Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIAL

# TRABAJO FIN DE GRADO

## *<CONTROL PARA UNA MAQUETA FUNICULAR>*

**Alumna**: Lejarreta Rodríguez, Estibaliz

**Director:** Gómez Garay, Vicente

**Curso:** 2018-2019

**Fecha:** <Lunes, 15 de Julio de 2019>

# [ES] Resumen

En este Trabajo Fin de Grado se ha realizado el Sistema de Control de la maqueta del Funicular de Artxanda con el kit LEGO Mindstorms NXT utilizando el software Brickx Command Center (BricxCC) en un lenguaje similar al C (NXC). Se parte de una maqueta realizada por una compañera de Grado y se mejora el diseño de las vías, se implementa un controlador que regula la velocidad de los vagones y otras funciones adicionales de seguridad como una parada de emergencia, detección de fallo en frenos,… En este documento se exponen las diferentes tareas realizadas durante el proyecto realizando un análisis del problema y de alternativas cuando es necesario. Es un Trabajo Fin de Grado realizado en colaboración con el Departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingeniería de Bilbao.

# [EN] Abstract

In this Final Degree Project, the Control System of the Artxanda Funicular model was made with the LEGO Mindstorms NXT kit using the Brickx Command Center (BricxCC) software in a language similar to C (NXC). It is based on a model made by a University partner and the design of the tracks is improved, a controller that regulates the speed of the wagons and other additional safety functions such as an emergency stop and brake failure detection are implemented. In this document the different tasks carried out during the project are exposed, making an analysis of the problem and alternatives when necessary. It is a Final Degree Project carried out in collaboration with the Department of Systems Engineering and Automation of the School of Engineering of Bilbao.

# [EU] Laburpena

Graduko Amaierako Lan honetan, Artxanda Funikularraren ereduaren Kontrol Sistema egin da LEGO Mindstorms NXT kitarekin, Brickx Command Center (BricxCC) softwarea erabiliz C antzeko hizkuntza batean (NXC). Graduaren ikaskide batek egindako eredu batean oinarritzen da eta ibilbideen diseinua hobetzen da, abiadura erregulatzen duen kontrolatzaile bat eta bestelako segurtasun funtzio, adibidez larrialdietarako geldialdia edo balazta akatsak hautematea, ezartzen dira. Dokumentu honetan proiektuan zehar burutzen diren zereginak aurkesten dira, eta arazoen eta alternatiben azterketa beharreskoak direnean egiten dira. Bilboko Ingeniaritza Eskolako Sistemen Ingeniaritza eta Automatizazio Sailarekin lankidetzan egindako Graduko Amaierako Lana da.

# Palabras clave

Lego NXT, maqueta funicular de Artxanda, control de velocidad.

# Índice

# Índice de Ilustraciones

# Índice de Tablas

# Índice de gráficas

# 1. Introducción

La finalidad del TFG es poner en práctica los conocimientos adquiridos durante el Grado y superar el reto que supone enfrentarse a la definición de un proyecto, llevarlo adelante resolviendo las dificultades que surjan, documentarlo ajustándose a las directrices marcadas por la normativa y defenderlo ante un tribunal académico. Este TFG parte de uno anterior realizado por otra alumna de grado y respetando la esencia de su trabajo incorpora mejoras que permiten un mejor funcionamiento. En concreto, a partir de una maqueta construida con el kit Lego Mindstorms que representa el funcionamiento del funicular de Artxanda, se analiza su comportamiento, se investigan los problemas detectados en su funcionamiento, se proponen mejoras y se llevan a cabo. El TFG se desarrolla haciendo uso de los recursos del Departamento de Ingeniería de Sistemas.

# 2. Contexto

Para el desarrollo de este trabajo se utiliza el kit de Lego Mindstorm. Lego Mindstorms Robotic nació de una colaboración entre el Instituto de Tecnología de Massachusetts (MIT) y el grupo Lego. En 1985, Kjeld Kirl Kristiansen, entonces Director Ejecutivo (CEO) de The Lego Group se enteró del trabajo de Seymour Papert en MIT y se sorprendió de lo similares que eran sus objetivos de aprendizaje a través de la construcción. El grupo Lego comenzó una asociación con el Media Lab de Papert, financiando su investigación y compartiendo ideas. De esta colaboración nació la idea de que los niños usaran el lenguaje de programación LOGO para comandar robots de Lego Brick a partir de su propio diseño. Tras un largo periodo de investigación comienza el desarrollo del ladrillo RCX, un bloque de Lego que contaba con un microcontrolador. De este modo, las estructuras de Lego pasaban de ser estructuras estáticas a estructuras dinámicas capaces de interactuar con el entorno. La idea se fue desarrollando gradualmente hasta lograr la versión definitiva de la versión Mindstorms NXT, la cual se comenzó a comercializar en 2006. Más adelante, en 2013 Lego comenzó comercializar la tercera versión denominada Mindstorms EV3.

LEGO Mindstorms NXT es un set programable que soporta hasta cuatro sensores y puede controlar hasta tres servomotores que se conectan mediante un cable RJ12. El ladrillo cuenta con una pantalla LCD monocromática de 100x60 píxeles, cuatro botones para su manejo y un microcontrolador de 32 bits con 256KB de memoria y 6KB de RAM. El dispositivo tiene baterías 6AA. Para programar este dispositivo existen diferentes entornos de programación, en este proyecto se utilizará el Bricx Command Center (BricxCC) donde el lenguaje de programación es el NXC (Not Exactly C) diseñado por John Hansen en 2006 [4]. Para ello se deberá conectar al ordenador, bien vía Bluetooth o vía USB.


**Ilustración 1: Ladrillo Mindstorms NXT**

Los **servomotores** [Ilustración 2] capacitan el movimiento del robot. Tienen un sensor de rotación incorporado que mide los grados girados por el motor, datos que pueden ser traspasados al ladrillo inteligente. En la tabla 1 se muestran algunas características del motor.

**Tabla 1: Características del motor**

| NXT | Fuerza de torsión | Velocidad de rotación | Intensidad | Potencia Mecánica | Potencia eléctrica | Eficiencia |
|---|---|---|---|---|---|---|
| **9V** | 16.7 N.cm | 117rmp | 0.55 A | 2.03 W | 4.95 W | 41% |

**Ilustración 2: Servomotor Lego**

Los **sensores** permiten al robot interactuar con lo que sucede en el entorno. Ante un cambio en el entorno reciben una señal y la trasmiten al ladrillo. Cada sensor tiene requisitos únicos para leer e interpretar sus valores por lo que se debe configurar cada puerto del sensor antes de poder utilizarlo. Hay dos configuraciones diferentes para especificar: el tipo de sensor y el modo de sensor.

El tipo de sensor determina como el NXT interactúa con el sensor. Existe una gran variedad de sensores, pero en el kit tan solo vienen incluidos cuatro tipos: sensor de contacto, sensor de luz, sensor de sonido y sensor ultrasónico.

El sensor de contacto tiene dos posiciones: presionado y no presionado. El sensor tendrá valor 1 cuando este presionado y 0 cuando no lo esté. El sensor cuando colisiona con algún elemento, una pequeña cabeza externa se contrae, permitiendo a una pieza interior del sensor cerrar el circuito eléctrico[Fig. 3]. Consiste en un interruptor de contacto abierto con un resorte y conectado en serie con una resistencia de carga.

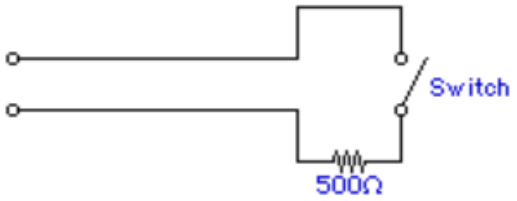



**Ilustración 3: Circuito eléctrico del sensor de contacto**

Este sensor no registra la intensidad del contacto, solo que el contacto se ha realizado. Un problema que surge es que algunos interruptores mecánicos como este tienden a "rebotar" cuando se presionan. Los rebotes surgen como consecuencia de las vibraciones mecánicas y eléctricas de los contactos. Estos son considerados como ruido.

El sensor de luz permite al robot distinguir entre luz y oscuridad. Puede leer la intensidad de luz en una habitación y medir la intensidad de la luz en superficies coloreadas. Este sensor emite luz y captura la cantidad de luz reflejada. Consiste en un diodo emisor de luz LED rojo y un fotoreceptor que responde a la luz entrante. El modo predeterminado para el sensor de luz es el modo porcentaje. Siendo 100 muy luminoso y 0 para la oscuridad. El sensor se puede usar en dos modos:

- El primer modo detecta la luz del ambiente.
- En el segundo modo el mismo sensor emite una luz y luego mide la intensidad de luz que se refleja en las superficies. Este modo lo podemos usar para diferenciar el brillo de los colores en una superficie.

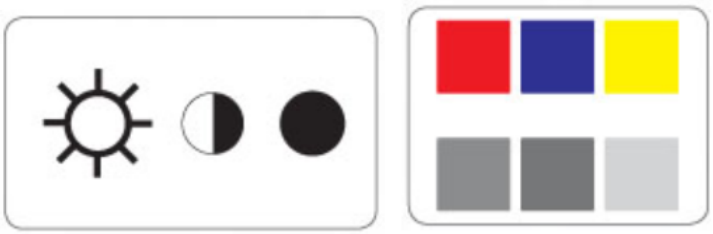


**Ilustración 4: Funcionamiento del sensor de luz**

El sensor de sonido mide el nivel de sonido que llega a un altavoz. El modo predeterminado es el modo porcentaje siendo 100 la saturación de ruido y 0 el completo silencio.

El sensor ultrasónico puede medir la distancia desde el sensor a algo que esta mirando y detectar movimiento. Puede mostrar la distancia tanto en centímetros como en pulgadas. La distancia máxima

que puede medir es 255cm con una precisión de 3cm. El sensor ultrasónico envía ondas de sonido ultrasónicas que rebotan con un objeto delante de él y luego regresan. Calcula la distancia en función del tiempo que tarde la onda en volver a él. Los objetos grandes y con superficie dura son detectados mucho mejor por el sensor, aquellos que son curvos, delgados o suaves, a veces tienen problemas para ser detectados.

**Ilustración 5: Sensor de contacto**

**Ilustración 6: Sensor de luz**

**Ilustración 7: Sensor de sonido**

**Ilustración 8: Sensor ultrasónico**

El modo de sensor le indica al ladrillo como interpretar los valores del sensor. Algunos lenguajes de programación, como el NXC (Not Exactly C), configuran automaticamente el modo de sensor según el tipo de sensor. Existen 8 modos de sensor, en la tabla 2 se recogen algunos de ellos:

**Tabla 2: Modos del sensor**

| Modo del sensor | Descripción |
|---|---|
| **Raw Mode** | Es el modo más sencillo. El valor devuelto por el sensor siempre se mueve en el rango [0-1023] |
| **Boolean Mode** | El sensor devuelve 0 o 1 |
| **Edge Count Mode** | El NXT cuenta cuantas veces el valor booleano cambia de valor. Este contador es inicialmente 0 y aumenta una unidad cada vez que el booleano cambia de valor |
| **Pulse Count Mode** | Similar al Edge Count Mode excepto que el contador solo se incrementa cuando el valor booleano va de 1 a 0 (no sucede lo mismo a la inversa). |
| **Percentage Mode** | El valor esta comprendido en un rango de 0 100. Los valores altos en modo Raw corresponden a bajos valores en el modo de porcentaje. Este es el modo predeterminado para un sensor de luz, donde 100 es la saturación completa. |
| **Rotation Mode** | Tal y como su nombre indica, este modo solo es adecuado para un sensor de rotación. El valor resultante del modo de rotación es una rotación acumulativa en incrementos de 22,5°. Por lo tanto, un valor de 16 representa una rotación completa. |

Para la conexión de los servomotores y los sensores con el ladrillo se utiliza un cable RJ12 con seis conductores. Este cable es similar al cable RJ12 estandar diferenciándose por la pestaña de enganche del conector.

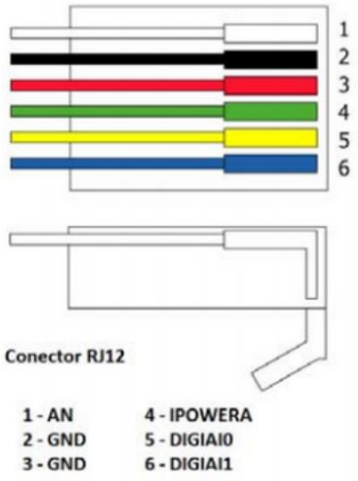



**Ilustración 9: Cable RJ12**

La función de cada uno de los pines se muestra en la tabla 3[11]:

**Tabla 3: Descripción pines de cable RJ12**

| Nº PIN | Color | Función |
|---|---|---|
| **1** | Blanco | Entrada analógica |
| | | Fuente de alimentación 9V DC para sensores RCX |
| **2** | Negro | Masa de referencia para la medida de señales. Tierra 0V |
| **3** | Rojo | |
| **4** | Verde | Fuente de alimentación de los sensores y encoders de los motores |
| **5** | Amarillo | Entrada/Salida para el protocolo de comunicación digital I2C |
| **6** | Azul | |

# 3. Antecedentes

En el laboratorio del departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingeniería de Bilbao se dispone de una maqueta Lego del funicular de Artxanda, desarrollada como trabajo de fin de grado por Aida De la Herrán Prada [5]. La maqueta se compone de una estructura inclinada de madera a modo de base sobre la que se colocan las vías. Los vagones construidos con piezas Lego recorren las vías de madera gracias a un servomotor por contrapeso por lo que están unidos por un hilo de nylon. En la mitad del recorrido hay una bifurcación, para la que se utilizan dos agujas y dos servomotores junto con un sensor de luz. En la parte superior e inferior de la maqueta se encuentran las estaciones construidas también con piezas Lego. Además, en la estación superior se dispone de un sensor de contacto.

El sistema realiza el movimiento de subida y bajada del funicular y una parada en la estación al final de cada recorrido gracias al sensor de contacto. El movimiento de los vagones se realiza por contrapeso a través de un motor situado en la parte superior de la maqueta. Para evitar la colisión de ambos vagones a mitad del recorrido hay una bifurcación con agujas que conducen a cada vagón por un carril. Para el movimiento de dichas agujas se dispone de otros dos motores, uno por cada aguja. Los motores son accionados a la señal de un sensor de luz que detecta la llegada de los vagones a la bifurcación.

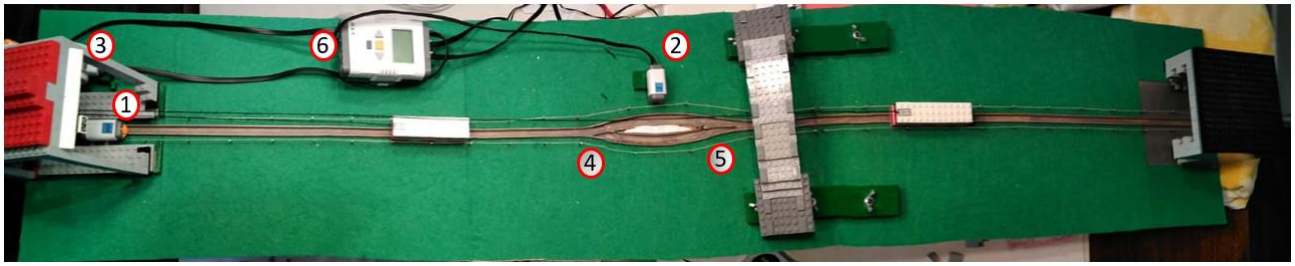



**Ilustración 10: Maqueta original**

**Tabla 4: Leyenda ilustración 10**

| | |
|---|---|
| **1.** | **Sensor de contacto** |
| **2.** | Sensor de luz |
| **3.** | Motor principal |
| **4.** | Motor aguja 1 |
| **5.** | Motor aguja 2 |
| **6.** | Brick NXT |

# 4. Objetivo y alcance del trabajo

El objetivo de este trabajo es mejorar el funcionamiento de la maqueta de la que se parte, analizar los problemas detectados, discutir las posibles alternativas, proponer las consiguientes acciones de mejora y llevarlas a cabo. Todo ello respetando en lo posible, el diseño original.

En concreto se llevarán a cabo actuaciones en los siguientes ámbitos:

- Mejora del diseño de las vías: La maqueta del funicular de la que se parte, durante el funcionamiento en ocasiones los vagones se tambalean, llegando a parase y volcar en algunos de los viajes realizados de prueba. Por lo tanto, se deberá buscar una solución a dicho tambaleo y vuelco de manera que los vagones del funicular "fluyan" correctamente.
- Automatizar la calibración del sensor de luz o bien buscar una alternativa: El sistema de calibración actual de la maqueta implica que cada vez que se ejecute el programa haya que calibrar el sensor de luz manualmente.
- Diseñar e implementar un control que regule la velocidad de los vagones.
- Implementar funciones adicionales de seguridad:
    - Parada de emergencia: El objetivo de la parada de emergencia es que el vagón sea capaz de detener su funcionamiento ante una alarma y volver a su condición normal una vez solucionado el problema que causó la alarma.
    - Determinación de la ubicación exacta de los vagones a lo largo de la vía: Localizar los vagones sería una facilidad para detectar cuando los vagones se encuentran parados debido a alguna avería o imprevisto.
    - Detección de paradas accidentales: Si por cualquier causa, durante el trayecto el vagón no avanzase pese a estar el motor en marcha, el motor debe pararse.
    - Simulación de un fallo en los frenos del funicular.

# 5. Beneficios que aporta el trabajo

Los beneficios que aporta el trabajo han sido clasificados en dos categorías diferentes. La primera se corresponde con los beneficios personales y la segunda con los beneficios sociales.

Personales: Partir de una maqueta realizada por una compañera ha supuesto un gran reto, ha habido que respetar su trabajo y mejorar los aspectos relacionados con los sistemas de control. Había que crear algo nuevo sobre algo ya establecido respetándolo y mejorándolo. Se han encontrado obstáculos en cuanto a los elementos con los que se contaba, tanto en lo referente a las piezas de Lego como en cuanto al tipo de cables compatibles con estas piezas. Para poder trabajar con las piezas originales ha habido que agudizar el ingenio. Se han ido realizando diferentes pruebas hasta conseguir las piezas y elementos adecuados para que el sistema funcionase correctamente. Se han aplicado conocimientos adquiridos durante el grado como Informática, Automática y Control o Control por Computador y además se han adquirido nuevos conocimientos técnicos ya que se ha utilizado un lenguaje de programación distinto al estudiado durante el grado.

