Calibrando", seguidamente se escuchara un pitido, lo cual indica que ha finalizado la calibración, tres segundos después se escuchará otro pitido; en esos tres segundos se ha de posicionar el robot en vertical, cuando se escucha el segundo pitido se debe soltar el robot para que empiece el seguimiento de la trayectoria en equilibrio. El momento de soltar el robot para que empiece el equilibrio es un momento crítico, ya que se debe

colocar lo más vertical posible para que el robot consiga el equilibrio. En algunos casos en que el usuario no deje totalmente vertical el robot, es posible que inicie el equilibrio desplazándose hacia atrás y oscile para conseguir afrontar el ángulo inicial con el que se ha dejado, generalmente tarda poco en estabilizarse.

En cuanto a la trayectoria se puede ver un ejemplo en la figura 59. La trayectoria debe tener el suelo de color blanco y con una línea negra que marcará la trayectoria. La línea debe ser de 1,5 a 2 cm de grosor aproximadamente, puede incluir rectas y curvas, y no esquinas. Si el circuito contiene esquinas podría perder la trayectoria en este punto. Aunque es posible disminuir la velocidad a la que circula el robot al mínimo, y en algunos casos consigue seguir una esquina, aunque por su bajo índice de aciertos no se recomienda. En el apartado 7.2.3.2. se indica las variables a cambiar en el caso de querer cambiar la velocidad.

La trayectoria también puede incluir pendientes de subida y bajada. Las pendientes se han de incluir en un tramo de recto de la trayectoria y un ángulo máximo de 10° para la pendiente subida y 4° para la bajada.

Durante la trayectoria se puede situar un obstáculo delante del sensor ultrasonido, para las pruebas se ha utilizado la mano del usuario, seguidamente el robot se parará. El obstáculo solo podrá estar delante durante un periodo corto de tiempo sin llegar a "perderse".

El robot también permite pequeños empujones y consigue mantener la estabilidad.

El programa se ejecuta infinitamente, si se desea parar basta con coger el robot y pulsar el botón central del *Brick*.

8.3. Resultados de las pruebas.

Se han realizado muchas pruebas con el robot. Se debe destacar que para realizar las pruebas en el momento de soltar el robot e iniciar el equilibrio se debe colocar el robot lo más vertical posible, si esta condición no se cumple es posible que el robot se caiga sin conseguir el equilibrio inicial.

Las pruebas más significativas son las siguientes:

Prueba 1:

Esta prueba se ha realizado con el objetivo de obtener las variaciones que experimenta el ángulo del robot. En esta prueba no se ha realizado seguimiento de la línea, el robot se mantiene el equilibrio durante la prueba. En la figura 64 se puede ver una gráfica los

valores del ángulo durante la duración de esta prueba, el inicio de la gráfica corresponde con el momento crítico en que el usuario suelta el robot, en este momento es frecuente que se deje con cierto ángulo y aceleración, como se puede ver hay una gran variación del ángulo, en este momento el controlador necesita aportar gran potencia a los motores, lo que le hace oscilar.

Para conseguir el equilibrio las acciones de control hacen oscilar el robot hasta conseguirlo, una vez se encuentra en equilibrio las oscilaciones son mínimas.

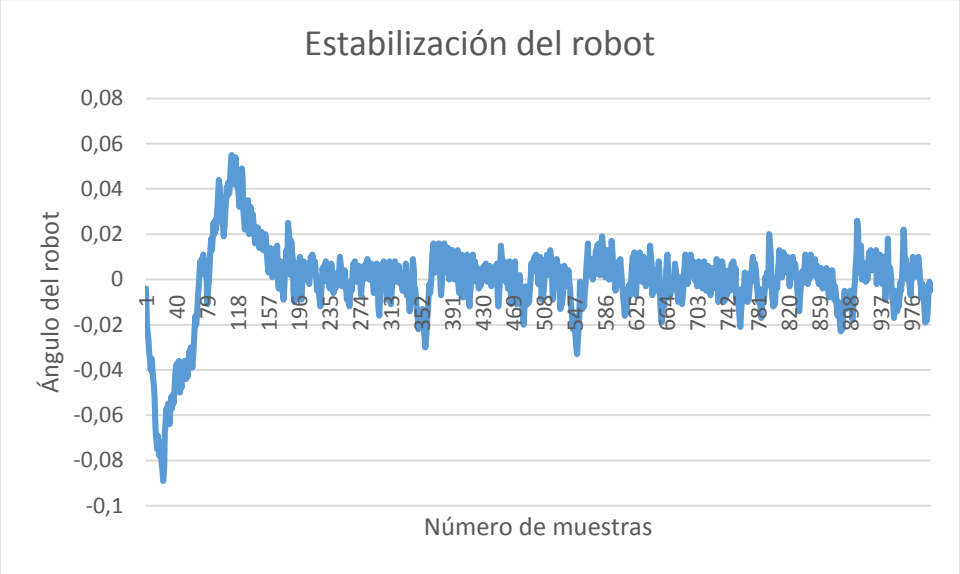


Figura 64: Valores del ángulo del robot, prueba 1.

Prueba 2:

Esta prueba consiste en una vuelta al circuito de la figura 59. En la figura 62 se puede ver los valores para las variables powerRight y powerLeft en forma de gráfica, estas dos variables son la potencia que se da a los motores derecho e izquierdo respectivamente.

Para una mejor comprensión de la gráfica se debe saber que los valores positivos de potencia suponen desplazamientos de la rueda hacia atrás, y valores negativos supone desplazamientos hacia delante. La gráfica se ha dividido por tramos:

- Tramo 1: corresponde al inicio del equilibrio, el momento crítico en el que el usuario suelta el robot, como al inicio de la prueba 1 el robot debe conseguir el equilibrio a partir de unas condiciones iniciales con las que el usuario deja el

robot. Para ello se desplaza hacia atrás y seguidamente hacia delante, como se ve en la gráfica.

- Tramos 2: corresponde a un tramo recto de la trayectoria, como se ve los valores para las dos variables son muy parecidos ya que los giros que se realizan son mínimos, los que ha de realizar el robot para hacer el seguimiento de una recta.
- Tramo 3: concierne a los diferentes giros del circuito, es por esto que robot debe dar mayores giros y por lo tanto los valores de las variables se distancian.
- Tramo 4: después de las curvas se encuentra otro tramo recto.
- Tramo 5: en este tramo se le ha colocado brevemente un obstáculo al robot, por lo que se para un momento y por lo tanto los valores de las variables son muy parecidos.
- Después de este tramo se encuentra otro giro.

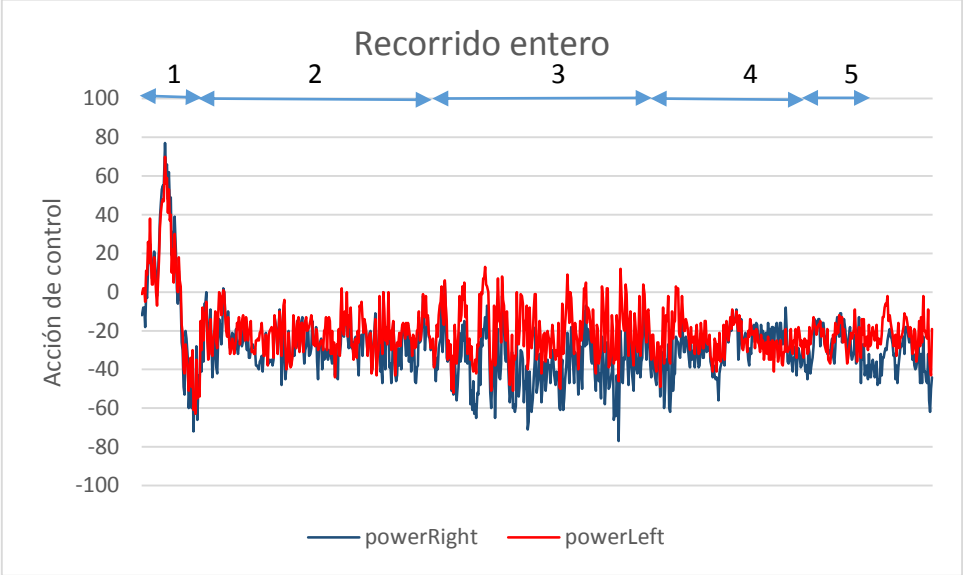


Figura 65: Potencia de los motores prueba 2.

Para el tramo 5 se puso la mano delante del robot para que se parará durante un momento. La situación de paro no puede prolongarse durante mucho tiempo puesto que en realidad el robot no puede parar del todo el movimiento ya que perdería el equilibrio, si la situación de paro se mantiene este movimiento puede hacer que pierda la trayectoria y por lo tanto llegue a "perderse". Por lo tanto la situación de paro sólo se puede mantener durante un par de segundos.

Prueba 3:

Esta prueba consiste a un circuito con un desnivel de subida. Este circuito incluye tramos rectos y curvas. El la figura 66 se puede ver los valores de las variables powerRight y powerLeft, la gráfica se ha dividido en zonas de colores para identificar los diferentes tramos. Los tramos de color verde se encuentran los tramos rectos, en rojo las curvas y en amarillo la cuesta de subida.

Los tramos rectos y con curvas se comportan igual que en la prueba anterior, en el tramo que corresponde a la curva como se puede ver se obtienen valores similares para las dos variables, puesto que continúa recto. Los picos que se ven en este tramo corresponden al inicio de la cuesta, donde el robot realiza mayor acción de control para estabilizarse debido a un pequeño escalón que tiene la cuesta, una vez estabilizado sube la cuesta.

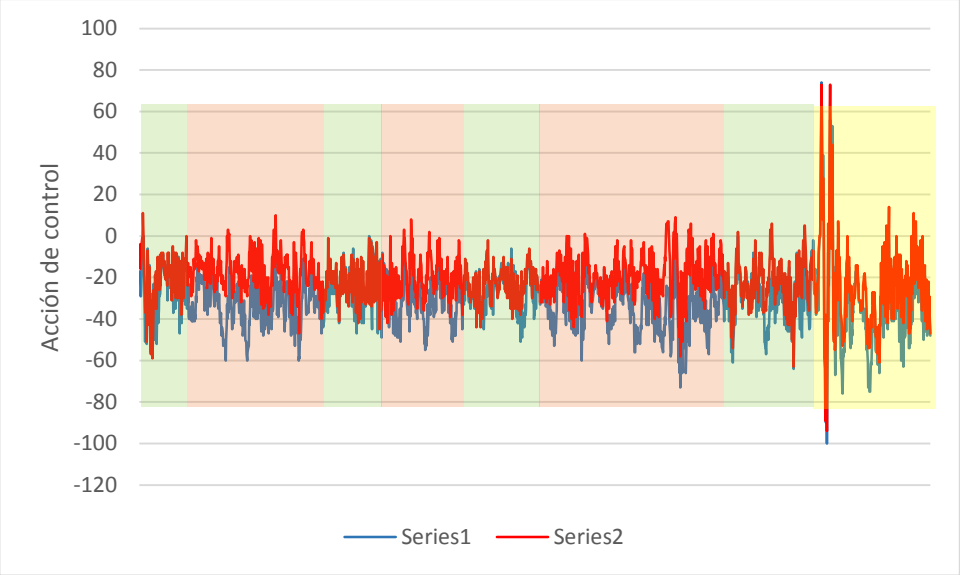


Figura 66: Potencia de los motores prueba 3.

Prueba 4:

En esta prueba se realiza el recorrido con desniveles de subida y bajada, consiste en realizar el recorrido subiendo y bajando dos veces por la misma cuesta. Los datos recogidos de esta prueba se reflejan en la figura 67, en ella se puede ver que los tramos marcados en color amarillo corresponden a las pendientes de subida y bajada. La primera vez que empieza a subir la cuesta se puede ver bastante ruido u oscilaciones, esto significa que le cuesta un poco mantener el equilibrio en ese momento debido al escalón de la cuesta, el final de la cuesta también se obtiene un poco de inestabilidad debido al escalón del final de la cuesta. La segunda vez que sube y baja la misma cuesta no tiene tanta

dificultad. Lo importante es que el robot no llega a perder el equilibrio en ningún momento.

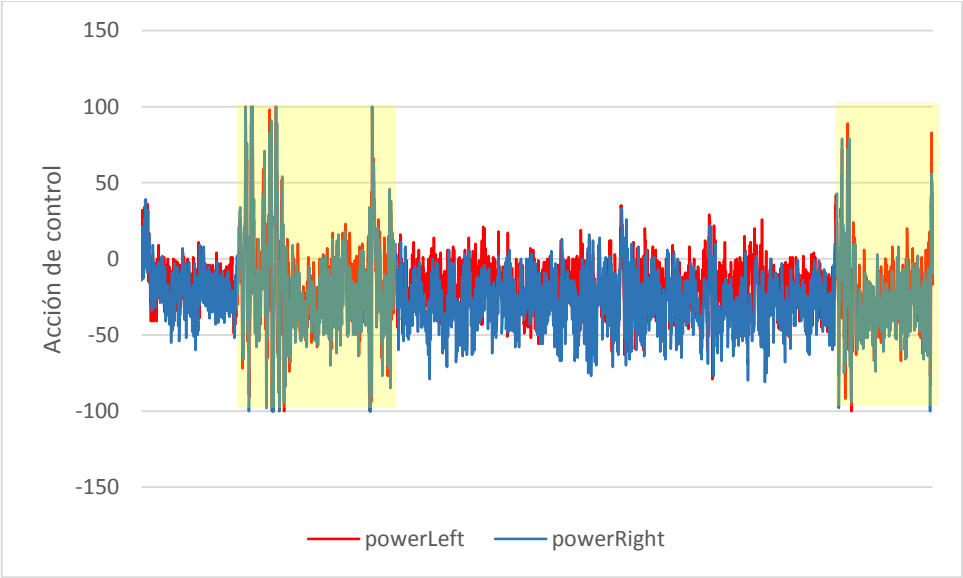


Figura 67: Potencia de los motores prueba 4.

9. PRESUPUESTO Y ANALISIS DE MERCADO.

En este apartado se encuentra el presupuesto del proyecto, en la tabla 6 se encuentra el presupuesto para el primer prototipo, en él se han incluido las horas de trabajo realizadas y el material necesario para el primer ejemplar, para los siguientes el precio disminuiría considerablemente, como se puede ver en La tabla 7.

PRESUPUESTO		
Descripción	**Horas**	**Precio**
Investigación	400	8.000 €
Diseño	300	6.000 €
Programación y pruebas	450	9.000 €
Material	**Unidad**	
Kit lego Mindstorms NXT 2.0	1	450 €
Sensor Gyro Hitechnic	1	54,95 €
Sensor color V2 Hitechnic	1	54,95 €
Sensor acelerómetro Hitechnic	1	54,95 €
Sensor brújula Hitechnic	1	54,95 €
Ordenador	1	1.000 €
	Total:	24.670 €

Tabla 6: Presupuesto primer ejemplar.

PRESUPUESTO			
	Horas	**Precio**	
Montaje programación y pruebas.	5	150	
	Unidad		**Descuento**
Kit lego Mindstorms NXT 2.0	1	315 €	30%
Sensor Gyro Hitechnic	1	49,50 €	10%
Sensor color V2 Hitechnic	1	49,50 €	10%
Sensor acelerómetro Hitechnic	1	49,50 €	10%
Sensor brújula Hitechnic	1	49,50 €	10%
		513 €	
	Total:	663 €	

Tabla 7: Presupuesto por ejemplar.

En el caso de que se quisiera vender el robot de manera profesional, la mejor manera de venderlos sería vía web, para ello se ha de añadir 1.500 € al presupuesto del primer ejemplar, en total se tendía una suma de 26.170 €. Dado que el trabajo necesario para el primer ejemplar ya se ha realizado, la puesta en marcha del proyecto podría darse en un periodo de tiempo corto. Otro gasto a tener en cuenta es la publicidad que se le dará al producto, para atraer a los clientes, esto supondrá publicación en revistas, convenciones tecnológicas y publicidad en páginas web especializadas. La publicidad puede suponer una gran cantidad de dinero, para esta actividad se adjudicará 7.000 €. Para ellos será necesario un préstamo bancario.

Si se vendiera cada ejemplar por 700 €, y suponiendo que mensualmente se tengan unos 20 pedidos se obtendrían unas ganancias de 3.690 € como se puede ver en la tabla 8.

Ventas	14.000 €
Mantenimiento web	-50 €
Material	-10.260 €
Ganancias	3.690 €

Tabla 8: Ventas y ganancias.

Si se adjudica un sueldo de 1.500 €, para amortizar el proyecto se obtendrían 2.190 € mensuales. Con esta cantidad mensual se amortizarían los gastos del primer ejemplar y el préstamo bancario en un año y tres meses. Este periodo de amortización es mínimo, suponiendo una venta fija de 20 ejemplares al mes.

En cuanto al análisis de mercado se utiliza un análisis DAFO donde se enumeran las fortalezas, debilidades, oportunidades y amenazas del proyecto. En la tabla 9 se encuentra el análisis DAFO.

Análisis	Fortalezas	Debilidades
Interno	Personal capacitado El proyecto ya está desarrollado Tiempo corto de puesta en marcha Poca inversión inicial	Poca experiencia en el sector Reputación inexistente
	Oportunidades	**Amenazas**
Externo	Interés en la robótica Entorno educativo Poca competencia directa	Escasa demanda Dependencia de los productos Lego y Hitechnic

Tabla 9: Análisis DAFO.

10. CONCLUSIONES Y OBSERVACIONES.

Para cada uno de los objetivos marcados para este proyecto se comentaran las conclusiones.