Sociales: Al ser el funicular de Artxanda un elemento emblemático de la villa, esta maqueta y su funcionamiento mejorado podrá ser aprovechado como elemento de muestra para la captación de alumnado en Jornadas de Puertas Abiertas. También podrá ser utilizado, tal y como se ha hecho para el presente proyecto, por otros alumnos o alumnas que deseen mejorar el trabajo ya realizado.

# 6. Desarrollo del proyecto

En el desarrollo del proyecto se exponen las tareas realizadas para la mejora de la maqueta del funicular. Para el desarrollo del proyecto se distinguen las siguientes tareas:

- Mejora del diseño
- Detección de los vagones en la bifurcación
- Regulación de velocidad
- Freno del funicular
- Funciones de seguridad
- Programación
- Pruebas
- Escritura de la memoria

Para ello, en algunas de las tareas antes de llevarlas a cabo se deberá analizar cada problema observado las distintas alternativas posibles para solucionarlo, con el fin de encontrar la mejor opción y llevarla a cabo.

# 6.1. Mejora del diseño

## 6.1.1. Análisis del problema

Al poner en funcionamiento el funicular se observa que el movimiento de los vagones no es fluido, llegándose a parar en distintas zonas de la vía por diversos motivos. Mediante observación del funcionamiento, se han concretado los siguientes problemas:

**Problema 1:** El vagón no circula suavemente sobre el carril, mostrando un comportamiento oscilatorio.

**Problema 2:** Cuando -en cualquier punto de la vía- el vagón se queda atascado, el motor sigue girando. Se deduce que el vagón tiene poco peso. Una causa puede ser el exceso de pintura en los carriles de la vía.

**Problema 3:** Debido a la forma constructiva de la vía, los vagones se tambalean llegando en ocasiones a pararse, especialmente al entrar en la bifurcación.

**Problema 4:** Debido a que la parte de la vía de la bifurcación es un poco más ancha, el vagón se desvía, no llegando a pasar la bifurcación.

**Problema 5**: Al llegar los vagones a la bifurcación se quedan en situación de "Stand by"/atascados, de modo que no tienen impulso para realizar correctamente el pequeño giro necesario para entrar.

**Problema 6:** En alguna de las pruebas realizadas, al finalizar un recorrido, el vagón no llega con el suficiente impulso para generar señal en el sensor de contacto que se encuentra en la estación superior y proceder a la parada.

## 6.1.2. Propuesta de alternativas

Para la solución de los problemas se proponen distintas alternativas, de las cuales algunas podrán solucionar uno o varios de los problemas.

**Alternativa A:** Añadir una cuña con el fin de equilibrar la maqueta.

**Alternativa B:** Lijar la pintura antióxido del carril. La base es de madera de pino. Si se hace con cuidado es posible que mejore el contacto y por tanto la rodadura.

**Alternativa C:** Aumentar el peso de vagón de modo que se mejore el contacto entre el vagón y la vía, disminuyendo la oscilación en el movimiento de los vagones.

**Alternativa D:** Cambiar la posición de la unión del cable que soporta el peso del vagón; en vez de que la unión esté en la parte superior del vagón, que se una en la parte inferior del vagón, solución que parece más lógica y responde mejor a la realidad.

**Alternativa E:** Añadir un sobresuelo que impida que vagón se tambalee hacia los lados.

**Alternativa F:** Añadir relleno con cera en algunas zonas de la vía para uniformizar el trazado y ayudar a encaminar a los vagones.

**Alternativa G:** Aumentar la potencia del motor. En la maqueta actual la potencia es un 30% de la potencia total que permite el motor. Este método consistiría en ensayo-error hasta encontrar una velocidad a la que el vagón no se tambalee y a su vez no vaya demasiado rápido como para desviarse de las vías.

En la tabla 5 se observan las alternativas que se han utilizado para intentar resolver cada problema.

**Tabla 5: Análisis de alternativas mejora diseño (1)**

| | Problema 1 | Problema 2 | Problema 3 | Problema 4 | Problema 5 | Problema 6 |
|---|---|---|---|---|---|---|
| Alternativa A | - | | | | | |
| Alternativa B | | - | | | - | |
| Alternativa C | | - | - | | | |
| Alternativa D | | | - | | | |
| Alternativa E | | | - | | | |
| Alternativa F | | | | - | | |
| Alternativa G | | | - | | - | - |

Es complicado saber de antemano cuál o cuáles de las alternativas serán más eficaces. De hecho, probablemente la mejor solución sea una combinación de varias alternativas. Por lo tanto, se opta por realizar distintas pruebas experimentales para elegir cual o cuales de las alternativas son válidas.

### 6.1.3. Pruebas experimentales

**Alternativa A:** Para el equilibrio de la maqueta se añade una cuña en uno de los laterales. De este modo, la maqueta será más estable. Para su comprobación, utilizamos un nivel de burbuja. Este estará equilibrado cuando la burbuja este entre las dos marcas, tal y como se muestra en la imagen.

**Ilustración 11: Sistema de comprobación de equilibrio para la maqueta**

Tras poner el taco se puede observar como la trayectoria de los vagones mejora puesto que cesa la oscilación del vagón en su recorrido por la vía recta y se reduce levemente en la bifurcación. Por lo que dicha alternativa genera una mejoría notable.

**Alternativa B:** Para lijar el carril de la vía, se utilizan dos lijas del número cero, una un poco más gruesa que la otra. Se comienza con la lija más gruesa, mientras que se da un último acabado con la más fina. Se hace especial hincapié en la bifurcación, puesto que es donde más se interrumpe el trayecto.

**Ilustración 12: Lija gruesa**

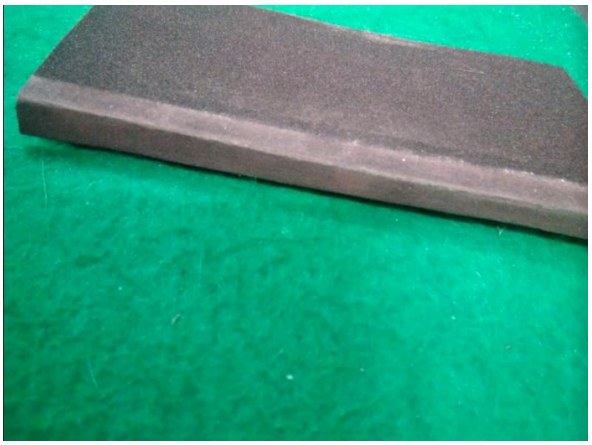


**Ilustración 13: Lija fina**

Tras lijar podemos observar como mejora ocasional y levemente el problema de atasco en la bifurcación (problema 5). Por lo que además de tomar esta medida será necesario tener en cuenta alguna otra alternativa. Sin embargo, no se da solución al problema de que cuando se atasca, el motor sigue en marcha (problema 2).

**Alternativa C:** Se pretende aumentar el peso del vagón de modo que al aumentar la fricción se mejore también la rodadura y por tanto la trayectoria, obteniendo así un movimiento fluido de los vagones, evitando los atascos ocasionados por la oscilación.

Para realizar la prueba se añade una lámina de cristal en la parte superior de cada vagón de aproximadamente 9,2 gramos Tras realizar las pruebas se comprueba que esta alternativa no es válida, puesto que al aumentar el peso del vagón también aumenta la fuerza centrífuga, provocando el descarrilamiento. Por lo tanto, esta alternativa queda desestimada.

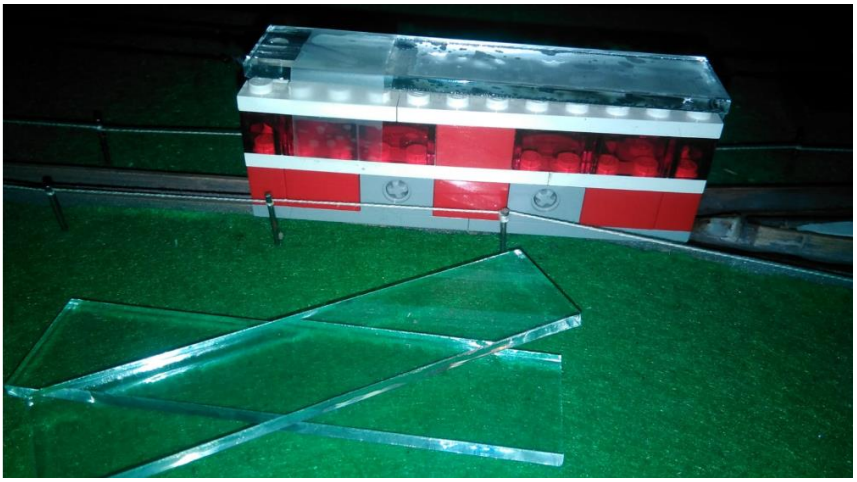


**Ilustración 14: Funicular con peso**

**Alternativa D:** Esta alternativa, a pesar de mejorar la trayectoria de los vagones reduciendo su balanceo, es complicada puesto que la pieza por abajo da problemas como el roce de la pieza con la vía. Además, el principal problema que supone amarrarlo por la parte inferior del vagón es que al llegar al sensor de contacto el vagón se eleva y no llega a pulsar nunca dicho sensor, tal y como se puede apreciar en la siguiente imagen. Por lo tanto, dicha alternativa no es válida.

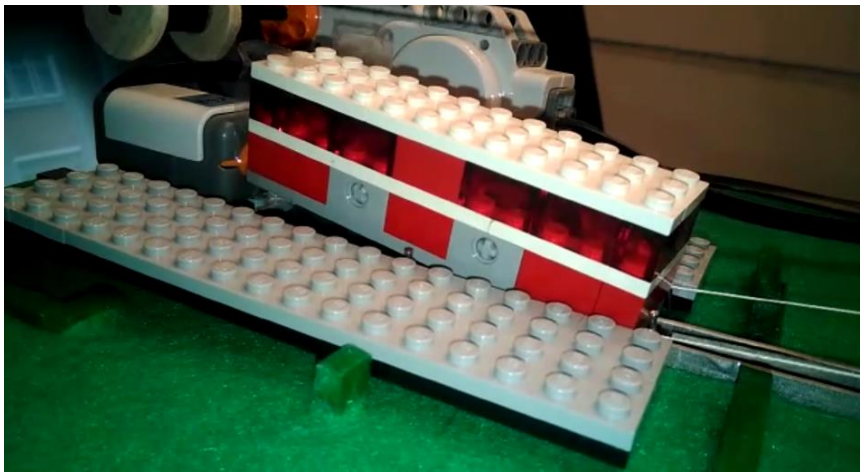


**Ilustración 15: Unión del hilo al vagón por debajo**

**Alternativa E:** Se opta por poner un sobresuelo pegado a la vía en la zona de la bifurcación que evite que este oscile al menos en esa zona, para así evitar la colisión de los dos vagones. Para ello se emplea un cartón corrugado de aproximadamente 4mm de espesor. De este modo se reduce la oscilación de los vagones en la zona de la bifurcación.

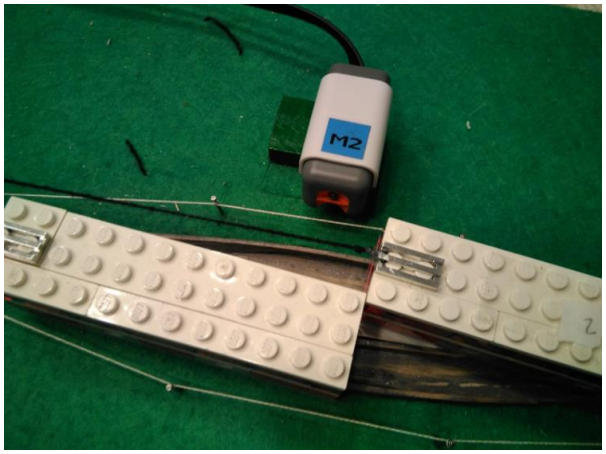



**Ilustración 16: Colisión de vagones**

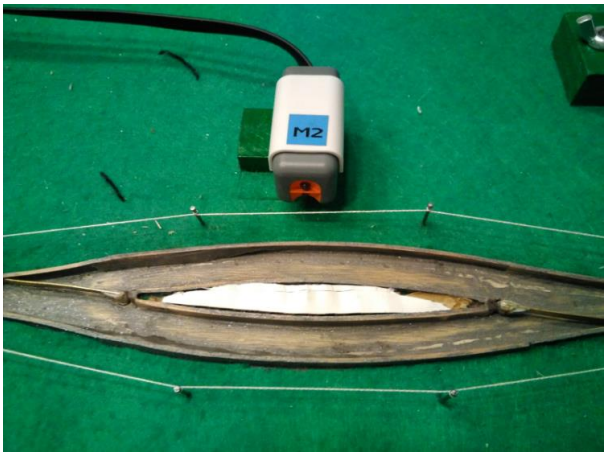



**Ilustración 17: Sobresuelo de cartón**

**Alternativa F:** Con el objetivo de encaminar al vagón en la zona de la bifurcación se añade cera en algunas zonas críticas donde se había observado que el vagón se quedaba atascado. Tras la aplicación de la cera, se observa que el vagón no se atasca, por lo que el vagón va mejor dirigido por la vía.

**Ilustración 18: Uso de la cera en la bifurcación**

**Alternativa G:** La maqueta de la que se parte tiene un potencia de 30%. A una potencia de 30% el trayecto se ve interrumpido debido a diversos problemas ya comentados. Por un lado, al llegar a la bifurcación los vagones se queda atascados puesto que a tal baja potencia no llega a hacer el giro para encaminarse al lado de la vía que le corresponde a cada vagón en la bifurcación (problema 5) [ilustración 19]. Por otro lado, debido a la poca potencia, al llegar al sensor de contacto no llega con el suficiente impulso, lo que provoca que le cueste pulsarlo (problema 6). Además, el funicular oscila y al oscilar choca con las varillas de la valla puesta alrededor de la vía, esto provoca que el funicular llegue a pararse (problema 3) [ilustración 20].

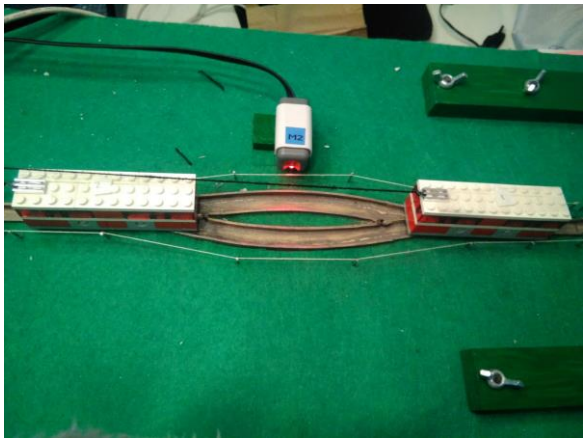


**Ilustración 19: Atasco en la bifurcación (P=30%)**

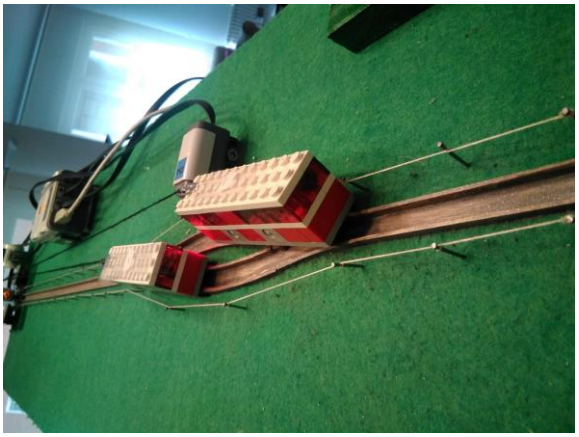


**Ilustración 20: Vagón volcado (P=30%)**

Tras estas observaciones se realiza una prueba incrementando la potencia al 70%, donde se comprueba que una excesiva potencia también causa problemas. Si bien es cierto que se han solucionado algunos problemas como la parada con el sensor de contacto, aparecen otros problemas. Debido a la gran potencia del vagón, este descarrila debido al aumento de la fuerza centrífuga. Además, a mayor potencia, mayor ruido hace el motor, lo cual resulta molesto.

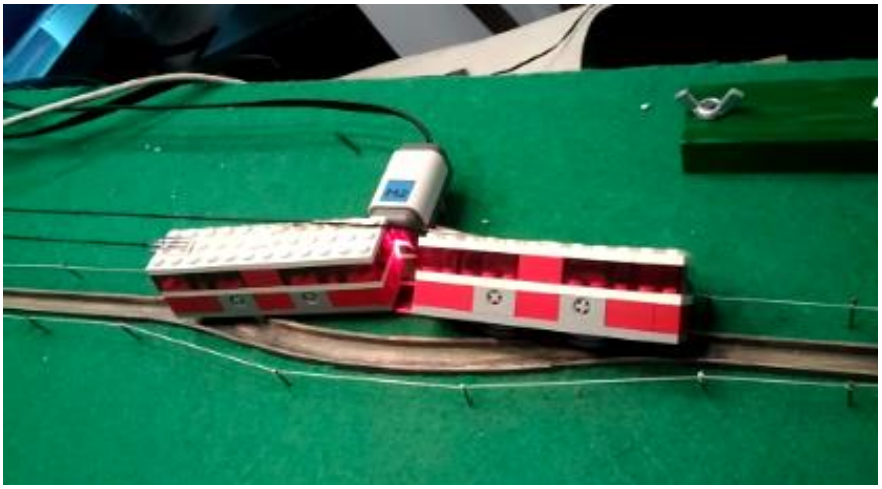


**Ilustración 21: Vagón descarrilado (P=70%)**

A continuación se muestran unos fotogramas del descarrilamiento:

**Ilustración 22: Fotograma descarrilamiento 1**

**Ilustración 23: Fotograma descarrilamiento 2**

**Ilustración 24: Fotograma descarrilamiento 3**

**Ilustración 25: Fotograma descarrilamiento 4**

Por lo tanto, una velocidad demasiado alta provoca que el vagón descarrile aunque el contacto con el sensor de la estación superior es correcto, mientras que a una velocidad pequeña, el vagón no llega al sensor de contacto ni a la bifurcación con el impulso suficiente, aunque tampoco descarrila. Sin embargo, tanto a una velocidad alta como baja el vagón oscila, si bien es cierto que la oscilación a velocidades bajas es menor. Por tanto a pesar de las oscilaciones del vagón se opta por una potencia intermedia: P=50%.

### 6.1.4. Solución propuesta

Durante las pruebas experimentales realizadas se ha podido comprobar la eficacia de las alternativas, de modo que las alternativas C y D quedan desestimadas mientras que todas las demás se llevan a cabo.

El equilibrado de la maqueta junto con el incremento de la velocidad y la adición de un sobresuelo en la bifurcación han hecho que los vagones oscilen menos (problemas 1 y 3). Además, el trayecto no se ve interrumpido en la bifurcación gracias al relleno de cera que encamina a los vagones y al incremento de la velocidad (problema 4 y 5). Además, gracias al incremento de la velocidad, el vagón llega con suficiente impulso al sensor de contacto (problema 6). En cuanto al problema que cuando se atasca el vagón, el motor sigue girando, este desaparece al realizar las modificaciones en el diseño de la vía, puesto que ya no se da ningún caso en el que el vagón se atasque (problema 2).

En la tabla 6 se muestra gráficamente las alternativas utilizadas para cada problema mostrando su eficacia, verde si es eficaz y rojo si no lo es.

**Tabla 6: Análisis de alternativas mejora diseño (2)**

| | Problema 1 | Problema 2 | Problema 3 | Problema 4 | Problema 5 | Problema 6 |
|---|---|---|---|---|---|---|
| Alternativa A | - | | | | | |
| Alternativa B | | - | | | - | |
| Alternativa C | | - | - | | | |
| Alternativa D | | | - | | | |
| Alternativa E | | | - | | | |
| Alternativa F | | | | - | | |
| Alternativa G | | | - | | - | - |

## 6.2. Detección de vagones en la bifurcación

### 6.2.1. Análisis del problema

En la maqueta para evitar la colisión de los vagones en el momento del cruce, hay una bifurcación. En dicha bifurcación es necesaria la presencia de un sensor que detecte la llegada de los vagones de modo que se accionen los motores que mueven las agujas. En la maqueta previa se utiliza un sensor de luz que se debe calibrar cada vez que el programa se ejecuta. Para ello antes de poner en marcha el programa principal, se realiza la medida mínima del sensor, sobre negro, y la máxima, sobre blanco. Tras la calibración cualquier medida sobre negro dará 0% mientras que sobre blanco dará 100%, eliminando así la influencia de la luz de ambiente que haya a la hora de poner en marcha el programa. El sensor está fijo en la maqueta, por lo que para hacer las medidas sobre blanco y negro se utiliza un cartón que se pone frente al sensor manualmente al calibrarlo. Esto supone que el proceso de calibración del sensor es manual.

### 6.2.2. Análisis de alternativas

Con el objetivo de automatizar este proceso se proponen las siguientes alternativas:

**Alternativa 1:** Utilizar sensor de luz suprimiendo su calibración.

**Alternativa 2:** Utilizar sensor de luz evitando el exceso de luminosidad en la cara activa del sensor.

**Alternativa 3:** Modificar los parámetros de funcionamiento del sensor de luz.

**Alternativa 4:** Sustituir el sensor de luz por un sensor infrarrojo.

El sensor de luz mide la luz ambiente y tiene un rango absoluto de 0 a 100, siendo 0 la total oscuridad y 100 luminosidad total. Los valores típicos suelen rondar entre 40 y 65. El método de calibración consiste en realizar la medida mínima del sensor sobre negro, y la máxima sobre blanco. Tras la calibración cualquier medida sobre negro dará 0 mientras que sobre blanco dará 100.

Los valores medidos por el sensor dependerán de la distancia a la que este colocado el sensor, así como de la luz ambiental. El mejor escenario en cuanto a la distancia del objeto se da cuando el sensor se

encuentra entre 5 y 8 mm del objeto. En cuanto a la luz ambiental, demasiada, como la luz solar directa, puede saturar el sensor, de modo que siempre devolverá el mismo valor: 100. Por lo tanto, se deberá calibrar del sensor de luz siempre y cuando la maqueta se vaya a poner en funcionamiento en escenarios en los que esté expuesto a mucha o poca luminosidad.

Ante las grandes variaciones de luz otra alternativa sería construir un buen blindaje que proteja al sensor de cualquier foco no deseable que pueda haber en el ambiente, de modo que las lecturas del sensor sean las mismas independientemente de la luminosidad del entorno.

Otra opción podría ser cambiar la escala a modo Raw, para así, obtener una mayor precisión. El modo predeterminado para el sensor de luz es el modo de porcentaje. El modo Raw tiene una escala 0-1023 mientras que el rango de modo porcentajes tiene una escala 0-100. Por ello, al ofrecer un rango de valores potenciales mayor nos proporcionara una mayor precisión.

No obstante, también cabe la posibilidad de otras opciones que no implican el uso de un sensor de luz. El sensor de luz no es el único sensor capaz de detectar un objeto. Otro sensor que podríamos utilizar es el sensor infrarrojo de HiTechnic (IRSeeker)[13].

El NXT IRSeeker V2 (Versión 2) es un detector infrarrojo de elementos múltiples que detecta señales infrarrojas de fuentes como el balón de fútbol HiTechnic IRBall, los controles remotos infrarrojos y la luz solar. El IRSeeker V2 funciona en 2 modos seleccionables:

- Modo modulado (AC): el sensor detectará señales IR moduladas, como las del IRTall HiTechnic o algunos controles remotos IR. En el modo Modulado, el sensor filtrará la mayoría de las otras señales de IR para disminuir la interferencia de las luces y la luz solar, por ejemplo. El sensor está sintonizado para señales de onda cuadrada a 1200Hz.
- Modo no modulado (DC): el sensor detectará señales IR no moduladas, como las IRBalls de estilo más antiguo o la luz solar.

El IRSeeker V2 utiliza técnicas avanzadas de procesamiento de señales digitales para filtrar las señales recibidas y seleccionar solo las señales requeridas. Una carcasa con un radio constante en la tapa del extremo incrementa el rendimiento direccional al minimizar la distorsión de las señales de luz que llegan al sensor.

Los valores de dirección del IRSeeker se muestran en la siguiente imagen, donde se puede observar que 1 indica que el objetivo infrarrojo se ha dejado atrás, mientras que 5 indica si el objetivo está directamente delante y 9 que el objetivo está a la derecha y detrás. Se devuelve un valor de 0 si no se detecta ninguna señal.

**Ilustración 26: Sensor infrarrojo**

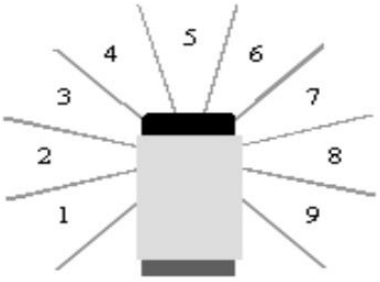



**Ilustración 27: Funcionamiento del sensor infrarrojo**

En la tabla 7 se resumen las principales diferencias entre el sensor de luz y el sensor infrarrojo:

**Tabla 7: Comparación sensores de luz e infrarrojo**

| | Sensor de luz | Sensor infrarrojo |
|---|---|---|
| **Nombre** | Sensor de color/luz LEGO MINDSTORMS EV3 | NXT IRSeeker V2 |
| **Precio** | 29,95€ | 48,55€ |
| **Medición** | Intensidad de luz<br>➔ Medición de la distancia relativa al objeto | Dirección y fuerza de la señal<br>➔ Medición de la distancia relativa al objeto<br>➔ Localización |
| **Salida** | Porcentaje (0-100) | Bool (0,1) |
| **2 modos** | | |
| **I** | Detecta luz ambiente | Modulado (AC): Detecta señales IR moduladas (controles remotos). |
| **II** | Emite luz y mide que tanto rebota o refleja en la superficie | No modulado (DC): Detecta señales IR no moduladas (IRBalls, luz solar) |
| | Sensible a la luz. Por ejemplo, a gran exposición solar el valor de salida sería siempre 100. | Disminuye la interferencia a las luces y la luz solar |

Como primera medida se decide comprobar si es necesaria la calibración del sensor de luz. Para comprobar que el programa funciona correctamente con diferentes iluminaciones, la maqueta será sometida a tres pruebas en diferentes entornos: expuesto a gran luz, luz ambiente neutra y penumbra.

### 6.2.3. Solución propuesta

Se realiza una prueba donde se comprueba el funcionamiento del funicular en distintos ambientes luminarios. De este modo, se pretende comprobar si el sensor de luz puede funcionar correctamente, es decir, detectar los vagones en la bifurcación, sin necesidad de ser calibrados cada vez que se ponga en marcha el funicular. El umbral del sensor de luz ha sido definido en 50 (valor modo porcentaje). Se

comprueba el funcionamiento del sensor de luz en tres ambientes: penumbra, luz ambiente y sobreexposición de luz.

**Ilustración 28: Ambiente de penumbra**

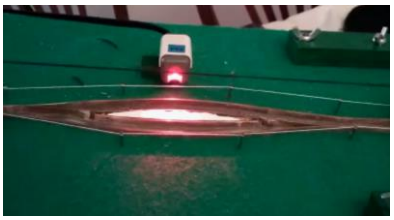


**Ilustración 29: Luz ambiente**

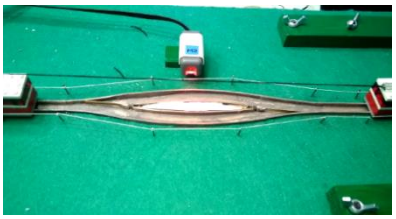


**Ilustración 30: Sobreexposición de luz**

Durante la prueba además se ha comprobado que independientemente de la luz del entorno, penumbra, ambiente o sobreexposición, si existe un foco de luz de incidencia directa el sensor de luz no funcionaría correctamente debido a la foto sensibilidad del sensor. No obstante esta situación se dará en raras ocasiones.

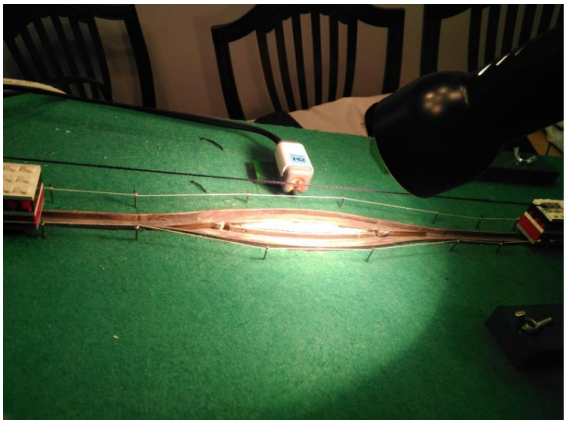


**Ilustración 31: Foco directo**

Tras la realización de estas pruebas se concluye que no hace falta calibrar el sensor de luz, ya que se ha comprobado que este funciona tanto en ambiente de penumbra como en luz ambiente y sobreexposición de luz. No obstante, el sensor de luz no debe estar expuesto a un foco directo, de ser así el sensor de luz no funcionaría correctamente.

## 6.3. Regulación de la velocidad

### 6.3.1. Análisis del problema

En la maqueta de la que se parte el control del funicular es en bucle abierto, es decir, es un sistema en el que solo actúa el proceso sobre la señal de entrada y da como resultado una señal de salida independiente de la entrada. Al no haber realimentación, la salida no se compara con la entrada por lo que nada asegura su estabilidad ante una perturbación. Es decir, por ejemplo, el funicular a una potencia del 50% en condiciones normales implicaría una velocidad de 62,5 rpm. Sin embargo, si se le aplicase una carga mayor, por ejemplo añadiendo peso al vagón, el motor a una potencia del 50% tendría una velocidad

menor que 62,5 rpm. Puesto que no se tiene control de la velocidad de salida esto no sería regulable en un sistema en bucle abierto.


**Ilustración 32: Esquema control bucle abierto**

### 6.3.2. Análisis de alternativas

Con el objetivo de mejorar el funcionamiento de la maqueta se proponen **el diseño e implementación de un controlador** que regule la velocidad. Con un sistema con realimentación, en bucle cerrado, la salida se comparará con la entrada de tal modo que la velocidad pueda ser regulada. En bucle abierto si se incrementase la carga del vagón, el sistema se vería afectado en la velocidad. Sin embargo en bucle abierto esta disminución de velocidad seria percibida por un sensor y se incrementaría la potencia de tal modo que la velocidad de salida sea la de consigna.

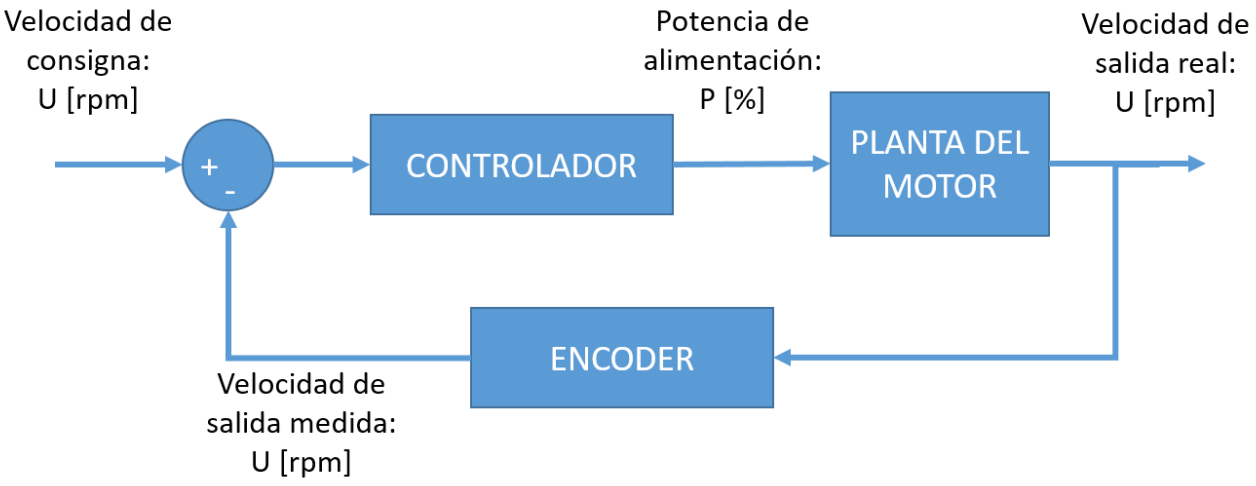



**Ilustración 33: Esquema control realimentado bucle cerrado (1)**

Puesto que en este caso el encoder está implementado en el motor el diagrama real sería el siguiente, donde el sistema incluye tanto el encoder como el motor y los vagones:

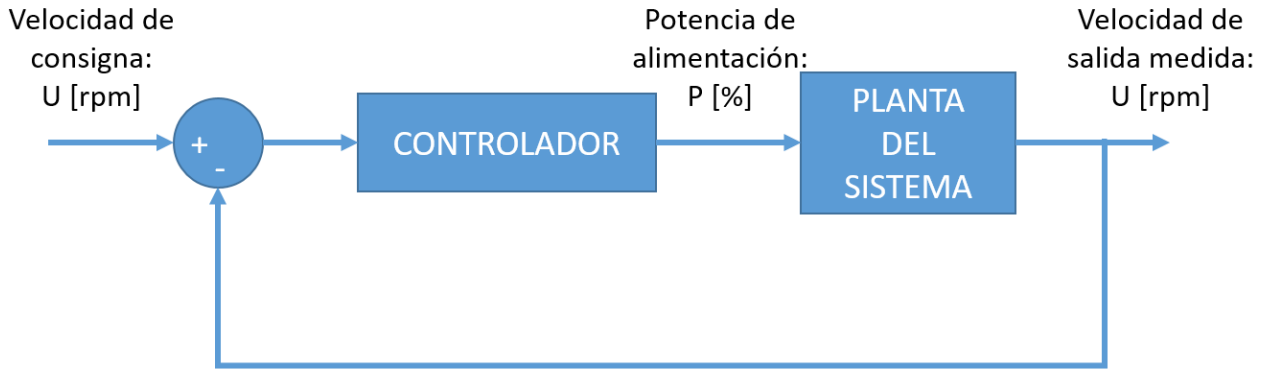



**Ilustración 34: Esquema control realimentado bucle cerrado (2)**

Para este proyecto el controlador utilizado es un PID (acción proporcional, integradora y derivativa). A continuación, se realiza un estudio de alternativas para encontrar la herramienta más adecuada para el diseño del controlador.

### 6.3.2.2. Diseño del controlador

**Alternativa 1:** Experimentalmente ensayo-error

Mediante el método de Ziegler-Nichols, experimentalmente se podría obtener los valores de los parámetros proporcional (Kp), integrador (Ki) y derivativo (Kd).

Este método se realiza en lazo cerrado. Para determinar dichos valores se siguen los siguientes pasos: (Ziegler y Nichols)

1. Se inicializan los valores I(t) y D(t) a cero.
2. Se incrementa Kp hasta que la salida oscile, ese será el punto crítico. A la ganancia para la que el bucle de control tiene oscilaciones estables y consistentes, se le denomina ultima ganancia, Ku, mientras que al periodo en el que se alcanza la inestabilidad se le denomina periodo de oscilación, Tu. Se fija la Kp a la mitad de ese valor.
3. Se incrementa Ki para reducir el error en estacionario, teniendo en cuenta que incrementarlo mucho puede provocar inestabilidad.
4. Se incrementa Kd hasta obtener un lazo lo suficientemente rápido tras una variación de la carga.

Tras los dos primeros pasos, una vez obtenidos la Ku, se obtienen todos los parámetros necesarios con la tabla 8.

**Tabla 8: Método de Ziegler y Nichols**

| Tipo de Control | Kp | Ti | Td | Ki | Kd |
|---|---|---|---|---|---|
| P | 0,5*Ku | - | - | - | - |
| PI | 0,45*Ku | Tu/1,2 | - | 0,54Ku/Tu | - |
| PD | 0,*Ku | - | Tu/8 | - | Ku*Tu/10 |
| PID | 0,6*Ku | Tu/2 | Tu/8 | 1,2Ku/Tu | 3*Ku*Tu/40 |

**Alternativa 2:** Diseño analítico

Durante el grado tanto en la asignatura de Automatización y Control (3º curso) como en Control por Computador (4º curso) se ha estudiado el diseño de controladores PID, de modo que se puede diseñar un controlador mediante el lugar de las raíces con los conocimientos adquiridos en dichas asignaturas.

**Alternativa 3:** Herramientas computacionales

En el software de Matlab, en el entorno de programación de Simulink, utilizando los bloques PID Controller, hay una herramienta llamada AutoTune. Autotune puede ajustar automáticamente los controladores PID para lograr un diseño del sistema que cumpla con los requisitos de diseño, incluso para los modelos de planta que los métodos tradicionales basados en reglas no pueden manejar bien.

De las tres alternativas propuestas se elegirá la que mejor se adecue a los requisitos del controlador que se especificarán más adelante y la herramienta que sea sencilla de manejar.

Se opta por realizar el diseño del controlador tanto mediante diseño analítico mediante el lugar de las raíces como mediante la herramienta de Matlab.