- Construir con los elementos del kit Lego Mindstorms NXT 2.0 un robot del tipo *Segway*, este objetivo se realizó satisfactoriamente, ya que se consiguió construir el robot.
- Desarrollar e implementar algoritmos de control que permitan mantener el robot en equilibrio durante todo el recorrido. Se consiguió implementar un controlador capaz de mantener el robot en equilibrio. Con todas las pruebas realizadas se consiguió que en el 100 % de los casos el robot se mantuviera en equilibrio durante todo el recorrido.
- Desarrollar e implementar algoritmos de control que permitan al robot seguir una trayectoria desconocida. Gracias a un controlador PID se consiguió que el robot pudiera seguir una línea que le indica la trayectoria.
- La trayectoria que ha seguir estará pintada en el suelo y comprenderá rectas, curvas, desniveles con pendientes de subida y bajada. En el 95 % de las pruebas que se realizaron el robot consigue realizar un seguimiento satisfactorio de rectas y curvas. En cuanto a los desniveles, se ha conseguido que pueda subir pendientes de hasta 10° y de bajada 4°, se ha conseguido que el 90 % de las veces el robot suba y baje las pendientes.
- Las acciones a realizar podrán ser la de parar de seguir la trayectoria durante un momento sin perder el equilibrio si se encuentra con un objeto delante o parar si encuentra en la trayectoria un color específico. Se ha conseguido que el robot pare ante un objeto durante un par de segundos sin perder la trayectoria, en el 100 % de la pruebas el robot ha parado, aunque en algunos casos llegaba a perder la trayectoria pero nunca el equilibrio. En cuanto a la acción a realizar de parar ante un color fijado no se ha podido conseguir debido a la inestabilidad de los datos que se obtienen del sensor de color utilizado.
- El robot debe soportar perturbaciones externas moderadas sin perder el equilibrio. El controlador desarrollado también permite pequeños empujones al robot sin perder el equilibrio, estas perturbaciones pueden suponer hasta unos 15° aproximadamente sin que el robot llegue a perder el equilibrio.

Este proyecto ha supuesto llevar a la práctica diversas técnicas aprendidas en el máter, como conocimientos de control, simulación e implementación.

10.1. Problemas encontrados y aspectos a destacar.

En este apartado se explicará los problemas más destacados encontrados en la realización de este trabajo, así como las soluciones a éstos, aspectos a destacar y los hechos que se han descartado. A continuación se describen en el orden en que han ido apareciendo:

- Cabe destacar que existe mucha información sobre el péndulo invertido y por lo tanto diversos modelos matemáticos, en este punto el problema es la elección de un modelo matemático fidedigno del sistema real. Como se ha dicho anteriormente en este documento, se llegaron a implementar diferentes modelos matemáticos obteniendo diferentes controladores.

  En este punto del trabajo aún no se disponía del sistema real, es decir del robot, para realizar directamente las pruebas e ir descartando los controladores; por esta razón se implementaron los programas correspondientes con los diferentes controladores, para probarlos todos con el robot. Se llegaron a tener muchos valores para los diferentes controladores. Esto supuso una gran cantidad de trabajo. En el momento que se tuvo disposición del robot, se empezaron a descartar los controladores, uno a uno.
- El caso del controlador *Fuzzy* explicado en el apartado 5.1 no se consiguió implementar para el sistema real, dado que el programa *BricxCC* con el que se desarrollaron los programas, no disponía de funciones para este tipo de controladores. Debido a ésta dificultad y a que en la simulación no se obtuvieron buenos resultados para el tiempo de establecimiento, se descartó la realización de un controlador *Fuzzy*.

  Cabe destacar que los modelos matemáticos no se acercan 100% a la realidad, eso supone que aunque en la simulación se obtengan resultados prometedores no significa que en el sistema real funciones igual de bien; pero resulta muy útil ya que supone unos valores con los que partir. Es decir, a partir de los controladores encontrados y realizando pequeñas variaciones se puede conseguir lo deseado. Para el controlador seleccionado fue imprescindible la simulación realizada, ya que es a partir de los valores encontrados que se llega a los que realmente se utilizaron.
- Se tuvieron muchos problemas a la hora de encontrar un controlador con el que se obtuviera un buen equilibrio para el robot. Para cada controlador encontrado en simulación se variaban los valores ligeramente con el objetivo de mejorarlos y obtener un buen equilibrio.

  Un buen equilibrio se le ha considerado a aquel con el que el robot pueda soportar el momento crítico en el que el usuario suelta el robot, con pocas oscilaciones;

que mantiene el equilibrio con perturbaciones moderadas externas como pequeños escalones o empujones y que permita subir y bajar cuestas sin caer. Esto es lo que se busca con el controlador.

En este punto del trabajo también se invirtieron muchas horas, ya que las condiciones deseadas para un buen controlador, como se ha explicado, son muy exigentes.

La solución a estos problemas llego con el controlador desarrollado en el apartado 5.4.1., que aportaba una simulación prometedora, y fue a partir del controlador encontrado que se llegó a los valores finalmente utilizados.

- Muchos controladores no conseguían salvar el momento crítico inicial, este momento supone que el usuario puede no dejar el robot totalmente vertical y por lo tanto el controlador parte con un ángulo y también con aceleración inicial. El controlador ha de ser capaz de estabilizar este momento, deben de ser agresivos ante estas condiciones sin provocar suficientes oscilaciones que hagan que el robot se desestabilice.

  La solución a este problema la traían Los valores de los controladores que conseguían aportar una potencia bastante alta a los motores en ese momento y por lo tanto el robot realizaba oscilaciones en este punto, estas oscilaciones también pueden llegar a hacer inestable el sistema y puede llegar a caer. Con el controlador seleccionado es con el que se obtuvieron menores oscilaciones, como se ha visto en la pruebas necesita realizar una acción de control alta y oscilaciones, el número de oscilaciones que necesita para solventar este punto depende de las condiciones iniciales.
- En cuanto al seguimiento de trayectoria dado que no se disponía de un modelo matemático, no se disponía de unos valores con los que iniciar el controlador. Por ello se seleccionó un controlados PID, ya que se conoce su funcionamiento; y se pudo conseguir unos valores con los que se obtenía un buen seguimiento.

  Se consiguió que con este controlador que el robot siguiera en algunos casos esquinas en el circuito, pero sólo ante una velocidad de circulación muy baja, dado a su bajo número de aciertos en estos casos y a que no es un objetivo marcado en el trabajo, se descarta que el sensor pueda hacer seguimiento de esquinas en el circuito.
- La acción de parar o de aportar una velocidad hacia delante, se pudo solventar fácilmente incluyendo esto en la acción de control de equilibrio, como se ha visto anteriormente.
- La acción de parar ante un objeto se pudo integrar fácilmente gracias al sensor ultrasónico, y su fácil utilización.

- La acción de parar ante un color seleccionado que se encuentre en la trayectoria, no se consiguió debido a la inestabilidad en los valores que se obtienen del sensor de color utilizado. Este problema se podría llegar a solucionar con la aplicación de filtros, pero esto supondría la implementación de diferentes filtros y realizar diversas pruebas. Dado en este punto del trabajo se disponía de poco tiempo, dado al tiempo gastado solventando otros problemas de mayor importancia y a que la acción de parar ya se consiguió con el sensor ultrasónico, se decidió descartar esta acción.

10.2. Posibles mejoras.

En cuanto a las posibles mejoras para este prototipo se proponen las siguientes:

- Utilizar dos sensores de color en vez de uno solo, de esta manera se conseguiría bajar el porcentaje de veces que el robot deja de seguir la trayectoria.
- Realizar un estudio e implementación de filtros con lo que se consiga detectar fiablemente un color seleccionado en la trayectoria, con el objetivo de realizar la acción de parar ante la detección de un color seleccionado en la trayectoria.
- Otra alternativa al punto anterior es obtener otro sensor de color con el que se obtengan valores estables para diferentes colores, obteniendo un nuevo sensor de color también se podría probar si se consigue subir y bajar pendientes mayores a las que se consiguen con el actual sensor.
- Ya que se ha dotado al robot con un acelerómetro que no se utiliza para la realización de este proyecto, se podría utilizar para realizar el equilibrio con este sensor. Se podría calcular la inclinación por trigonometría.
- La brújula se podría utilizar para orientar el sensor hacia un punto en concreto. Podría tener dos configuraciones, seguimiento de trayectorias por línea y orientación por brújula.

11. IMPLICACIONES AMBIENTALES.

La elaboración de este proyecto supone un mínimo impacto medioambiental, ya que sólo se ha impreso una mínima parte de la información necesaria, dado que el papel es reciclable eso supondrá un mínimo efecto medioambiental, el resto de información se ha consultado en formato digital. También se ha impreso en papel la trayectoria, sólo se ha impreso dos veces. Se ha utilizado poliuretano para hacer una de las rampas utilizadas, este es el material con más efecto medioambiental, aunque los impactos ambientales de la espuma de poliuretano son relativamente leves y más favorable que otras espumas de polímero, y se ha utilizado poca cantidad esto supondrá un impacto ambiental muy bajo. Se ha utilizado otra rampa con objetos cotidianos como una carpeta que se puede seguir utilizando.