Tras el diseño del controlador, este tendrá que ser implementado en el programa del funicular. La implementación del programa se puede realizar mediante los comandos de Lego o bien programándolo. A continuación se realiza un análisis de alternativas para elegir el modo más adecuado para implementar el controlador.

#### 6.3.2.3. Implementación del controlador

**Alternativa 1:** Comandos de Lego

Para la implementación del PID, el software BricxCC, cuenta con el siguiente comando:

```
OnFwdRegPID (outputs,pwr,regmode,p,i,d)
```

En cuanto a los parámetros P,I y D, los valores a introducir son los de la siguiente expresión:

$$U(t) = P(t) + I(t) + D(t)$$

Donde:

$$P(t) = k_p * e(t)$$

$$I(t) = k_i * \int_0^t e(t)dt$$

$$D(t) = k_d * \frac{de(t)}{dt}$$

**Alternativa 2:** Programación

El programa para la implementación del controlador deberá seguir la siguiente estructura:

1. Definición de variables, es decir, decir de qué tipo son las variables utilizadas en el programa (long, int, float…).
2. Definición de constantes. En este caso las contantes serán las ganancias (Kp,Ki,Kd), el periodo y la velocidad de referencia.

3. Inicialización de las variables. Las variables, como por ejemplo la diferencia entre la velocidad y la velocidad de referencia en un principio, deben inicializare a cero.
4. Bucle iterativo del controlador, que se ejecuta cada T segundos. Para realizar el bucle, al inicio debe escribirse `While(TRUE)`, o bien, `While(1)`.
5. Actualización de variables, por ejemplo el error anterior para el siguiente bucle será el error actual. Dicha sentencia debe estar dentro del bucle. Tras esto, se debe cerrar el bucle.

En este proyecto, se opta por la implementación del PID mediante programación, con el fin de aplicar los conocimientos adquiridos durante el grado.

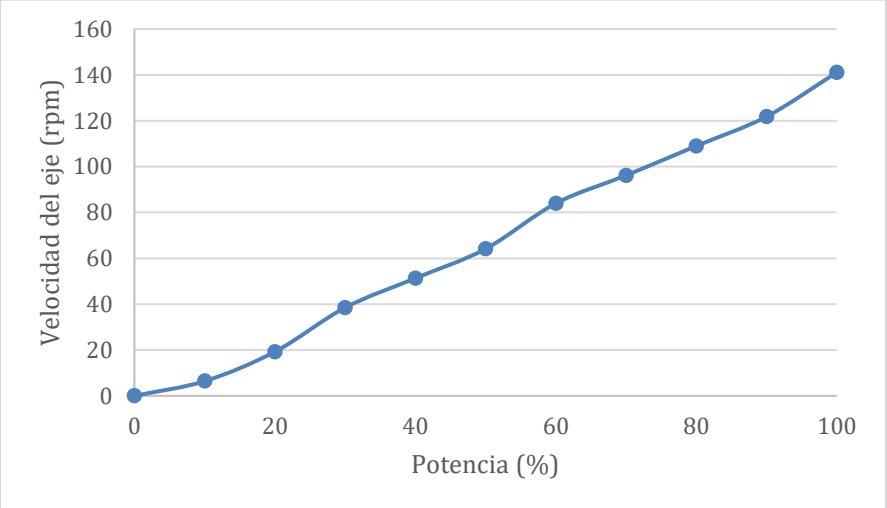
## 6.3.3. Solución propuesta

### 6.3.3.1. Diseño del controlador

Para el diseño del controlador, en primer lugar se realiza la caracterización del sistema analizando la variación de la velocidad en el tiempo a través de un registro de datos (Anexo 2). Una vez caracterizada la planta del sistema se procederá a la sintonización del controlador. Para ello, en primer lugar, se debe seleccionar el algoritmo: P, PI, PD o PID. Tras ello, se seleccionarán los parámetros ($K_c$,$T_i$,$T_d$), para lo que se tendrán en cuenta las especificaciones de las características de respuesta en bucle cerrado.

*Caracterización de la planta*

La planta del sistema en este proyecto incluye tanto el motor como el encoder y los vagones. La caracterización de la planta se realiza experimentalmente a una potencia de 50%. Para realizar este método experimental de obtención de la planta en primer lugar se debe determinar si el sistema es lineal. Para ello realizamos una prueba donde se obtiene la gráfica 1, donde se observa la variación de la velocidad en función de la potencia.




**Gráfica 1: Variación de la velocidad con la potencia**

Se considera que el sistema es lineal dentro del rango de potencia de (0,100). Además, cabe considerar que el motor no puede entregar más potencia del 100% que corresponde a 140 rpm, por lo que es

conveniente complementar el modelo con un elemento saturador de modo que la señal del controlador nunca supere ese valor de potencia. El límite inferior de velocidad corresponderá a 140 rpm en el sentido contrario.

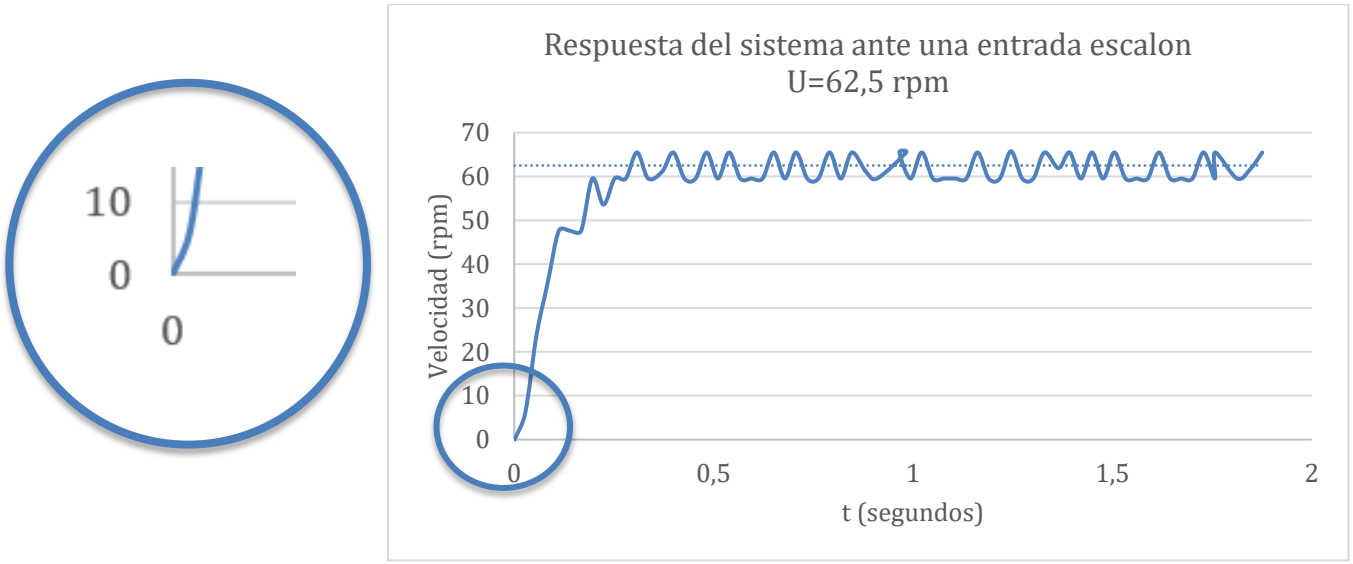
Para la caracterización de la planta será necesario conocer la velocidad del funicular en cada instante. Para ello, será necesario medir los grados girados por el motor así como el tiempo transcurrido. Como ya se ha mencionado anteriormente, el servomotor, además de incluir un motor eléctrico contiene un encoder de posición que nos permitirá, haciendo el cálculo correspondiente, saber a qué velocidad está girando el motor. El encoder incorpora una ruleta que deja pasar o bloquea la luz, con una precisión de un grado. Además, el firmware incorporado en NXT tiene un reloj interno de 32 bits que cuenta en intervalos de 1 ms. Cada milisegundo es un tick.


**Ilustración 35: Variación de ángulo del motor**

Por lo tanto, mediante la medición de grados girados y tiempo transcurrido en diferentes instantes, el cociente entre ambas diferencias será la velocidad. Dicha velocidad será en grados/milisegundos, para pasarlo a revoluciones por minuto (rpm) habrá que multiplicar por 166,667.

$$Velocidad\ [rpm] = \frac{\Delta\ ángulo\ [grados]}{\Delta\ tiempo\ [milisegundos]} * \frac{1000\ ms}{1\ s} * \frac{60\ s}{1\ min} * \frac{1\ rev}{360\ grados} = 166{,}667 rpm$$

La entrada a la que el sistema se somete es una entrada escalón U=62,5 rpm. Para obtener la variación de la velocidad en el tiempo se realiza un registro en el que se mide la velocidad de la manera explicada anteriormente en cada instante. Con el programa de registro de datos incluido en el anexo 2 se obtiene la gráfica 2:




**Gráfica 2: Variación de la velocidad en el tiempo**

En la gráfica de la respuesta del sistema ante una entrada escalón se observa un mínimo retardo que se considera irrelevante a efectos de modelado. Se observa también un rizado considerable en la respuesta, posiblemente debido al ruido introducido por los engranajes internos que realizan la reducción de velocidad interna del motor. Bajo estas consideraciones la planta estudiada se puede aproximar a un sistema de primer orden.

Esta consideración es lógica puesto que normalmente los motores eléctricos de continua tienen dos partes: eléctrica y mecánica, y consecuentemente dos polos. El polo correspondiente a la parte eléctrica tiene una constante de tiempo mucho menor que el polo correspondiente a la parte mecánica, más lenta, por lo que resulta razonable despreciar el polo rápido (el eléctrico) y asumir que la respuesta del sistema se comporta aproximadamente como un sistema de primer orden con un único polo debido a la parte mecánica.

La función de trasferencia de un sistema de primer orden tiene la siguiente forma:

$$G_p(s) = \frac{K}{1 + \tau s}$$

Donde:

K es la ganancia en estado estacionario, para calcularlo:

$$K = \frac{\Delta y}{\Delta u} = \frac{62.5}{50} = 1{,}25\ \frac{rpm}{\%}$$

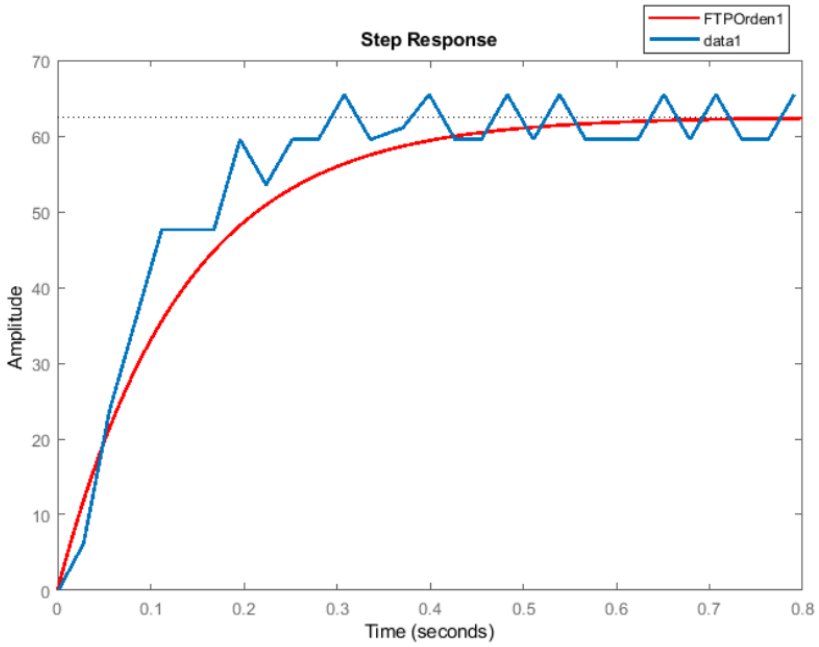
τ es la constante de tiempo del sistema, mide la rapidez de sistema. Para calcular el valor de τ trazamos la tangente a la máxima pendiente hasta que interseccione con el valor final de la salida, en este caso 62,5.

$$\tau = 0{,}1330s$$

La función de trasferencia para una aproximación a un sistema de primer orden es:

$$G_p(s) = \frac{1{,}25}{1 + 0{,}1330s}$$

En la gráfica 3 podemos comparar la respuesta a la entrada escalón real (azul) con la aproximación a un sistema de primer orden (rojo):




**Gráfica 3: Comparación de la respuesta del sistema ante un escalón**

*Selección del algoritmo*

El requerimiento del sistema de control es un error cero a entrada escalón en error estacionario. El sistema es tipo 0 por lo que es necesario aumentarlo a tipo 1, es decir, se precisa de acción integradora, por lo que el abanico se restringe a un PI o un PID. En cuanto a la acción derivativa, no sería conveniente puesto que con toda probabilidad será muy sensible al ruido debido a la construcción del motor, principalmente hecho con piezas plásticas en sus engranajes y en el encoder.

*Selección de parámetros*

Para la selección de parámetros, en primer lugar, se deben dar los requerimientos exigidos para un óptimo funcionamiento del sistema.

La sintonización de parámetros del PI se realizará en primer lugar analíticamente y en segundo lugar mediante la herramienta que proporciona Matlab, con propósitos de comparación.

### Diseño analítico

Siendo la planta:

$$G_p(s) = \frac{1{,}25}{1 + 0{,}1330s}$$

Y siendo la función de transferencia del controlador PI:

$$G_{PI}(s) = K_c\left[1 + \frac{1}{sT_i}\right] = \frac{K_c}{T_i}\left[\frac{sT_i + 1}{s}\right]$$

Colocando el cero de diseño sobre el polo de la planta: $T_i$=0,1330

El sistema en bucle abierto queda:

$$G_{BA} = \frac{K_c}{0{,}1330}\left[\frac{1 + 0{,}1330s}{s}\right]\left[\frac{1{,}25}{(1 + 0{,}1330s)}\right] = \frac{9{,}4K_c}{s}$$

El sistema en bucle cerrado queda:

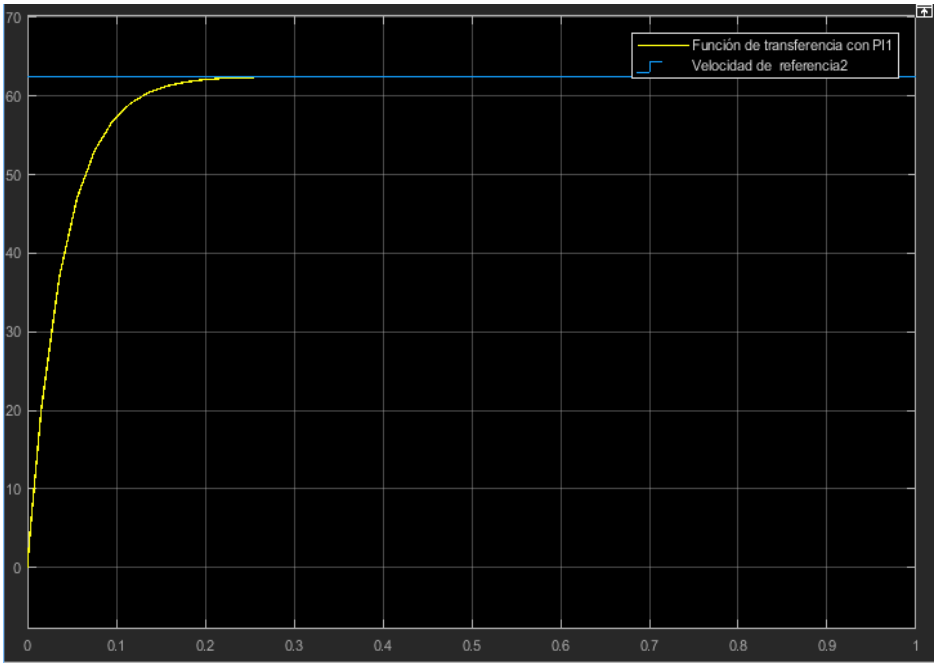
$$G_{BC} = \frac{1}{\frac{s}{9{,}4K_c} + 1}$$

Es decir, como era de esperar, la ganancia del sistema es la unidad (no tendrá error en régimen permanente a entrada escalón, y la constante de tiempo queda en función de Kc, lo cual nos permite seleccionar un valor para este parámetro lo mayor posible para que el sistema alcance lo antes posible el régimen permanente, y a la vez no sature el motor.

Para un sistema de primer orden el tiempo de establecimiento (2%):

$$t_{ss} = 4\tau = \frac{4}{9{,}4K_c}$$

Para determinar el valor de $K_c$ de modo que el sistema responda lo más rápido posible sin saturar el motor se realizan simulaciones con distintos valores de $K_c$. Tras las simulaciones, con un valor de $K_c$=2,66 se obtiene la siguiente respuesta del sistema con un tiempo de establecimiento de 0,16s:




**Gráfica 4: Respuesta del sistema con el controlador diseñado analíticamente**

Por lo tanto se ha obtenido el siguiente controlador:

$$G_{PI}(s) = K_c + \frac{K_c}{T_i\ s} = P + \frac{I}{s}$$

Dónde: P=2,66

I=20

$$G_{PI}(s) = 2{,}66 + \frac{20}{s}$$

Herramienta computacional, Matlab

Con la herramienta de Matlab, en el entorno de Simulink, una vez obtenido el modelo de la planta del sistema, se realizan simulaciones donde el sistema de control con realimentación unitaria constará principalmente de tres bloques. En primer lugar, la excitación del sistema que será una entrada escalón. Tras ello se añadirá un controlador continuo PI, seguido de la función de transferencia.

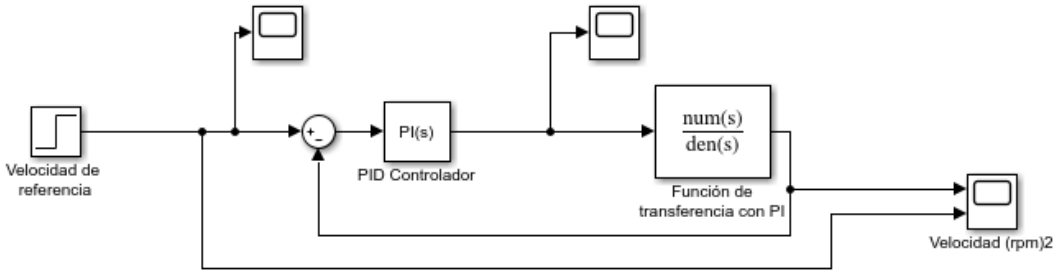
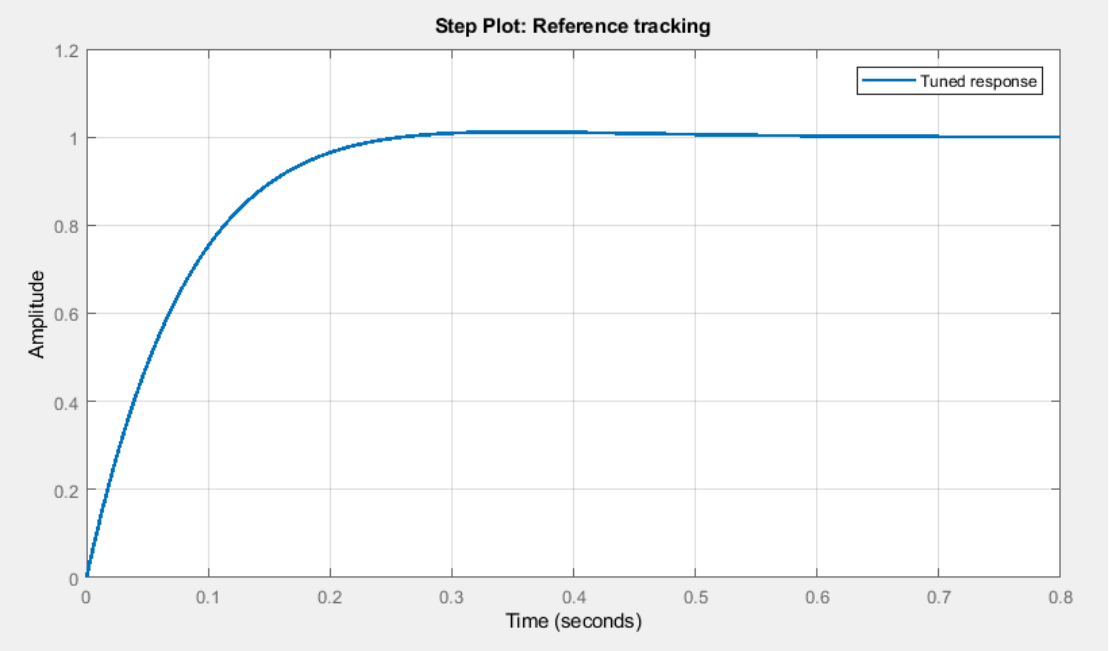



**Ilustración 36: Esquema del sistema en Simulink**

Para la elección de los parámetros del controlador se utiliza la herramienta de AutoTune, que calcula automáticamente los parámetros óptimos para el controlador. A continuación se realiza un ajuste manual.


**Ilustración 37: Variación del tiempo de respuesta y el comportamiento del sistema**

Tras los ajustes realizados la respuesta al sistema se muestra en la gráfica 5:




**Gráfica 5: Respuesta del sistema con el controlador tras Autotune**

Aplicando este controlador se resumen los principales parámetros del sistema en la tabla 9:

**Tabla 9: Parámetros del sistema**

| | |
|---|---|
| **Tiempo de subida** | 0.145 segundos |
| **Tiempo de establecimiento** | 0.219 segundos |
| **Sobre impulso** | 1.18% |
| **Valor de pico** | 1.01 |

De este modo se ha llegado al siguiente controlador PI:

$$G_c = P + I\frac{1}{s}$$

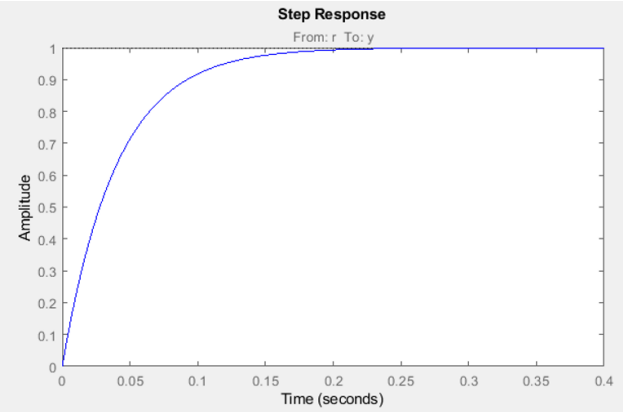
Dónde: P=1.3497

I=12.335

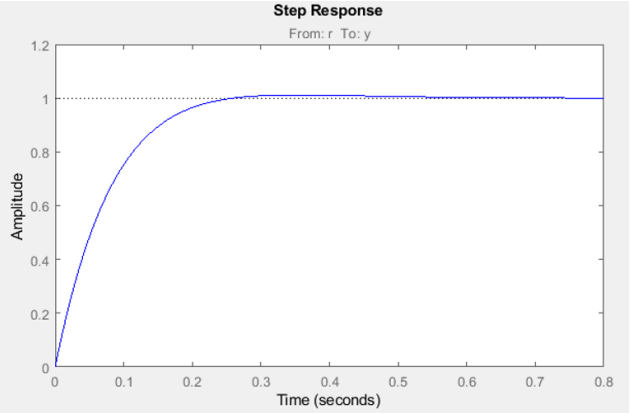
$$G_c = 1.3497 + 12.335\frac{1}{s}$$

### Comparación

Tras el diseño de los dos controladores, mediante diseño analítico y mediante la herramienta AutoTune se compara la respuesta del sistema con cada controlador. En las gráficas 6 y 7 se pueden observar las respuestas del sistema a una entrada escalón unitaria, a la izquierda con el controlador diseñado analíticamente y a la derecha con AutoTune. Se puede observar como el controlador diseñado analíticamente responde más rápido.



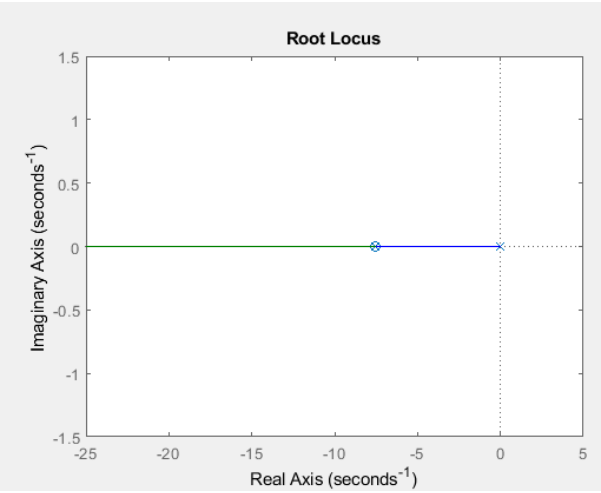

**Gráfica 6: Respuesta a escalón. Diseño analítico**



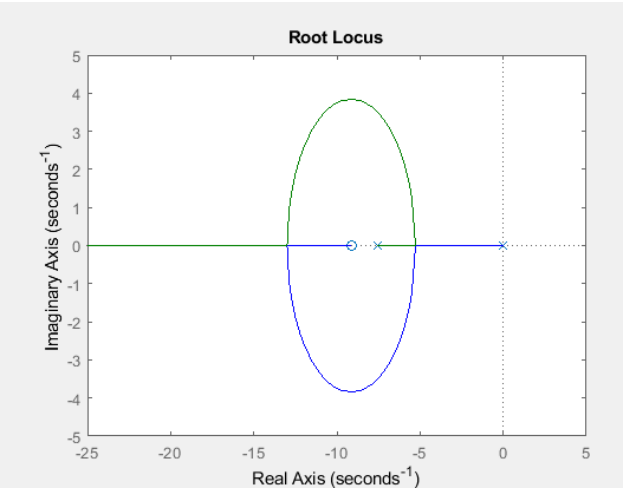

**Gráfica 7: Respuesta a escalón. AutoTune**

Además, con el controlador de AutoTune se puede apreciar un pequeño sobreimpulso (Mp=1,18%), mientas que en el de diseño analítico no lo hay. Esto se debe a que el sistema con el controlador diseñado analíticamente el polo de la planta del sistema ha sido cancelado con el cero del controlador de tal modo que el sistema solo tiene un polo. Sin embargo, tal y como se puede observar en la gráfica 9, el sistema con el controlador diseñado con AutoTune ha situado el cero del controlador en otra posición distinta, dando lugar a un sistema de segundo orden en lazo cerrado y generando un pequeño sobreimpulso.

No obstante, el aspecto más importante es la eliminación en error en régimen permanente que por supuesto se logra con ambos diseños.




Gráfica 8: Lugar de las raíces. Diseño analítico




Gráfica 9: Lugar de las raíces. AutoTune

Tabla 10: Comparación controladores

| Diseño analítico | AutoTune |
|---|---|
| $t_{ss}$=0,16 segundos | $t_{ss}$=0,22 segundos |
| Sin sobreimpulso | Mp=1,18% |

Puesto que el controlador diseñado analíticamente ofrece una respuesta más rápida y sin sobreimpulso en teoría es el mejor controlador para regular la velocidad del sistema.

Además de los parámetros de la función del controlador, se debe determinar la frecuencia de muestreo. Es obvio, que cuanto más pequeña sea la frecuencia, el control del sistema será mejor. Para un buen controlador, el periodo mínimo de muestreo sería una décima parta de la constante de tiempo de la función de transferencia del sistema ($\tau = 0{,}1330$). Por lo tanto:

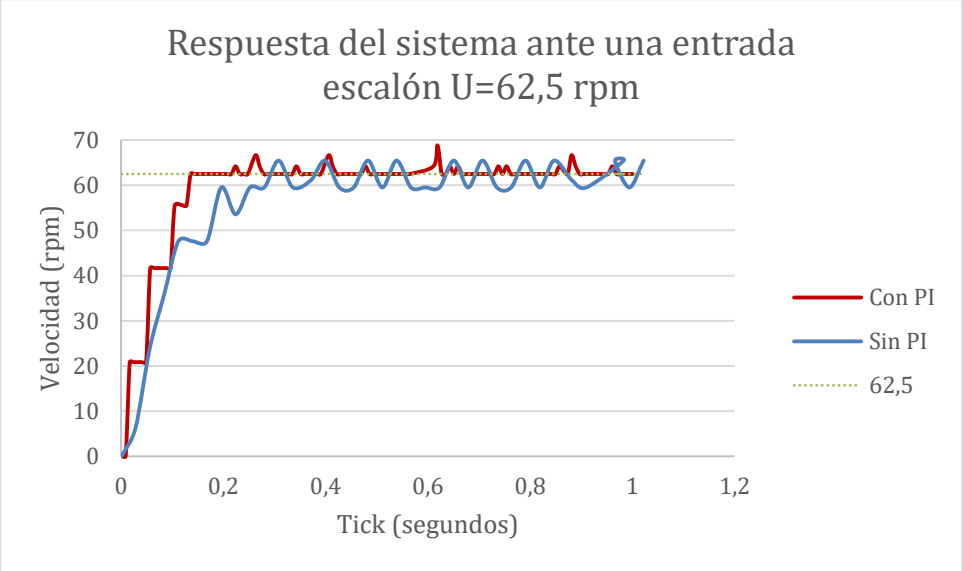
$$T \leq \frac{0{,}1330}{10} \approx 10ms$$

Para asegurar un buen control, se opta por un periodo de 5ms.

### 6.3.3.2. Implementación del controlador

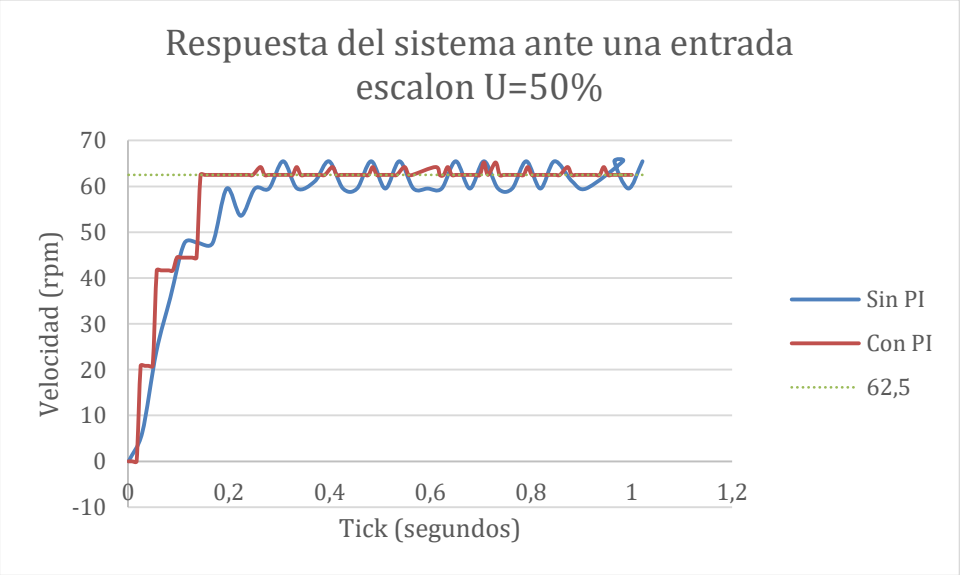
Una vez obtenidos los parámetros necesarios, se codifica. El código puede verse en el anexo 2.

En primer lugar se implementa el controlador diseñado analíticamente.




**Gráfica 10: Respuesta real del sistema tras la implementación del controlador. Diseño analítico**

En segundo lugar se implementa el controlador diseñado con AutoTune.




**Gráfica 11: Respuesta real del sistema tras la implementación del controlador. AutoTune**

Tal y como se puede observar en las gráficas 10 y 11, aunque en teoría el sistema con el controlador diseñado analíticamente era más rápido, en la práctica el tiempo de establecimiento de la respuesta del sistema con ambos controladores es similar. De hecho, el controlador diseñado con AutoTune es menos

sensible al ruido introducido por los engranajes internos, por lo que finalmente se decide utilizar el controlador diseñado con AutoTune.

## 6.4. Freno del funicular

### 6.4.1. Análisis del problema

En la maqueta original de la que se parte, para frenar el sistema funicular cuando sea preciso, debe pulsarse el botón exit del ladrillo (tecla inferior), lo que supone dejar de ejecutar el programa.

### 6.4.2. Solución propuesta

Por ello, en esta mejora del sistema se ha decidido implementar una función que permite frenar el vagón cuando se desea. Para ello se ha decidido diferenciar dos tipos de frenos: un freno brusco para emergencias y un freno suave para una parada convencional.

Parada de emergencia: Para activar la parada de emergencia se debe pulsar el botón central (naranja). Tras pulsar ese botón el funicular frena bruscamente apagando el motor. En caso de querer volver a arrancar, tal y como lo indica la pantalla se debe pulsar la tecla derecha. El mensaje mostrado en pantalla es el siguiente:

```
"PARADA EMERGENCIA
 Para continuar…
  tecla derecha
  <distancia>"
```

Parada convencional: Para realizar una parada normal, por el motivo que fuese, se debe pulsar la tecla izquierda. Tras pulsar este botón el motor se queda en punto muerto, lo que hace que la parada sea más suave que la anterior. Para volver a la marcha del funicular se debe pulsar la tecla derecha. El mensaje mostrado en pantalla durante la parada es el siguiente:

```
    "PARADA
Para continuar…
  tecla derecha
  <distancia>"
```

## 6.5. Funciones de seguridad

### 6.5.1. Análisis del problema

Además de solucionar los problemas observados en la maqueta, con el objetivo de mejorar el funcionamiento de la maqueta se propone implementar algunas funciones adicionales de seguridad de tal modo que sea posible: la localización de los vagones, la detección de paradas accidentales y la detección de fallo en los frenos. Para el desarrollo de dichas funciones resulta útil el sensor ultrasónico.

El sensor ultrasónico genera ondas de sonido de alta frecuencia y lee los retardos de sus ecos para detectar y medir la distancia de objetos. Este sensor es de gran utilidad para la consecución de varios de los objetivos, ya que además de localizar los vagones en cada momento se podría utilizar para la detección

de paradas accidentales de los vagones y detección de fallo en los frenos. Sin embargo, esto genera una cuestión, dónde poner el sensor ultrasónico. La estación superior sería el lugar idóneo pero puesto que es donde está colocado el sensor de contacto es imposible, por lo que habrá que colocarlo en la estación inferior. Esto supone un problema puesto que el cable de Lego que une los sensores y servomotores con el ladrillo mide 50cm mientras que la distancia desde el ladrillo a la estación inferior supera el metro.

### 6.5.2. Análisis de alternativas

Para la fabricación de un RJ12 para NXT lo suficientemente extenso se consideran las siguientes alternativas:

**Alternativa 1:** Soldadura

Este método consiste en unir dos cables con estaño utilizando un soldador para obtener un cable con una largura aceptable

- Ventajas: Proceso sencillo y de bajo coste
- Desventajas: Aspecto poco estético y poca resistencia a los tirones

Para realizar este método se necesitaría:

- 2 x Cable RJ12 NXT
- 1 x Alambre de estaño
- 1x Soldador
- 6 x Cable

**Alternativa 2:** Fabricar cable RJ12 usando crimpadora

Para ello, habrá que comprar conectores RJ12 específicos y un cable de 6 pines. Dichos conectores son difíciles de conseguir ya que son muy específicos. Se venden en grandes cantidades y tan solo pueden obtenerse por Internet. Además, debido a la singularidad de la pieza, para unir el cable con el conector se necesitará una crimpadora especial. En la siguiente figura se puede observar la diferencia entre un cable RJ12 estándar con un RJ12 específico para NXT.

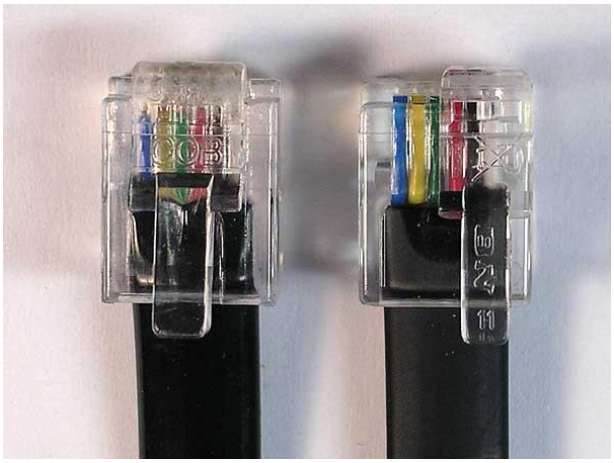


**Ilustración 38: Comparación RJ12 Lego y estándar**

Para realizar este método se necesitaría:

- 1 x Cable RJ12 NXT
- 6 x Cable
- 2 x Conector RJ12 específico
- 1 x Crimpadora

**Alternativa 3:** Adaptar conector RJ12

Este método consiste en modificar la estructura del conector RJ12 estándar. Como se puede observar la pestaña del RJ12 es más ancha que la de Lego y está centrada mientras que la de Lego se encuentra a un lado. En primer lugar, se corta la pestaña con una sierra y se lima hasta que encaje correctamente con el RJ12 hembra del sensor y del ladrillo. Una vez la pestaña encaje correctamente se pega la pestaña en la parte derecha del conector.