En cuanto al robot empleado de lego la empresa en una breve presentación de 2014 dice lo siguientes con respecto a su impacto medioambiental [8]:

"*Redujimos nuestro impacto ambiental Para minimizar nuestro impacto negativo en el mundo en el que vivimos actualmente y el planeta que nuestros niños heredarán mañana, constantemente buscamos medidas de operación más sustentables; entre otras iniciativas en 2013:*

- *Nos unimos al programa WWF Climate Savers para mostrar nuestro compromiso para reducir nuestras emisiones totales de CO2 (incluidos los proveedores)*
- *Mejoramos nuestra eficiencia energética en un 6.7 por ciento, lo que suma una mejora total del 30 por ciento durante los últimos cinco años*
- *Logramos nuestro objetivo de reciclar el 90 por ciento de nuestros desechos*
- *Redujimos el tamaño de todas las nuevas cajas de LEGO en un 18 por ciento, lo que reduce nuestro impacto de CO2 de empaquetado en un 10 por ciento y ahorramos aproximadamente 4,000 toneladas de cartón al año*"

Por lo que dicen parecen estar comprometidos con el medioambiente. El robot utilizado para este proyecto es de propiedad de la universidad, por lo que tiene fines educativos y se imparte clases con él. Por lo que su impacto ambiental es poco con relación al número de usuarios.

12. BIBLIOGRAFÍA.

[1] www.hitechnic.com/colorsensor

[2] http://www.coneural.org/florian/papers/05_cart_pole.pdf

[3] Modelo matemático utilizado: [http://www.academia.edu/4468049/Controlling_an_Inverted_pendulum_using_state_space_modeling_method_step_by_step_design_guide_for_control_students_]

[4]ftp://ftp.unicauca.edu.co/Facultades/FIET/DEIC/Materias/Teoria_Sistemas_Lineales/Archivos%20curso/Cap%207/optimo.pdf

[5] http://www.hitechnic.com/blog/gyro-sensor/htway/

[6]http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SystemModeling

[7] http://bricxcc.sourceforge.net/

[8] http://www.lego.com/es-ar/aboutus/news-room/2014/february/non-financial-result-2013

http://www.teamhassenplug.org/robots/legway

Seguimiento de líneas: [http://www.lejosconlego.com/2012/07/como-hacer-un-line-follower-seguir-la.html]

http://www.lego.com/es-es/mindstorms/?domainredir=mindstorms.lego.com

http://www.hitechnic.com/

Espacio de estados: http://www.ib.cnea.gov.ar/~instyctl/Tutorial_Matlab_esp/invSS.html

http://www.techbricks.nl/My-NXT-projects/nxt-self-balancing-Segway-nxtway-robot.html

ANEXOS

ANEXO A. Código del programa

```
//Definición constantes

//definiciones entradas y salidas

#define GYRO       IN_4

#define LEFT_MOTOR   OUT_B

#define RIGHT_MOTOR  OUT_A

#define MOTORS       OUT_AB


// Valores controlador equilibrio

#define KGYROANG  26.5

#define KGYROVEL  1.56

#define KPOS      0.15

#define KVEL      0.13


// Valores controlador seguidor línea


#define Kp 0.9

#define Ki 1.1

#define Kd 0.001


// Constante de velocidad máxima para los dos motores.

//300 grados/s para cada motor.

#define CONTROL_VEL  600.0
```

```
//Variables globales
float Sum_motores = 0;     //suma velocidad de los dos motores

//Variables para el cálculo del periodo de ejecución de la subrutina equilibrio
long tCalcStart = 0;     //Valor primer de periodo de equilibrio
float tInterval = 0;     //Periodo de ejecución de equilibrio

// Variables lectura Gyro.
float Gyro_Offset;
float Gyro_Angle_Global = 0;

//Variables PID seguidor líneas
float actual_position;
float actual_position_old;
float proportional;
float integral;
float derivativo;
float previous_error_color=0.0;
float error_color=0.0;
float offset_color=7.5;
float output;

//Variables globales motor
float Motor_Posicion = 0;
long Motor_Rot_Sum = 0;
long Motor_Rot_SumPrev = 0;
```

```
long Motor_Rot_D3 = 0;
long Motor_Rot_D2 = 0;
long Motor_Rot_D1 = 0;
long Motor_Rot_D = 0;
long Motor_Rot_Izq = 0;
long Motor_Rot_Der = 0;
int power, powerLeft, powerRight;
int LEFT = 0;
int RIGHT = 0;

// Variables sensor ultrasonido.
int distancia = 0;
int detectado_obs=0;

//=========================================//
// Tarea Offset. Se calcula el valor del offset inicial.
void Offset()
{
  float Gyro_Suma;
  int  i, Gyro_Valor;

  // Muestra de mensaje indicando inicio de calibrado
  ClearScreen();
  TextOut(0, LCD_LINE4, "Calibrando");

  Off(MOTORS);        //se apagan los motores.
```

```
  // se toman 100 valores del sensor gyro y se calcula el promedio
  Gyro_Suma = 0.0;
  for (i=0; i<100; i++) {
     Gyro_Valor = SensorHTGyro(GYRO);
     Gyro_Suma += Gyro_Valor;
     Wait(5);
  }

  //el promedio es el valor del offset inicial
  Gyro_Offset = Gyro_Suma / 100;

}

//=========================================//
// Tarea main. Se calcula offset inicial.
task main()
{
  // Inicialización de los sensores Gyro, color y ultrasónico
  SetSensorHTGyro(GYRO);
  SetSensorLowspeed(IN_3);
  SetSensorUltrasonic (IN_2);
  Wait(50);

  //se llama a la tarea que calcula en offset inicial
  Offset();
```

```
  //se deja 3 segundos para poner el robot en posición para equilibrio
  ClearScreen();
  PlayTone(440,1000);
  Wait(3000);
  PlayTone(440,1000);

}

//===========================================//
// Tarea DatosGyro. Donde se obtiene el valor actual del sensor Gyro,
// Se actualiza el valor de Gyro_Offset, se obtiene valor de gyroSpeed y gyroAngle

inline void DatosGyro(float &gyroSpeed, float &gyroAngle)
{
  float Gyro_Actual;
  Gyro_Actual = SensorHTGyro(GYRO);
  Gyro_Offset = 0.0005 * Gyro_Actual + (1-0.0005) * Gyro_Offset;
  gyroSpeed = Gyro_Actual - Gyro_Offset ;
  Gyro_Angle_Global += gyroSpeed*tInterval;
  gyroAngle = Gyro_Angle_Global;
}

//===========================================//
//Tarea DatosMotor. Donde se calcula el valor de los encoders de los motores
//para obtener Motor_Velocidad y Motor_Posicion
```

```
inline void DatosMotor(float &Motor_Velocidad, float &Motor_Posicion)
{

  // Datos rotación de los motores.
  Motor_Rot_Izq = MotorRotationCount(LEFT_MOTOR);
  Motor_Rot_Der = MotorRotationCount(RIGHT_MOTOR);

  // Se calcula la suma de la rotación de los motores.
  Motor_Rot_SumPrev = Motor_Rot_Sum;
  Motor_Rot_Sum = Motor_Rot_Izq + Motor_Rot_Der;

  //Calculo de Motor_Posicion a partir de la suma de la rotación de los motores.
  Motor_Rot_D = Motor_Rot_Sum - Motor_Rot_SumPrev;
  Motor_Posicion += Motor_Rot_D;

  // se calcula Motor_Velocidad a partir del promedio de las ultimas 4 deltas.
  Motor_Velocidad =
(Motor_Rot_D+Motor_Rot_D1+Motor_Rot_D2+Motor_Rot_D3)/(4*tInterval);

  // Se guardan los valores de las deltas.
  Motor_Rot_D3 = Motor_Rot_D2;
  Motor_Rot_D2 = Motor_Rot_D1;
  Motor_Rot_D1 = Motor_Rot_D;
}
```

```
//=========================================//
// Tarea ControlTrayectoria. Se calcula la cantidad de giro que se debe dar
// para seguir la línea.

inline void ControlTrayectoria(int power, int &powerLeft, int &powerRight)
{

 actual_position=SensorHTColorNum (IN_3);

 // Controlador PID de control de trayectoria.
 error_color = actual_position - offset_color;
 proportional = Kp * error_color;
 integral = integral + error_color*tInterval;
 derivativo = (error_color - previous_error_color)/ tInterval;
 output = proportional + Ki * integral + Kd * derivativo;
 previous_error_color = error_color;

 // No girar si se encuentra obstáculo.
 if (detectado_obs==1){

   powerLeft = power;
   powerRight = power;
 }
```

```
 // Girar si no se encuentra obstáculo.
if (detectado_obs==0){

 powerLeft = power - output;
 powerRight = power + output;
 }


 // Limitación de potencia a +/- 100.
 if (powerLeft > 100)   powerLeft = 100;
 if (powerLeft < -100)  powerLeft = -100;

 if (powerRight > 100)   powerRight = 100;
 if (powerRight < -100)  powerRight = -100;
}

//==========================================//
// Tarea CalcIntervalo. Se calcula el intervalo de cada iteración.

inline void CalcIntervalo(long cLoop)
{
  //la primera iteración se pone un valor fijo como intervalo
 if (cLoop == 0) {
   tInterval = 0.0055;
   tCalcStart = CurrentTick();
```

```
 } else {
    // Cálculo del intervalo a partir del valor medio.
    tInterval = (CurrentTick() - tCalcStart)/(cLoop*1000);
  }
}

//===============================================//
// Tarea Equilibrio. Esta tarea se ejecuta después del main.

task Equilibrio()
{

  Follows(main);

  float gyroSpeed, gyroAngle;
  float Motor_Velocidad;
  long cLoop = 0;


  // Se limpia la pantalla y se resetean los encoders.
  ClearScreen();
  ResetRotationCount(LEFT_MOTOR);
  ResetRotationCount(RIGHT_MOTOR);
```

```
  while(true) {
    CalcIntervalo(cLoop++);
    DatosGyro(gyroSpeed, gyroAngle);
    DatosMotor(Motor_Velocidad, Motor_Posicion);

    // Se aplica a Motor_Posicion la variable Sum_motores que indica la potencia
    // que se le quiere dar a los motores
    Motor_Posicion -= Sum_motores * tInterval;

    // Cálculo de la acción de control para mantener el equilibrio
    power = (KGYROVEL * gyroSpeed +
          KGYROANG * gyroAngle) / 1.1 +
          KPOS     * Motor_Posicion +
          KVEL     * Motor_Velocidad;

    actual_position_old= actual_position;
    ControlTrayectoria(power, powerLeft, powerRight);

    OnFwd(RIGHT_MOTOR, powerRigth);
    OnFwd(LEFT_MOTOR, powerLeft);

    Wait(8);
  }
}
```

```
//=============================================//
// Tares ParaOadelante se ejecuta después del main.
// Indica si se para porque se detecta objeto o se sigue adelante.

task ParaOadelante ()
{

 Follows (main);
 ClearScreen();
 int RIGHT;
 int LEFT;

 while (true)
  {

 if (detectado_obs==1){
       // Para si se detecta objeto
       LEFT = 0;
       RIGHT = 0;
       do
       {
       Sum_motores = (LEFT + RIGHT) * CONTROL_VEL / 200.0;
       }while (detectado_obs==1)
 }
```

```
  if (detectado_obs==0){
        // Sigue si no se detecta objeto
        LEFT = -50;
        RIGHT = -50;
        do
        {
        Sum_motores = (LEFT + RIGHT) * CONTROL_VEL / 200.0;
        }while (detectado_obs==0)
  }
  }

}

//===========================================//
// Tarea Ultrasonido. Se ejecuta después del main.
// Indica si un objeto se encuentra cerca.

task Ultrasonido ()
{

 Follows (main);
 while (true)
 {
     //lectura del sensor ultrasonido
     distancia = SensorUS(IN_2);
```

```
    // si se detecta un objeto a un distancia entre 4 y 40cm detectado_obs= 1
    if (distancia > 4)
    {
     if (distancia < 40){detectado_obs= 1; }
     }

     if (distancia < 4)  {
     detectado_obs= 0;}
     if (distancia > 40)  {
     detectado_obs= 0;}
   }
}
```