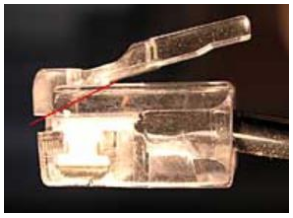


**Ilustración 39: Corte conector RJ12**

**Ilustración 40: Pestaña RJ12**

Para la realización de este método se necesitaría el siguiente material:

- 1 x Cable RJ12 estándar
- 1 x Sierra
- 1 x Lima
- 1 x Pegamento

**Alternativa 4:** Extender cable mediante conectores y acopladores RJ12

Para este método en primer lugar, se corta un cable RJ12 para NXT por la mitad y se añade un conector RJ12 (estándar) a las dos partes que se han cortado utilizando una crimpadora.

De este modo utilizando dos acopladores se podrán unir los dos cables obtenidos con un cable RJ12 formando así un cable más extenso. El resultado de los dos cables que se obtendrían por este método se observan en la siguiente imagen: (cables RJ12/RJ12)

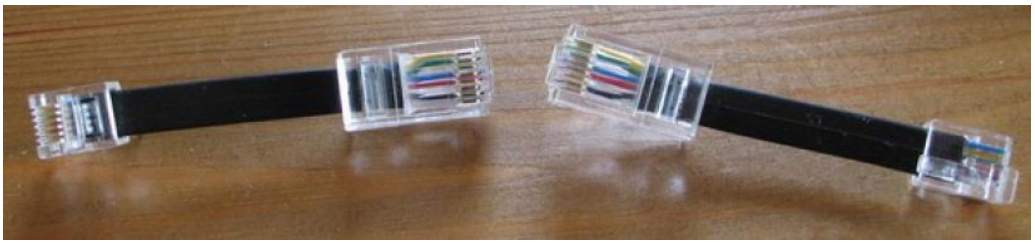


**Ilustración 41: Medios cables RJ12 Lego**

Para realizar esto sería necesario el siguiente material:

- 1 x Cable RJ12 NXT
- 2 x Conector RJ12 Macho
- 2 x Acoplador H/H RJ12
- 1 x Crimpadora
- 1 x Cable RJ12

A continuación, se incluye una tabla en la que se puntúa del 1 al 10 cada alternativa según cuatro criterios. No todos los criterios tienen la misma relevancia por lo que se les ha asignado una ponderación.

**Tabla 11: Análisis de alternativas extensión cable RJ12**

| ***CRITERIO*** | **Estética** | **Coste** | **Dificultad** | **Deterioro** | **TOTAL** |
|---|---|---|---|---|---|
| *Ponderación* | 0,1 | 0,2 | 0,4 | 0,3 | 1 |
| *Soldadura* | 2 | 8 | 7 | 1 | 4,9 |
| *Fabricar cable RJ12* | 8 | 6 | 4 | 5 | 5,1 |
| *Adaptar cable RJ12* | 6 | 7 | 4 | 3 | 4,5 |
| *Extender cable RJ12* | 7 | 5 | 6 | 5 | 5,6 |

Ante los resultados obtenidos, se elige la alternativa de extender un cable mediante conectores y acopladores RJ12.

### 6.5.3. Solución propuesta

El sensor ultrasónico se ubica en la estación inferior de la maqueta. Puesto que el cable proporcionado por Lego mide 0,5 metros y la distancia desde el brick hasta la estación inferior supera el metro, para su conexión es necesario alargar el cable. Para ello, en primer lugar se corta un cable Lego, pelándolo por

los extremos, de tal modo que obtenemos dos mitades de cable como se observa en la ilustración 42. A continuación, en cada una de las mitades del cable Lego, con la ayuda de una crimpadora, se añade un conector RJ12. Una vez realizado esto, utilizando un acoplador RJ12/RJ12, se puede conectar un cable RJ12 de la extensión necesaria que une ambas mitades del cable de Lego, obteniendo así un cable de la longitud que se desea, como se puede ver en la ilustración 46. A continuación se muestran algunas imágenes del proceso.

**Ilustración 42: Corte del cable RJ12**

**Ilustración 43: Cable RJ12 cortado**

**Ilustración 44: Cables RJ12 crimpados y crimpadora**

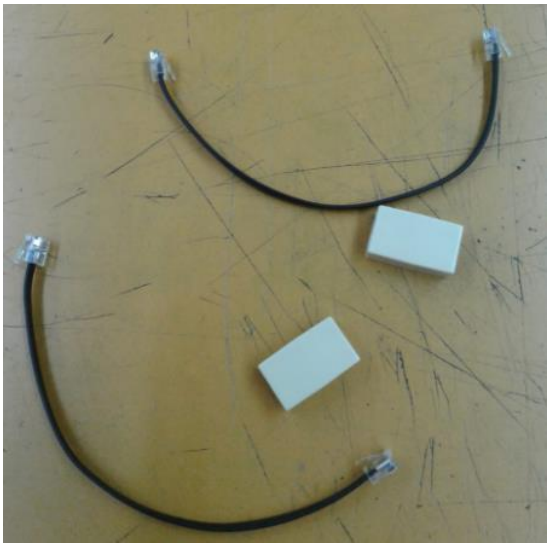


**Ilustración 45: Cables crimpados junto con los acopladores**

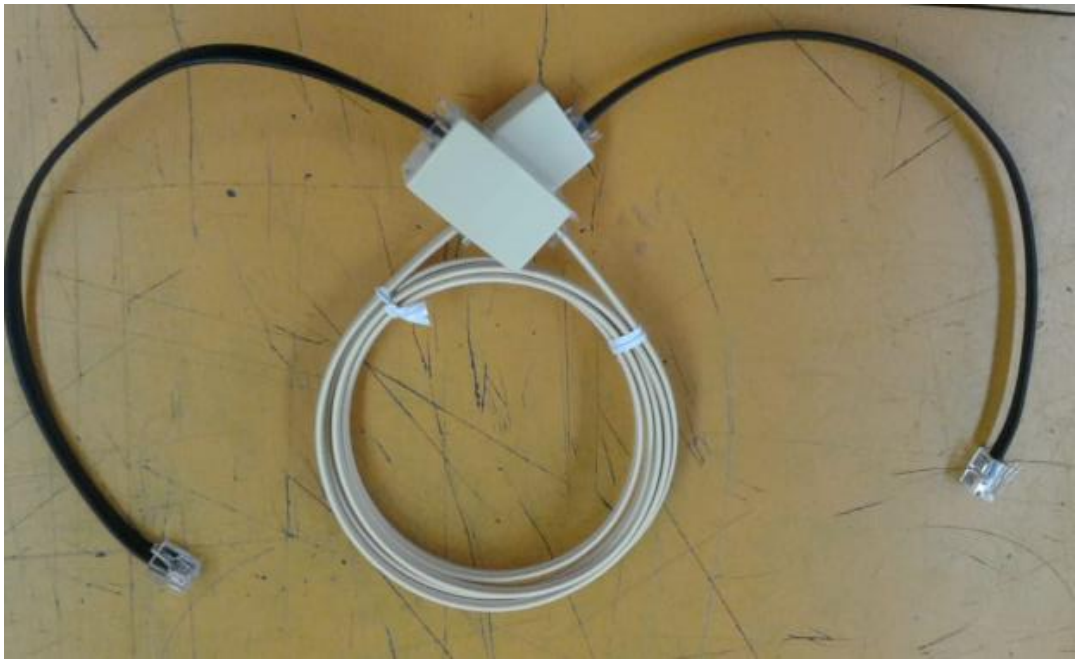


**Ilustración 46: Cable RJ12 extendido**

Como ya se ha mencionado, el sensor ultrasónico mide distancias. Envía ondas de sonido ultrasónicas que rebotan con un objeto delante de él y luego regresan. Calcula la distancia en función del tiempo que tarde la onda en volver a él. Para ello tiene dos objetivos, uno que actua como emisor y otro como receptor. Para un reconocimento de los obstáculos satisfactorio, estos deben estar dispuestos horizontalmente. Además, a la hora de programar las funciones se ha tenido en cuenta que la distancia máxima que puede medir es 255 cm con una precisión de 3 cm. Se considera que hasta los 5 cm no puede detectar, es punto ciego.

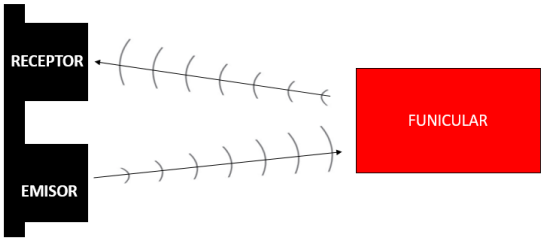



**Ilustración 47: Funcionamiento del sensor ultrasónico**

El sensor resulta más ancho que la abertura de la estación, es por ello que ha habido que realizar modificaciones en el diseño de la estación inferior para ampliar dicha abertura. De lo contrario, el sensor siempre detectaría la distancia a la que se encuentra la pared de la estación, puesto que ese será el primer obstáculo con el que se encontrará siempre.

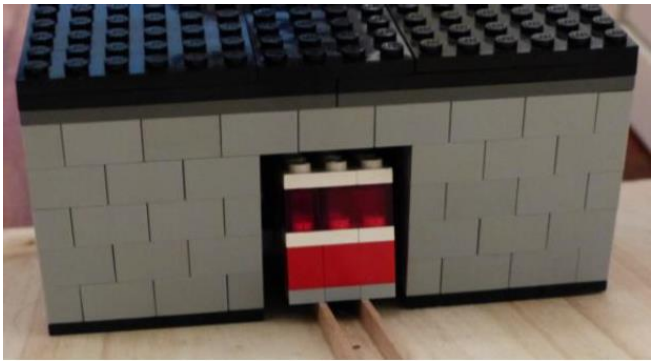


**Ilustración 48: Estación inferior anterior**

**Ilustración 49: Estación inferior modificada**

Además, al poner el sensor ultrasónico en la estación inferior, se ha acortado el recorrido, por lo que ha habido que adelantar el sensor de contacto.

El sensor ultrasonidos realiza tres funciones durante el funcionamiento del funicular:

- Localización de los vagones. El sensor ultrasonidos obviamente solo puede medir la distancia a la que se encuentra el vagón más próximo al sensor. La distancia a la que se encuentra el otro vagón puede ser calculada mediante la diferencia de la longitud total de la vía y la distancia a la que se encuentra el otro vagón, que se muestra en pantalla a lo largo de todo el recorrido.
- Detección de paradas accidentales. Puede darse el caso de que el funicular esté en marcha y por algún motivo se pare de manera accidental a pesar de estar el motor en marcha. Para la detección de este parón imprevisto el sensor ultrasónico es de gran utilidad. Si el motor está en marcha y la distancia no varía en un lapso de tiempo determinado, implica que el vagón se ha parado. En ese caso, se da la orden de parar el motor y aparece un mensaje en la pantalla del ladrillo que indica que ha habido una parada accidental. Para retomar la marcha del funicular se debe pulsar la tecla derecha. El mensaje mostrado en pantalla es el siguiente:

```
"PARADA ACCIDENTAL
 Para continuar…
  tecla derecha
  <distancia>"
```

- Detección de fallo en los frenos. Pese a que en la maqueta es imposible que pase, en la realidad puede darse el caso de que haya un fallo en los frenos y aunque el motor este apagado el funicular siga avanzando debido a su inercia y a la fuerza de la gravedad. En ese caso, se deberá activar automaticamente un freno auxiliar y deberá ser notificado el fallo para que el freno pueda ser revisado y así evitar males mayores en un futuro.

Con el fin de simular esta situación en la maqueta, el programa detecta que ha habido un fallo en los frenos cuando el motor esta apagado y la distancia medida por el sensor ultrasónico varia en el tiempo. En ese caso, aparece un mensaje en la pantalla del ladrillo que lo indica. Para retomar la marcha basta con pulsar el boton derecho. El mensaje mostrado en pantalla es el siguiente:

```
"FALLO FRENOS
FRENO AUXILIAR
   ACTIVADO
Para continuar…
tecla derecha
<distancia>"
```

Nótese que para las dos últimas funciones el programa debe estar constantemente comparando la distancia actual con la anterior. Por lo tanto, conviene plantearse el periodo en el que se comparan las distancias. Debido a la poca precisión del sensor ultrasónico (±3cm) y a la velocidad del motor, si se escoge un periodo excesivamente pequeño, como por ejemplo 5 ms, la parada accidental saltará pese a que el funcionamiento sea correcto, ya que en ese periodo es posible que al funicular todavía no le haya dado tiempo a avanzar lo suficiente como para que el movimiento sea perciba por el sensor ultrasónico . Por otro lado, un fallo de freno podrá dar un desenlace fatal, por lo que es importante que el periodo tampoco sea muy grande, para asegurar el mayor control posible. Por tanto, ha habido que determinar el periodo mínimo para que al sensor ultrasónico le de tiempo a detectar movimiento, de tal modo que el control de los frenos sea lo más riguroso posible y a su vez no salte la parada accidental de manera erronea. Finalmente, mediante ensayos se opta por un periodo de 150 ms.

Por otro lado, en cuanto a la programación, para la comparacion de la medida actual con la anterior lo más sencillo sería utilizar un bucle (`While(1)`) que guarde la distancia anterior y la compare con la actual. Sin embargo, en este programa ya tenemos un bucle iniciado, el del controlador PI que regula la velocidad. Debido a que la lectura del programa es de arriba abajo, no se pueden poner dos bucles paralelos, por lo que para su unificación en un solo bucle ambos deberían tener un mismo periodo. En el caso del controlador, como ya se ha comentado anteriormente, es estrictamente necesario que el periodo sea de aproximandamente 5 ms. Sin embargo, 5 ms no son suficientes para que el movimiento del vagón sea notorio, de tal modo que la parada accidental salta constantemente impidiendo el funcionamiento del funicular. Por todo ello, se decide crear un array dentro del bucle, donde poder almacenar las distancias medidas. Puesto que el array esta implementado en el bucle, cada 5 ms se escribe la medida tomada por el sensor ultrasónico en dicho array. Para la comparación de las medidas con un periodo de 150 ms se compara la distancia almacenada en la posición "i" con la de la posición "i-30".

**Tabla 12: Array**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | i-30 |
| i-29 | i-28 | i-27 | i-26 | i-25 | i-24 | i-23 | i-22 | i-21 | i-20 |
| i-19 | i-18 | i-17 | i-16 | i-15 | i-14 | i-13 | i-12 | i-11 | i-10 |
| i-9 | i-8 | i-7 | i-6 | i-5 | i-4 | i-3 | i-2 | i-1 | I |

El array inicializado es de una longitud de 4000 y es reinicializado cada vez que el vagón llega a la estación (cuando el sensor de contacto es presionado) y cuando pasa por la bifurcación (cuando el sensor de luz detecta el vagón). Por lo tanto, se ha comprobado mediante un ensayo que este valor de longitud del array es suficiente para almacenar todas las medidas tomadas. Es decir, el trayecto desde la bifurcación hasta el sensor de contacto dura menos de 20 s.

## 6.6. Programación

Todas las funciones descritas anteriormente deben estar implementadas en un mismo programa, junto con el controlador y el funcionamiento básico del funicular. Para la programación del programa se ha utilizado la Guía de Programación NXC de John Hansen[4], así como el Tutorial de NXC para programar Robots Lego Mindstorms NXT de Daniele Benedettelli[3]. El código completo del programa se encuentra en el anexo 2. El programa consta de las siguientes funciones:

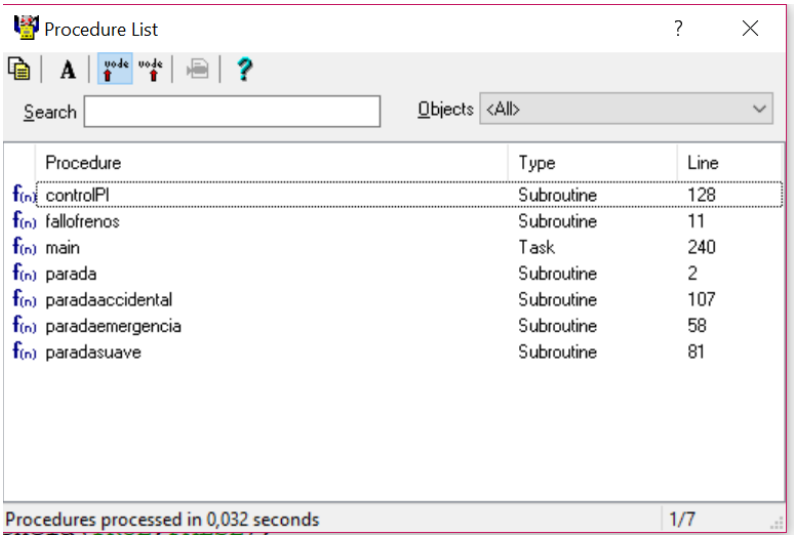



**Ilustración 50: Subrutinas del programa**

En los siguientes apartados se explican con más detalle las funciones.

### 6.6.1. Función principal

La función principal es la encargada de establecer los sensores, así como las variables principales como puede ser el umbral de luz para el que se activa el movimiento de las agujas (umbral=50) o la potencia de referencia a la que se debe alimentar el sistema (potencia=50%). En la tabla 13 se indican los sensores establecidos.

**Tabla 13: Relación de los sensores y los puertos de entrada**

| Sensor | Puerto |
|---|---|
| **de contacto** | 1 |
| **de luz** | 2 |
| **Ultrasónico** | 3 |

Además, al inicio del programa se emitirá un sonido que indique el comienzo del programa.