ANEXO B. Código para obtener color.

```
byte fileHandle;        // Inicialización de las variables necesarias para guardar los datos
short bytesWritten;
float actual_position;          // Inicialización de la variable que contendrá el color
task main ()
{
SetSensorLowspeed(IN_3);              // Inicialización del sensor de color
Off(OUT_AB);                          // Se paran los motores
DeleteFile("Datos.txt");       // se crea un archivo donde se guardarán los datos
CreateFile("Datos.txt", 50000, fileHandle);
while(true)
{
actual_position=SensorHTColorNum (IN_3);   // Se obtiene el valor de color del sensor
NumOut(0, LCD_LINE1, actual_position);     // se muestra por la pantalla del brick

string col = NumToStr(actual_position);                 // Se guardan los valores en un
                                                        // document de texto.
string write = StrCat(col);
WriteLnString(fileHandle,write, bytesWritten);
Wait(10);
}
}
```

AXEXO C. Construcción del Robot.

En este anexo de encuentran los pasos a seguir para la construcción del robot *Segway* con el kit Lego Mindstorms NXT. Los sensores que aparecen en las imágenes siguientes no corresponden con los sensores que se han utilizado, en la imagen 56 se debe utilizar el sensor Gyro de Hitechnic, en la figura 64 se debe utilizar un sensor acelerómetro y una brújula y en la figura 77 un sensor de color. Las ruedas que aparecen en las imágenes son de 43x22, las utilizadas para este proyecto fueron las de 56x25.

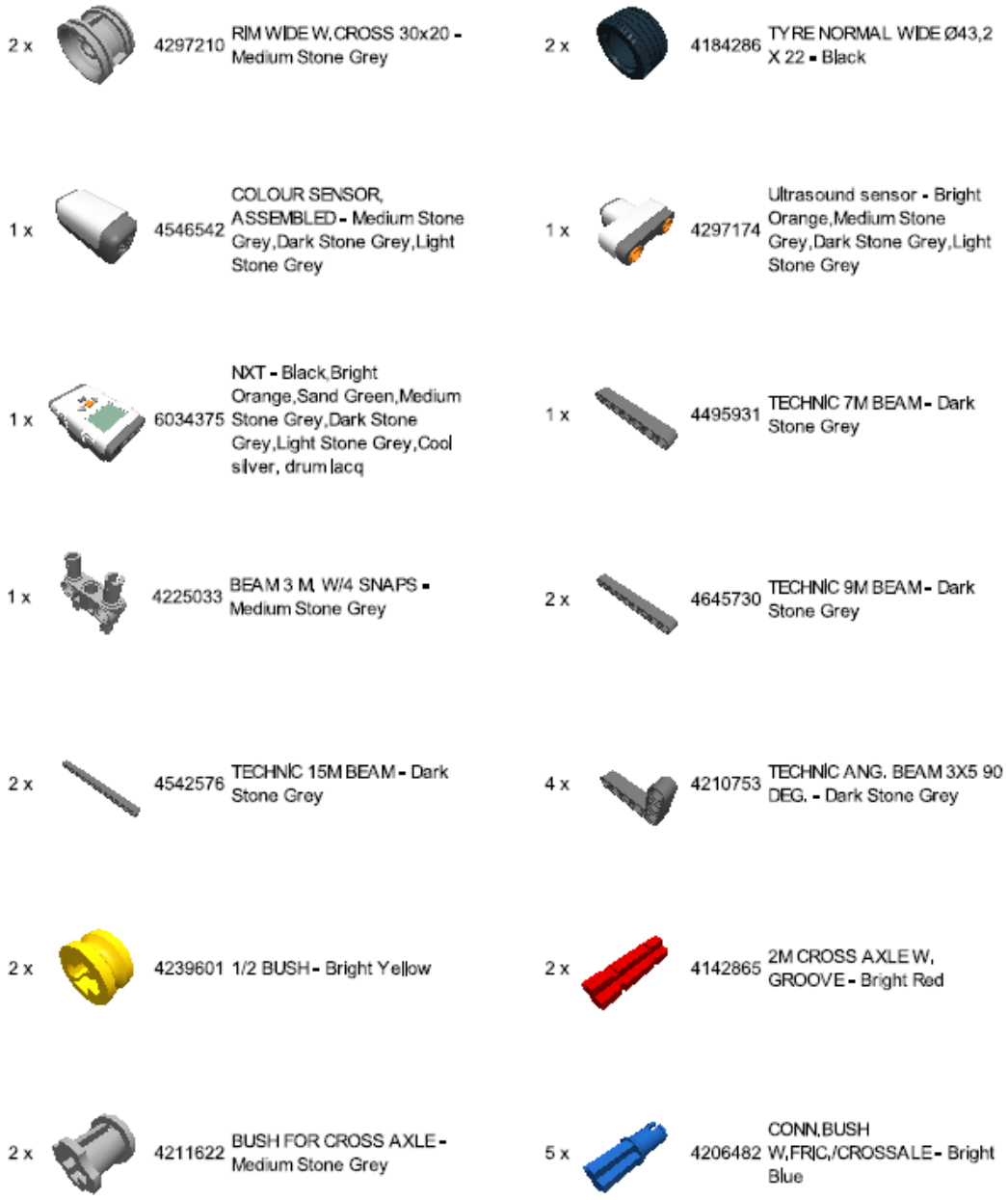


Figura 68: Elementos necesarios para la construcción del robot I.

Figura 69: Elementos necesarios para la construcción del robot II.

Figura 70: Construcción del robot paso 1.

Figura 71: Construcción del robot paso 2.