Tras ello se inicia un bucle en el que se ejecutan los movimientos principales del programa. El movimiento principal del programa consiste en el avance del vagón hasta que o bien llegue a la bifurcación o al sensor de contacto. Cuando se activa el sensor de contacto, se llama a la subrutina de parada(). Cuando los vagones llegan a la bifurcación el sensor de luz los detecta y activa los motores que mueven las agujas durante 700 ms. Tras ello, los motores de las agujas se paran mientras los vagones siguen su curso hasta volver a llegar a las estaciones donde se vuelve a activar el sensor de contacto. Este movimiento principal se repite constantemente a no ser que sea interrumpido por alguna subrutina, que serán descritas en los siguientes apartados. En la ilustración 51 se muestra un diagrama de flujo donde se observan las tareas ejecutadas dentro del bucle de esta función principal:

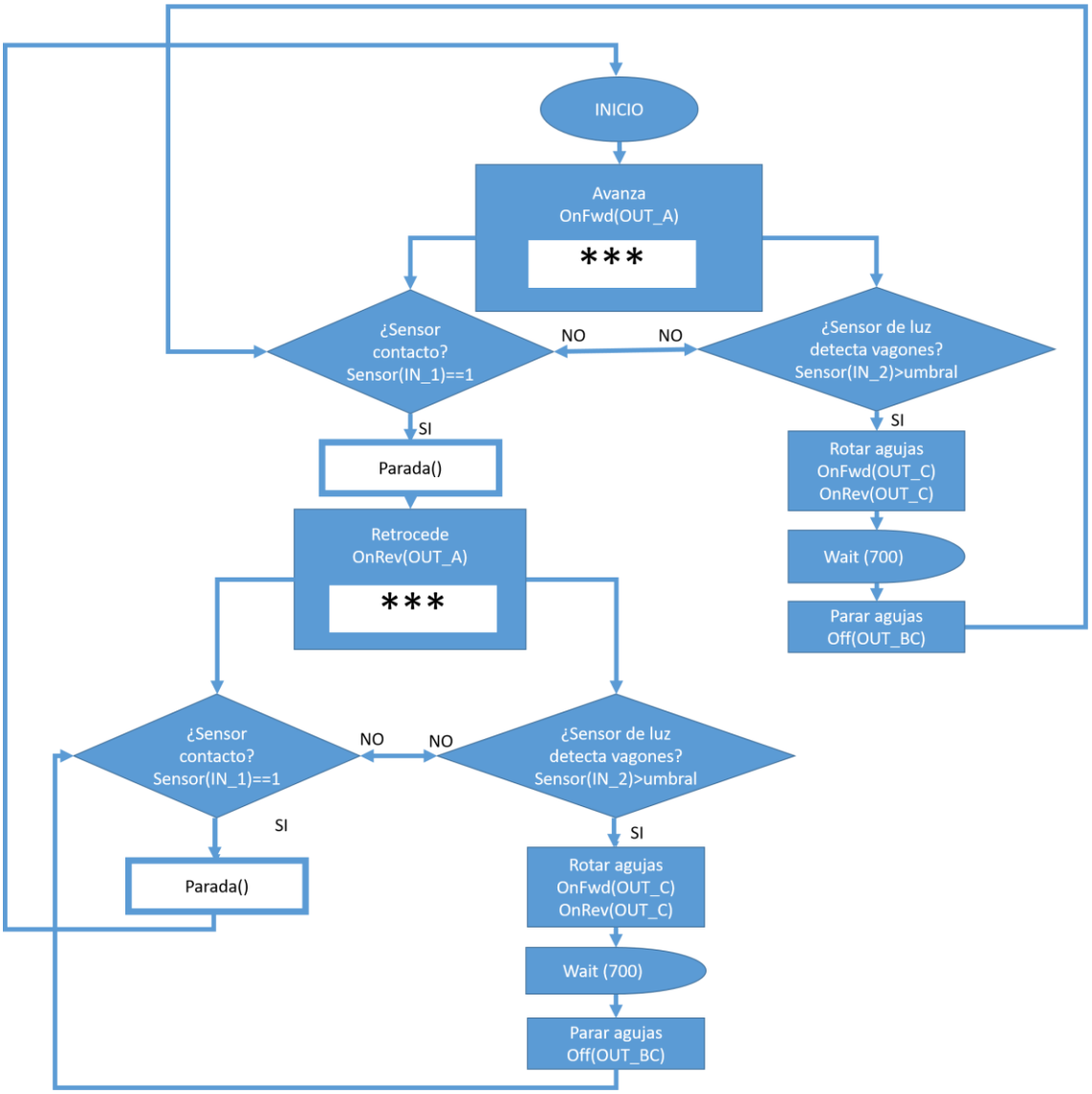



**Ilustración 51: Diagrama de flujo de la función principal**

En el diagrama de flujos de la ilustración 51, la casilla de los tres asteriscos significa que en ese mismo proceso se debe llamar a las funciones que se indican en la ilustración 52:

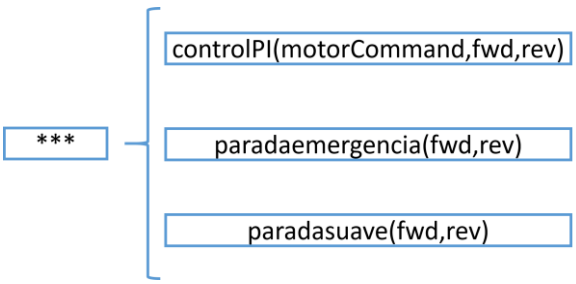



**Ilustración 52: Detalle del diagrama de flujo de la función principal**

### 6.6.2. Paradas

Durante el programa se pueden distinguir cuatro paradas diferentes:

- Parada: Cuando el sensor de contacto es activado, el funicular se detiene durante 1300 ms simulando de ese modo la parada en la estación del funicular.
- Parada de emergencia: Se activa al pulsar el botón central (botón naranja). Con ello el motor del sistema se apaga.
- Parada suave: Se activa al pulsar la tecla izquierda. Con ello el motor se queda en punto muerto, suponiendo así un freno suave del sistema.
- Parada accidental: Cuando el motor está en marcha pero el funicular no se mueve, el programa lo detecta y apaga el motor.

En cada parada aparecerá un mensaje en el ladrillo que indica cuál de las paradas ha ocurrido, como ya se ha mencionado anteriormente. Para volver a arrancar el funicular tras cualquiera de las tres últimas paradas bastará con pulsar la tecla derecha, como bien se indica en pantalla.

Cada parada tiene una programación diferente, no obstante, las cuatro se asimilan a la descrita en el diagrama de la ilustración 53:

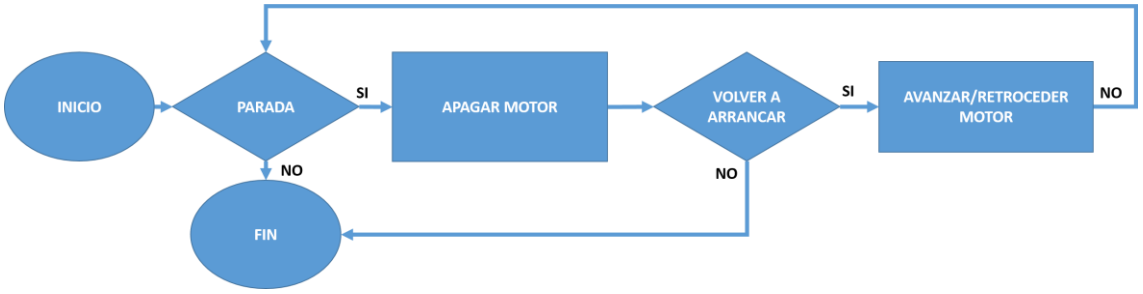



**Ilustración 53: Diagrama de flujo de las funciones de parada**

### 6.6.3. Fallo de frenos

La función de frenos es llamada tanto en la función de la parada de emergencia como en la parada suave. Esta detecta un fallo en los frenos cuando a pesar de estar el motor apagado el vagón está moviéndose. En caso, de darse un fallo en los frenos aparece un mensaje en pantalla que notifica que debido a un fallo en los frenos se ha activado el freno auxiliar. El diagrama de flujo de la función es el mostrado en la ilustración 54:

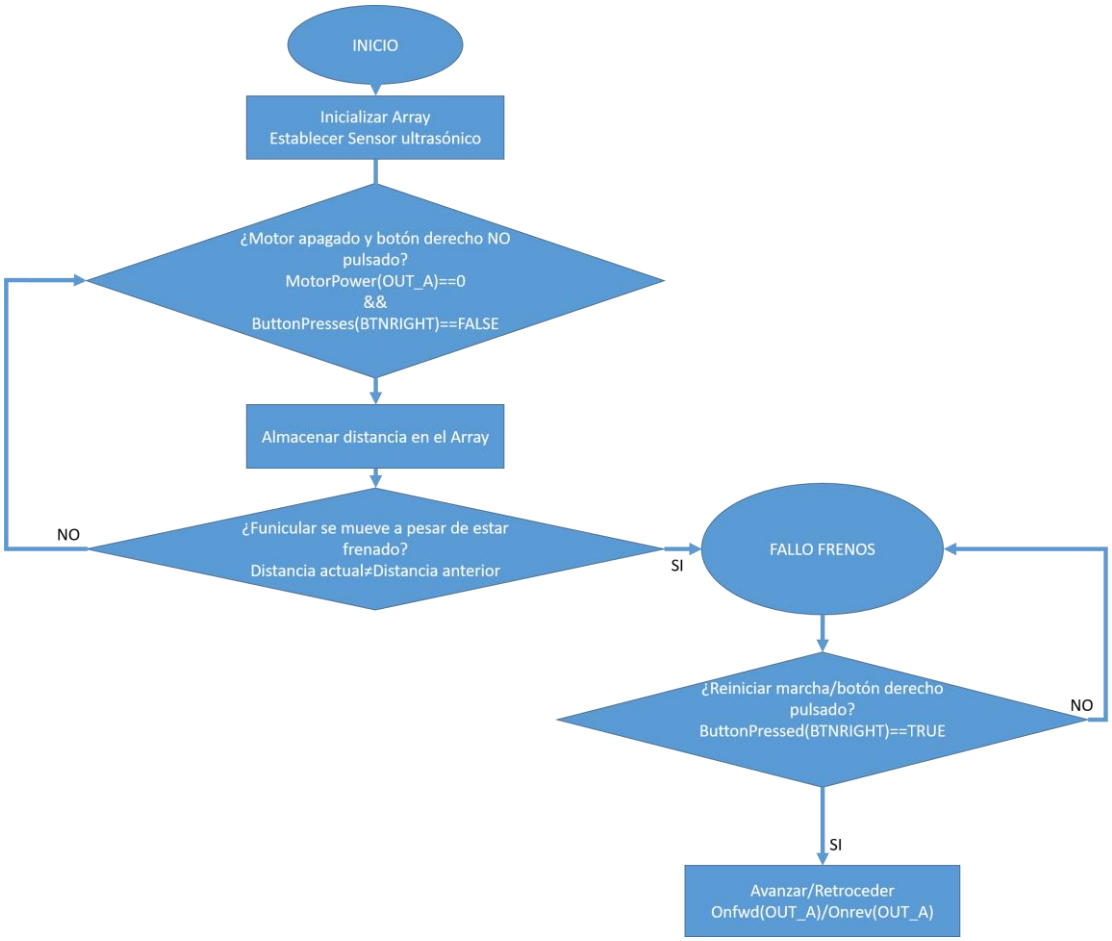



**Ilustración 54: Diagrama de flujo de la función de fallo en los frenos**

### 6.6.4. Control PI

Con el objetivo de regular la velocidad del motor se implementa un controlador PI. El controlador PI es programado en una subrutina, que en función de la velocidad de salida y el error respecto a la de referencia, indica la potencia que se le debe alimentar al motor para que la velocidad sea la de referencia. Además, en esta subrutina se mide la distancia con el sensor ultrasónico pudiendo así detectar la parada accidental mencionada anteriormente.

Esta subrutina podrá ser interrumpida por el sensor de contacto, el sensor de luz, la parada de emergencia o la suave.

En la ilustración 55 se muestra el diagrama de flujo del programa:

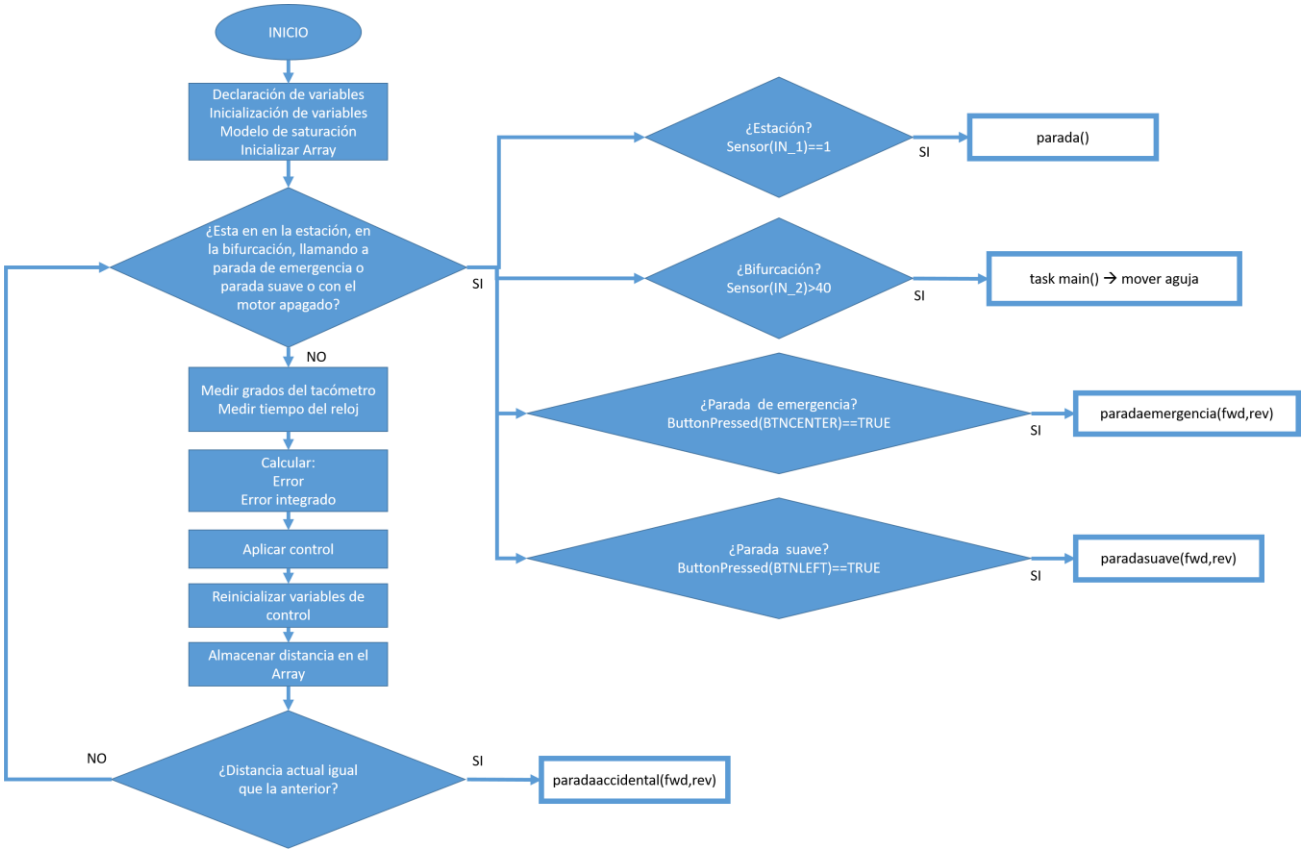



**Ilustración 55: Diagrama de flujo de la función control PI**

## 6.7. Pruebas

Para determinar el rango de operación del funicular se realizan pruebas. Si bien ya se ha comentado que el funcionamiento óptimo del sistema sería a una velocidad de 50, el funicular también podría funcionar a otras velocidades.

Para la determinación ese rango de operación el funicular se ha puesto a prueba variando la potencia de alimentación. Las observaciones dadas en la prueba se muestran en la tabla 14:

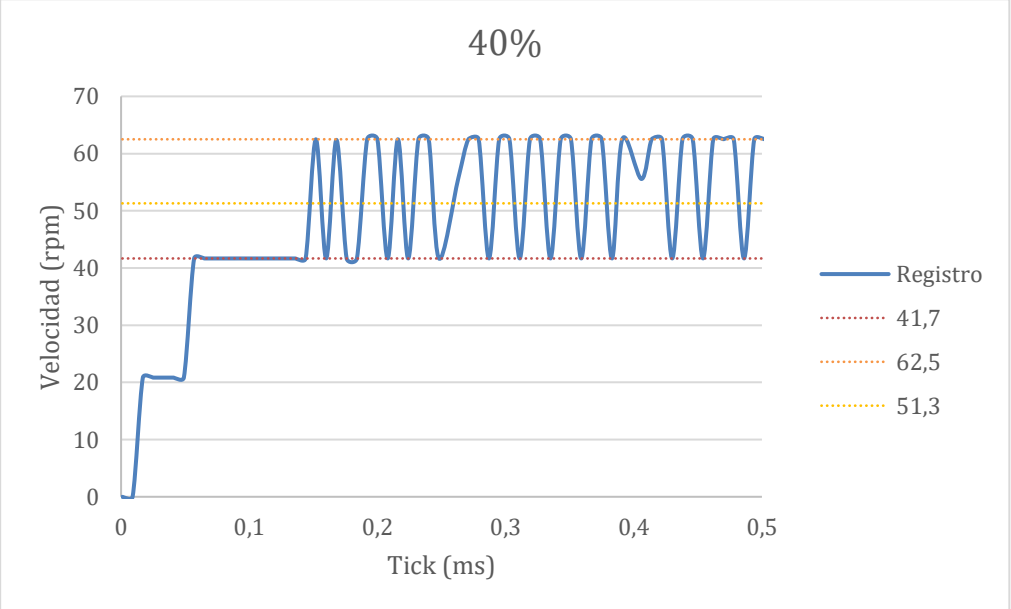
**Tabla 14: Determinación del rango de operación del sistema**

| P | OBSERVACIÓN |
|---|---|
| **10** | Excesivamente lento |
| **20** | El trayecto es lento sin oscilación. Al llegar al sensor de contacto en ocasiones no hace buen contacto |
| **30** | El trayecto es lento sin oscilación. Al llegar al sensor de contacto en ocasiones no hace buen contacto |
| **40** | Funcionamiento correcto |
| **50** | Funcionamiento correcto |
| **60** | Funcionamiento correcto |
| **70** | Funcionamiento con riesgo de descarrilamiento |
| **80** | Funcionamiento con riesgo de descarrilamiento |
| **90** | Funcionamiento con riesgo de descarrilamiento |
| **100** | Riesgo de descarrilamiento. Ruido excesivamente molesto |

De esto modo, se determina que el rango de funcionamiento óptimo del sistema sería una potencia de 40 a 60%.