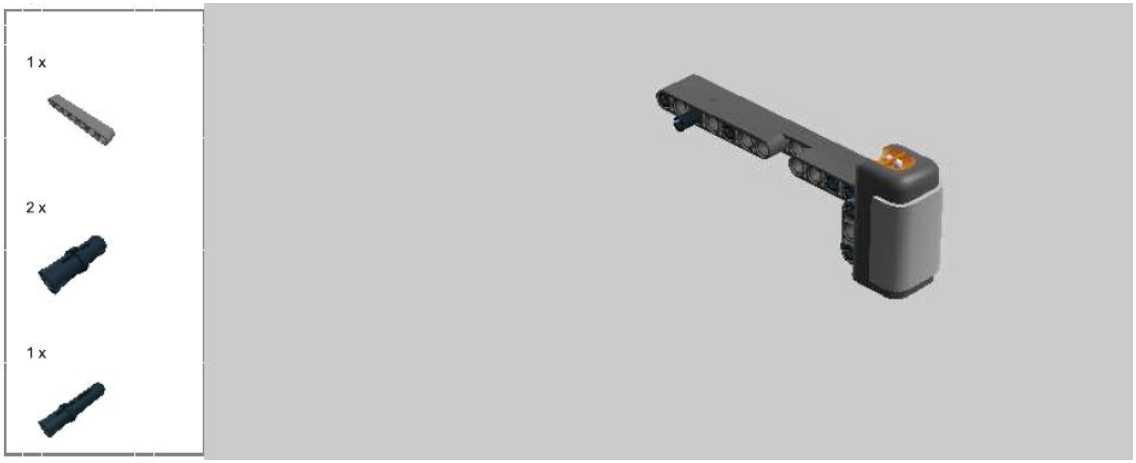


Figura 72: Construcción del robot paso 3.

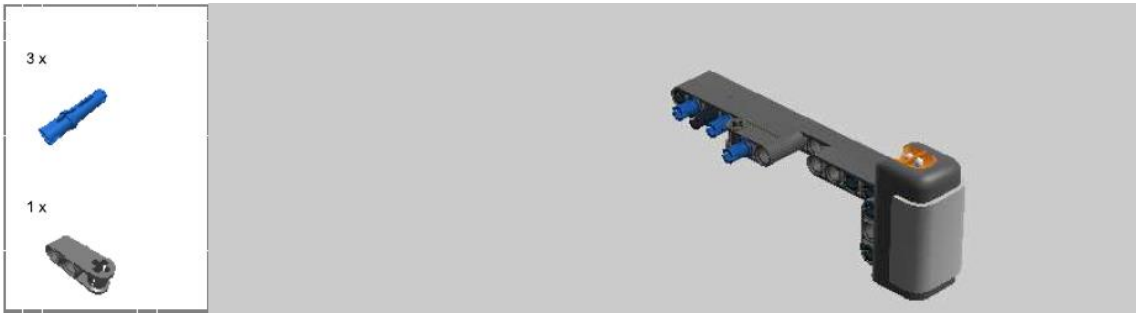


Figura 73: Construcción del robot paso 4.

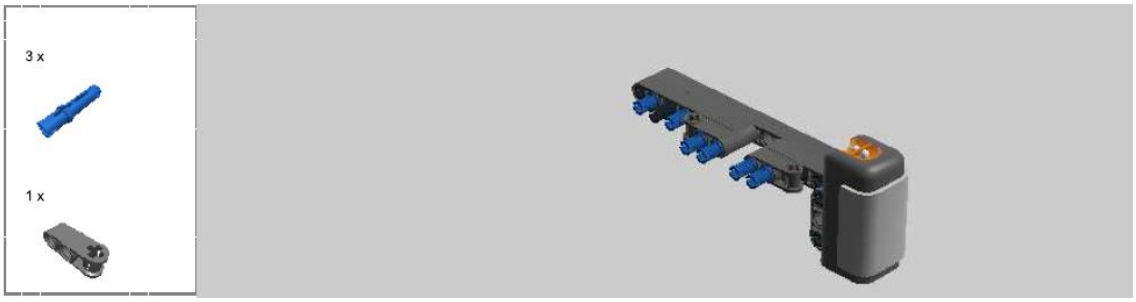


Figura 74: Construcción del robot paso 5.

Figura 75: Construcción del robot paso 6.

Figura 76: Construcción del robot paso 7.

Figura 77: Construcción del robot paso 8.

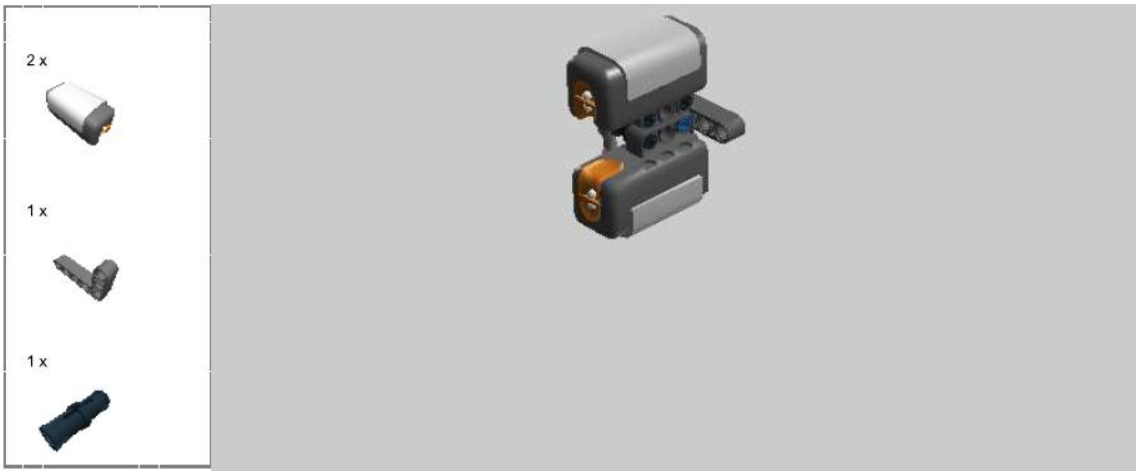


Figura 78: Construcción del robot paso 9.

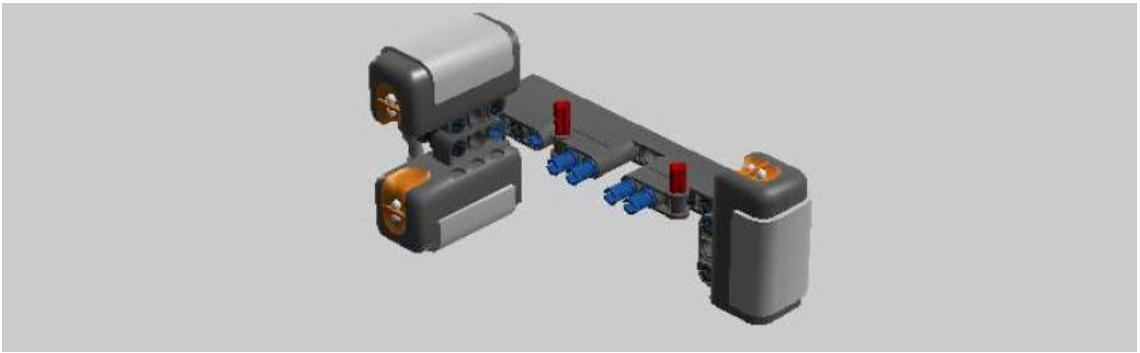


Figura 79: Construcción del robot paso 10.

Figura 80: Construcción del robot paso 11.

Figura 81: Construcción del robot paso 12.

Figura 82: Construcción del robot paso 13.

Figura 83: Construcción del robot paso 14.

Figura 84: Construcción del robot paso 15.

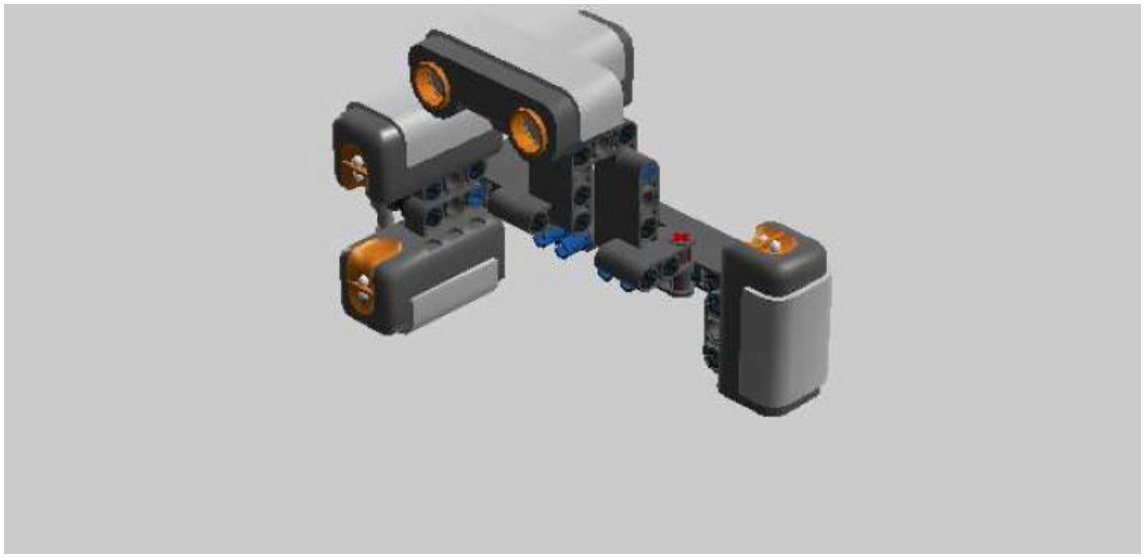


Figura 85: Construcción del robot paso 16.

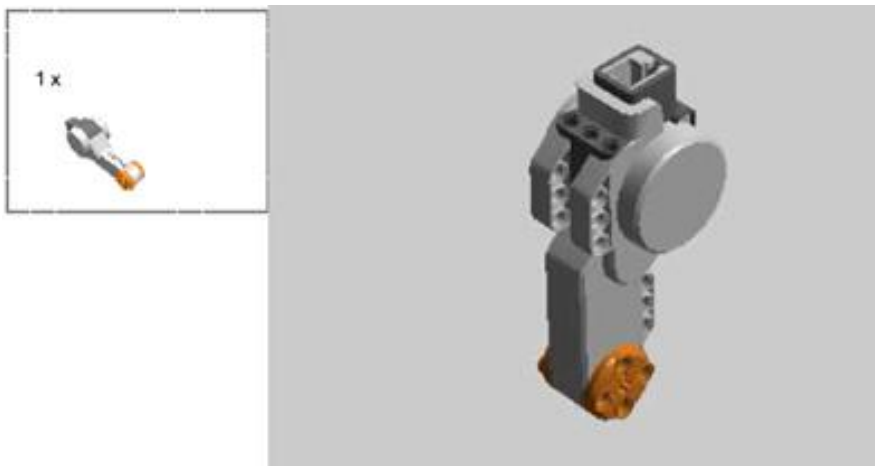


Figura 86: Construcción del robot paso 17.

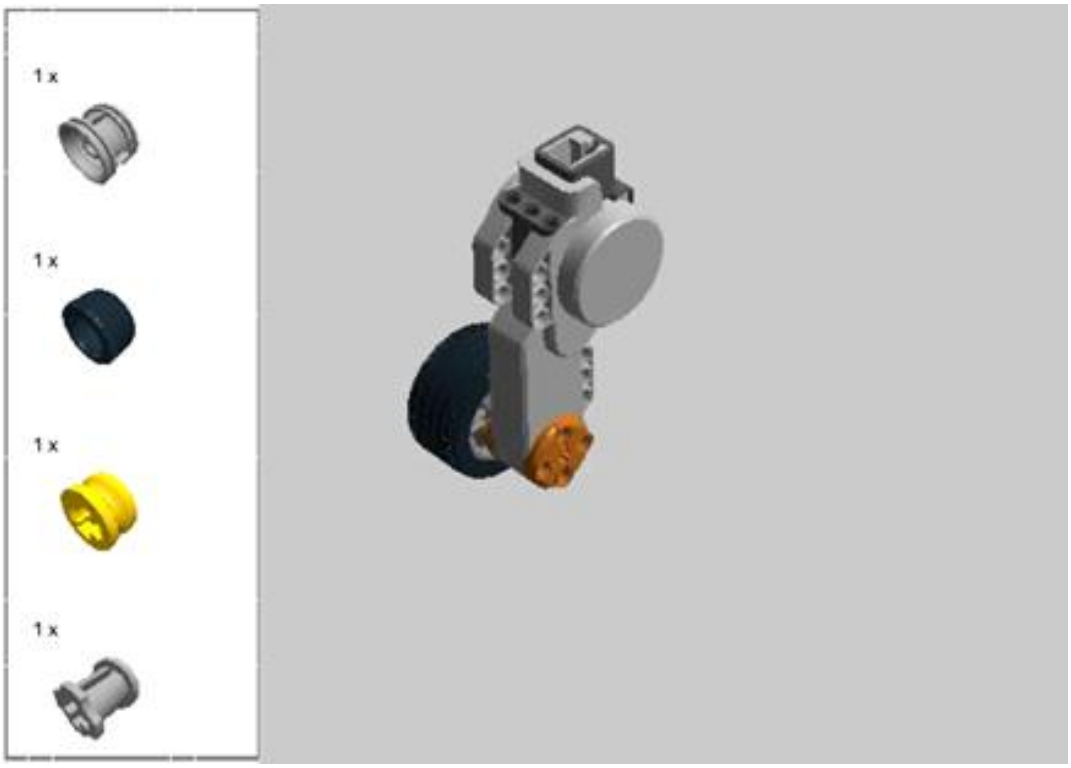


Figura 87: Construcción del robot paso 18.

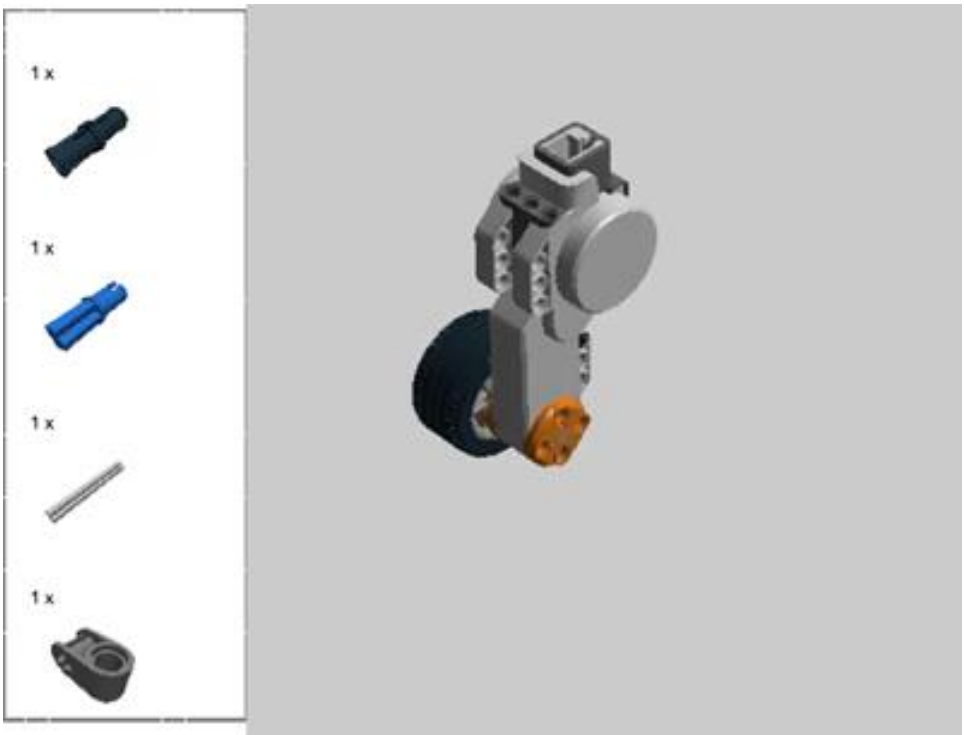


Figura 88: Construcción del robot paso 19.

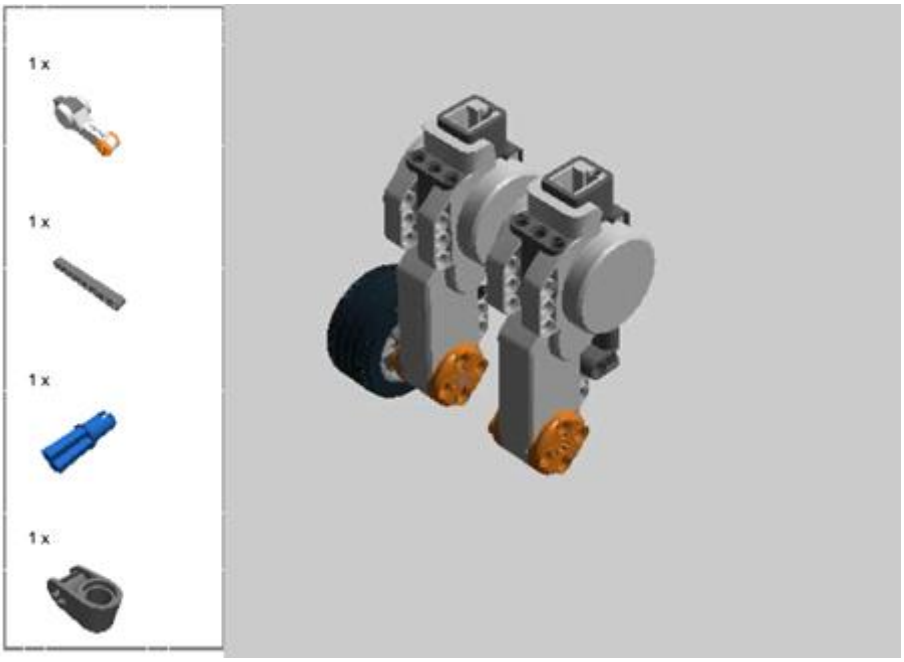


Figura 89: Construcción del robot paso 20.

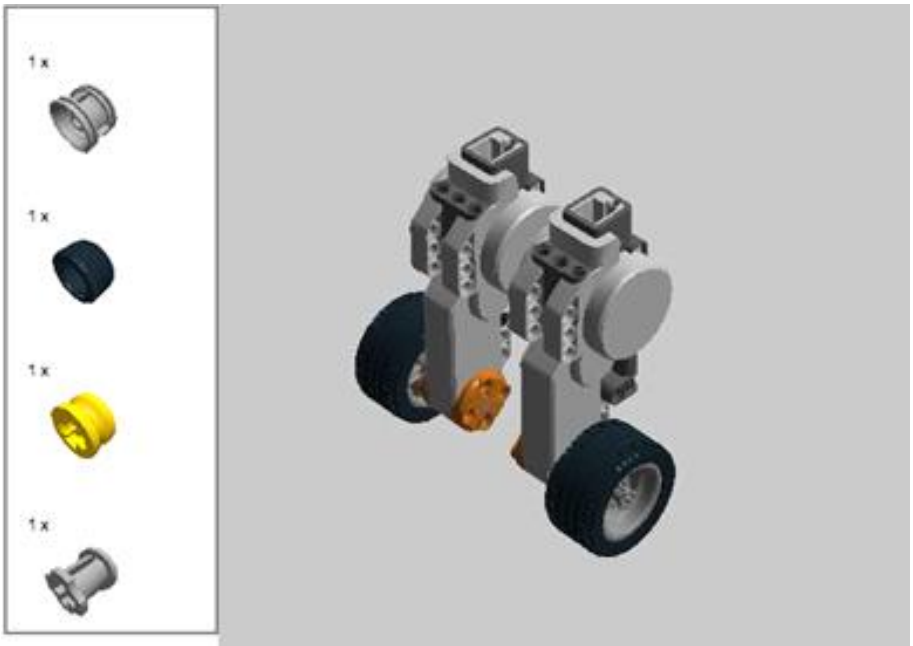


Figura 90: Construcción del robot paso 21.

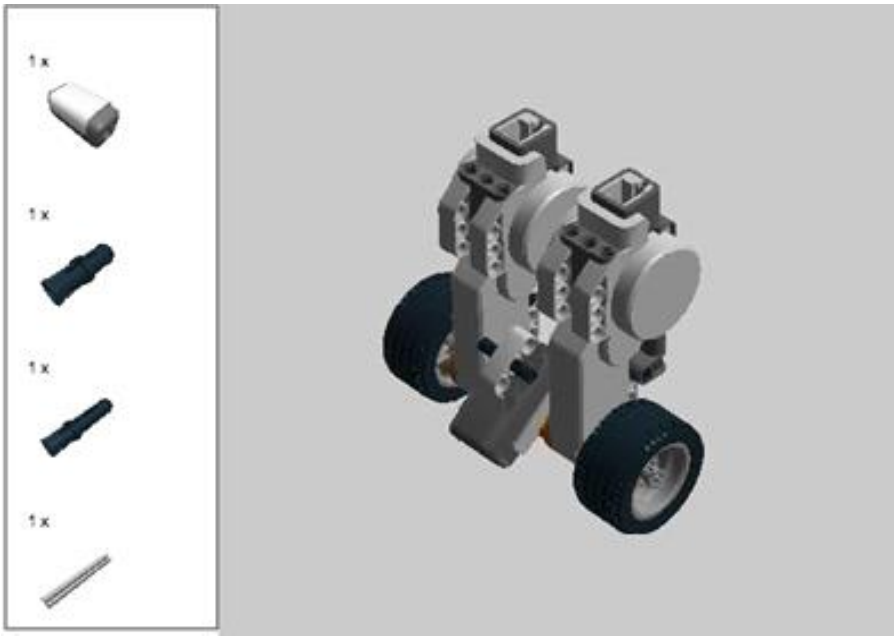


Figura 91: Construcción del robot paso 22.

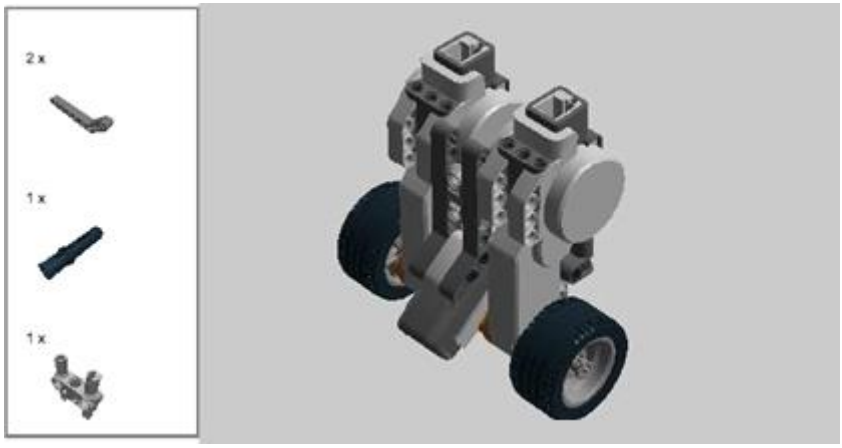


Figura 92: Construcción del robot paso 23.

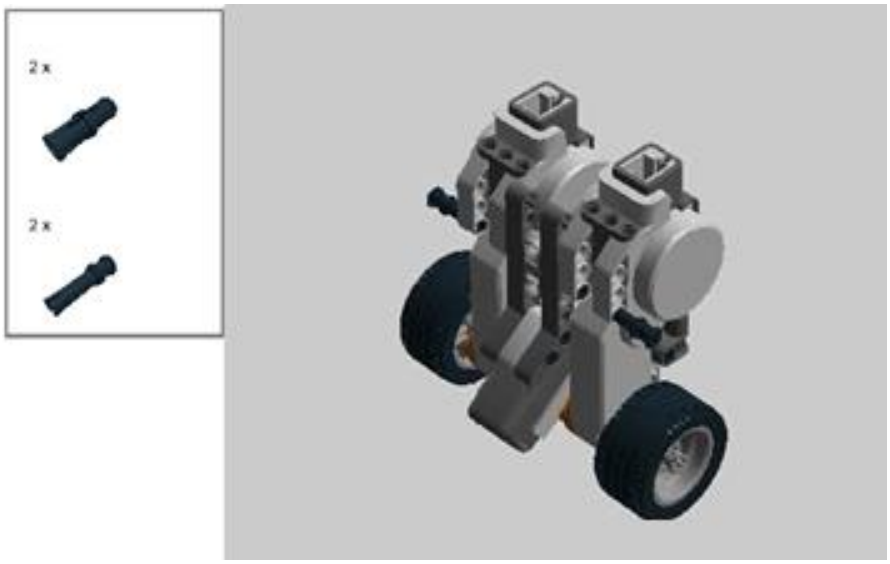


Figura 93: Construcción del robot paso 24.

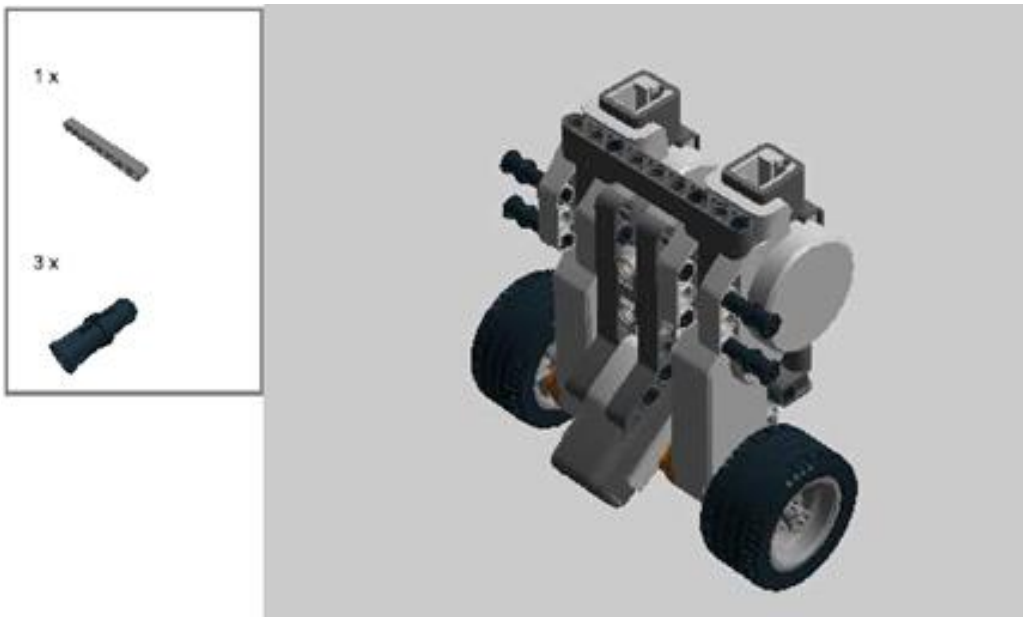


Figura 94: Construcción del robot paso 25.

Figura 95: Construcción del robot paso 26.

Figura 96: Construcción del robot paso 27.