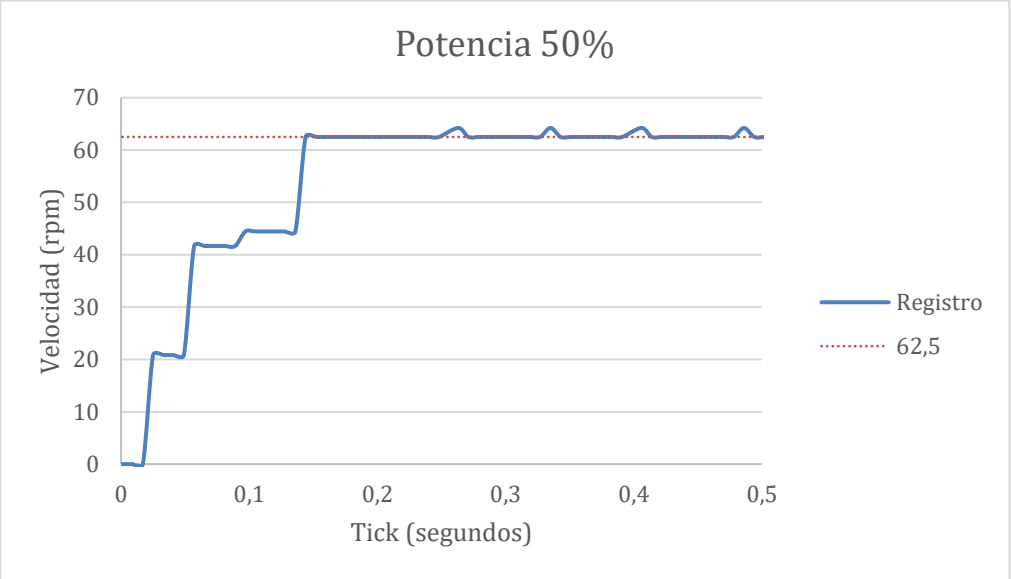
A la hora de diseñar e implementar el controlador se ha considerado que el sistema era lineal, de modo que el controlador debería de regular la velocidad adecuadamente independientemente de cual sea la entrada escalón. Para su comprobación se realizan algunas pruebas a distintas potencias.

En primer lugar se realiza la prueba a una potencia del 40% donde la entrada escalón al sistema es una velocidad de 51,3 rpm. Podemos observar como la velocidad va variando entre 41,7 y 62,5 rpm no llegando a estabilizarse en ningún momento.



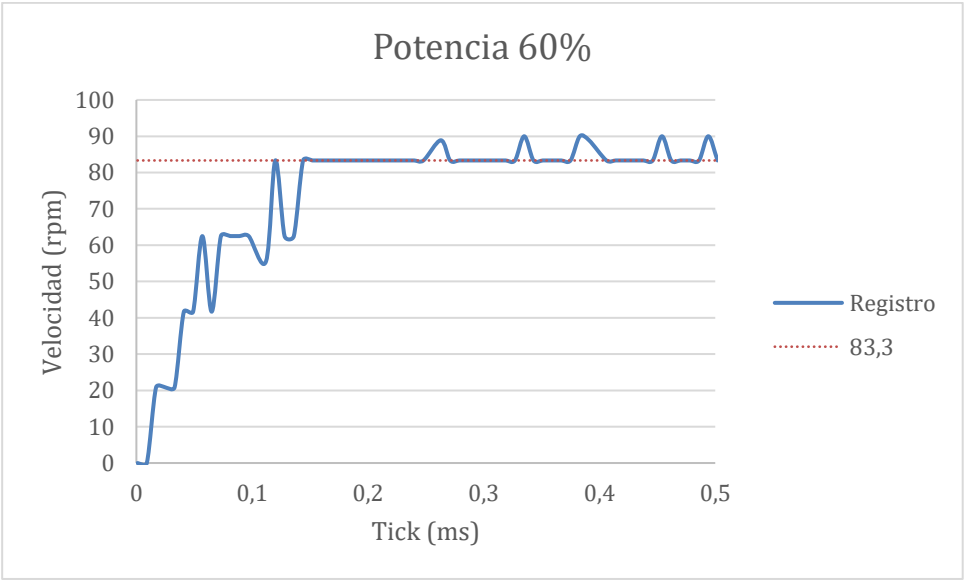

**Gráfica 12: Prueba controlador 40%**

En segundo lugar realizamos la misma prueba a un potencia del 50% donde la entrada escalón al sistema es 62,5 rpm. El controlador ha sido diseñado para este caso, por lo que el controlador es efectivo a pesar de haber ruido debido a los engranajes internos.



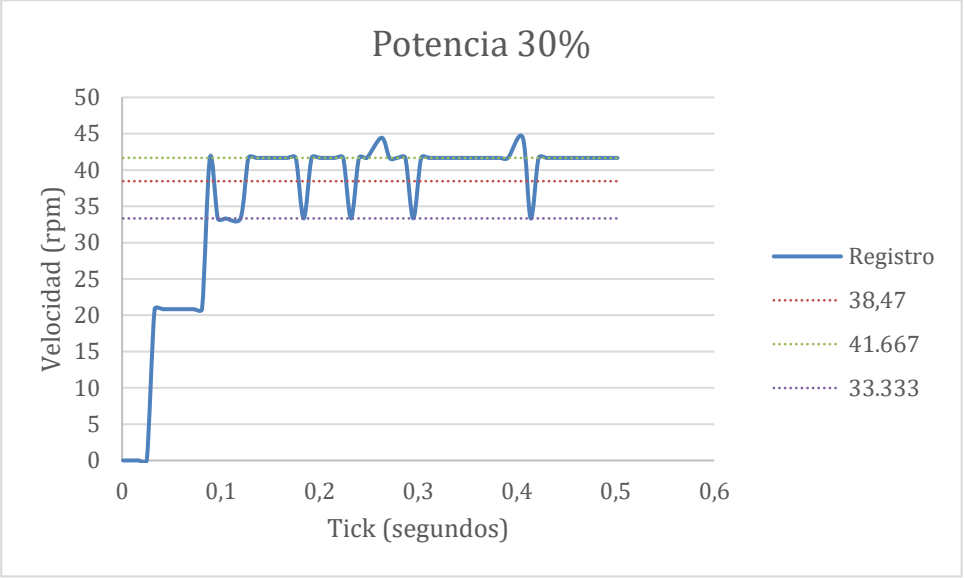

**Gráfica 13: Prueba controlador 50%**

Por último, realizamos la misma prueba a una potencia de alimentación del 60% donde la entrada escalón es una velocidad de 83,3 rpm. Como se puede observar en la gráfica 10, la respuesta del sistema es similar a la anterior.

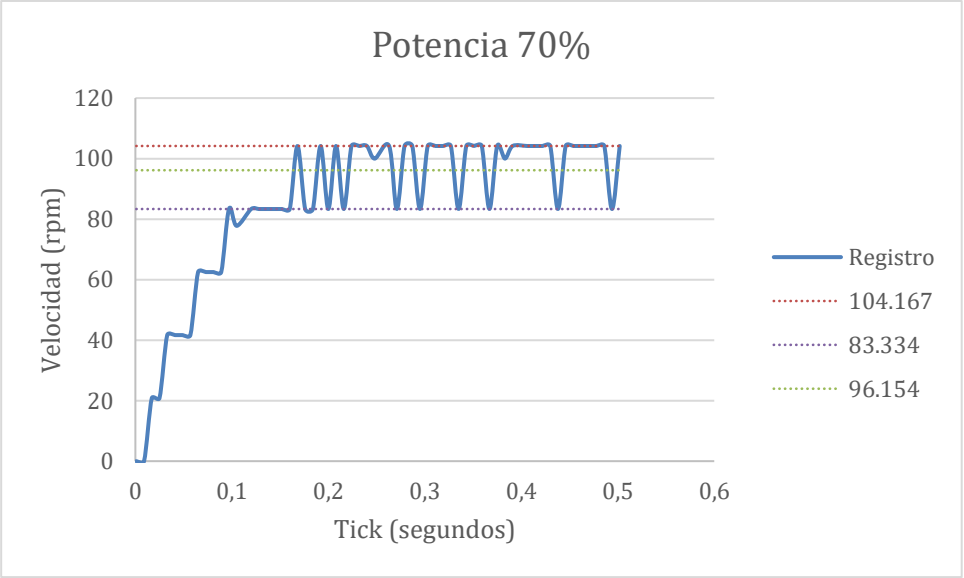



**Gráfica 14: Prueba controlador 60%**

Por lo tanto, el controlador es efectivo tanto a una potencia de 50% como de 60%, pero a una potencia de 40% no lo es. Ello puede ser debido a la fricción de los motores a distintas velocidades, a la fricción en el contacto rueda-carril y a la construcción interna, un producto de bajo coste realizado con piezas plásticas sometidas a mucho desgaste. También podría deberse a la consideración de que el sistema es lineal. Para ello realizamos las mismas pruebas a una potencia de 30% y 70%.




**Gráfica 15: Prueba controlador 30%**




**Gráfica 16: Prueba controlador 70%**

El controlador tampoco es efectivo a 30% ni a 70%, por lo que queda descarta la posibilidad de que a bajas potencias el motor no tenga suficiente impulso para vencer la fricción. Se puede concluir que el sistema no se puede considerar lineal. Por lo tanto, para asegurar que la velocidad sea regulada el rango de funcionamiento debe limitarse a 50-60%.

# 7. Planificación de tareas

La planificación de tareas se lleva a cabo en función de los plazos establecidos por la Escuela de Ingeniería de Bilbao, puesto que la documentación debe ser entregada antes del 22 de julio. El trabajo se inicia en febrero, una vez terminado los exámenes del primer cuatrimestre. El proyecto se compagina con las clases lectivas por lo que el número de horas que se trabaja cada día depende de la carga de trabajo de cada periodo. Se considera que aproximadamente se trabajan 2 horas al día, de lunes a sábado. Durante el periodo de exámenes del segundo cuatrimestre, del 14 de mayo al 2 de junio, se hace un parón reanudándose la tarea al finalizar dicho periodo. Tras ello, puesto que las clases cesan, el horario de trabajo se amplía a 5 horas al día. El trabajo queda finalizado el 15 de julio.

El trabajo se lleva a cabo en diferentes fases. Las tareas realizadas son las siguientes:

1) Búsqueda de información

La búsqueda de información se lleva a cabo durante 15 días y consiste en la recopilación de información sobre el kit Lego Mindstorms y las normas de programación del lenguaje NXC.

2) Análisis de alternativas

El análisis de alternativas se lleva a cabo durante 15 días y consiste en definir los objetivos y alcance del proyecto y la determinación de posibles soluciones a los problemas planteados.

3) Desarrollo del proyecto

Esta tarea ha sido descrita a lo largo del proyecto, por lo que en ese apartado tan solo se indicará su duración. La duración total de esta tarea es de 42 días y a su vez, en ella se distinguen distintas fases:

a) Mejora del diseño (17 días)
   i) Lijar (4 días)
   ii) Contratiempos aguja (3 días)
   iii) Pruebas de ensayo-error (10 días)
b) Alargar cable RJ12 (2dias)
c) Registro de velocidades (5 días)
d) Diseño del controlador (5 días)
e) Programación (9 días)
   i) Movimiento principal (2 días)
   ii) Funciones de seguridad (4 días)
   iii) Implementación del controlador (3 días)

f) Pruebas experimentales (3 días)

g) Últimos detalles (1 día)

4) Redacción del documento

La documentación del trabajo tiene una duración de 15 días.

Además, a lo largo del proyecto se dan los siguientes hitos:

- Reuniones con el tutor
- Entrega del trabajo

Puede concluirse que la ejecución total del trabajo lleva aproximadamente 250 horas. En la ilustración 57 se muestra un diagrama de Gantt donde se muestra la planificación del trabajo gráficamente.

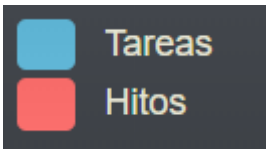
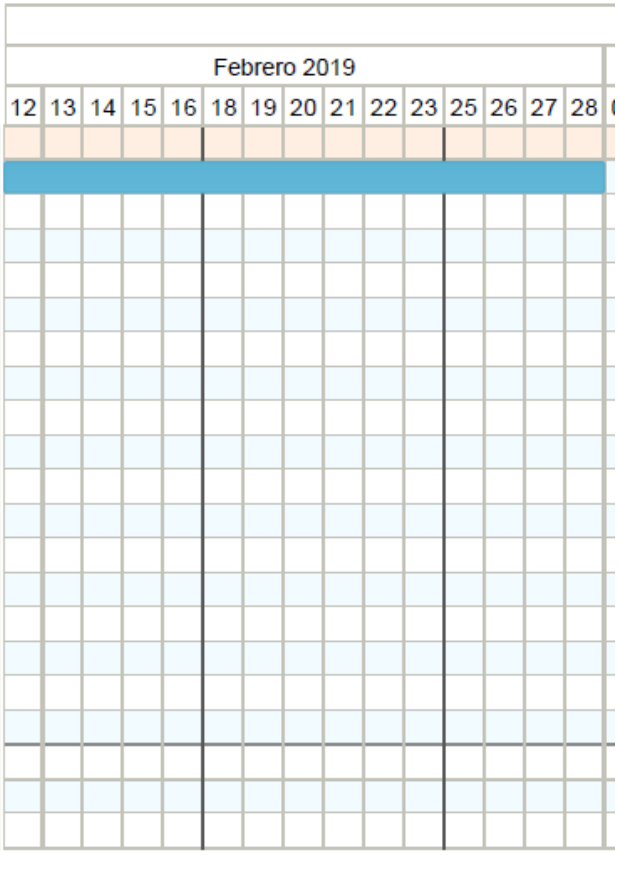
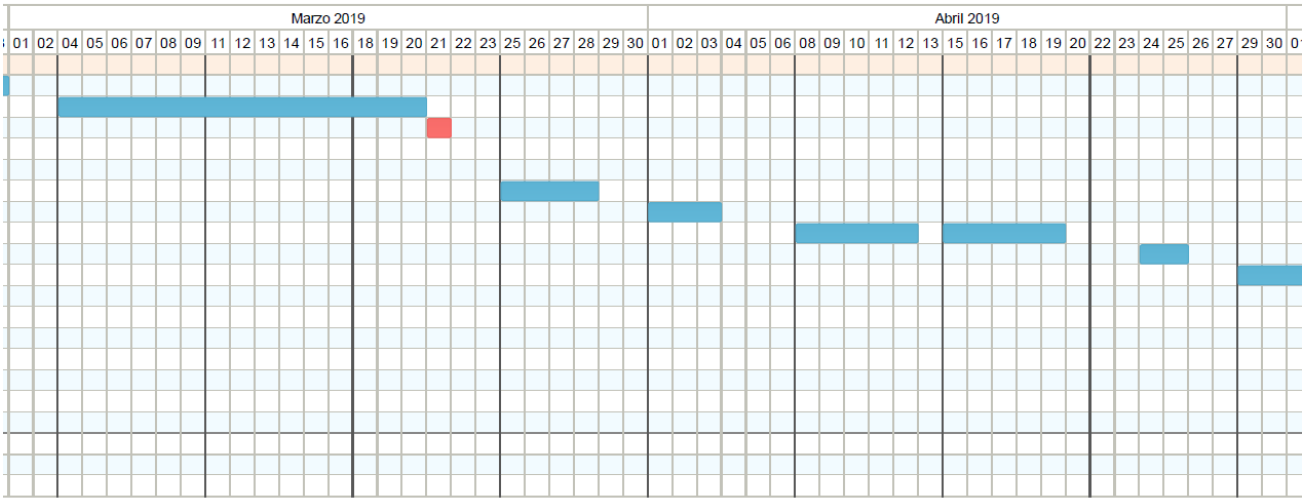



**Ilustración 56: Leyenda diagrama de Gantt**

| Activity | Días | |
|---|---|---|
| **TFG** | | **90.0** |
| Busqueda de información | 15.0 | |
| Análisis de alternativas | 15.0 | |
| Reunión con el tutor | 1.0 | |
| **Desarrolo del proyecto** | | 42.0 |
| **Mejora del diseño** | | 17.0 |
| Lijar | 4.0 | |
| Contratiempos aguja | 3.0 | |
| Pruebas de ensayo-error | 10.0 | |
| Alargar cable RJ12 | 2.0 | |
| Registro de velocidades | 5.0 | |
| Diseño controlador | 5.0 | |
| **Programación** | | 9.0 |
| Movimiento principal | 2.0 | |
| Funciones de seguridad | 4.0 | |
| Implementación del controlador | 3.0 | |
| Pruebas experimentales | 3.0 | |
| Últimos detalles | 1.0 | |
| Redacción del documento | 15.0 | |
| Reunión con el tutor | 1.0 | |
| Entraga del documento | 1.0 | |
| | | 90.0 |



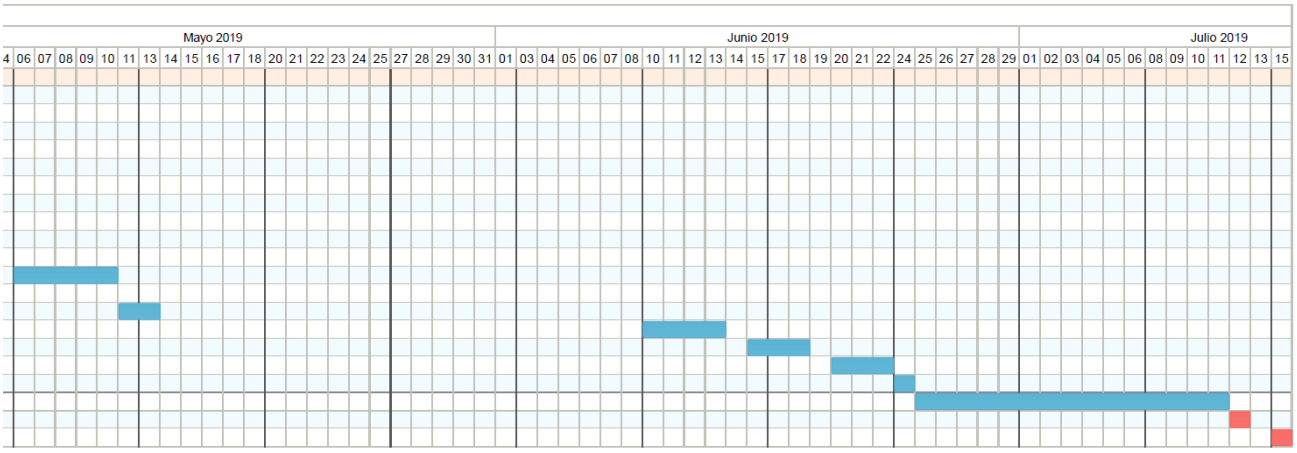



**Ilustración 57: Diagrama de Gantt**

# 8. Descargo de gastos

En este apartado se muestra un resumen de los gastos del proyecto tras finalizarlo. En el encontramos 5 partidas:

- Horas internas
- Amortizaciones
- Gastos
- Costes indirectos
- Imprevistos

En la tabla 15 se recogen los datos de cada partida, desglosada tanto en unidades necesitadas como en el coste de cada una de ellas:

**Tabla 15: Descargo de gastos**

| Concepto | Unidades | Nº Unidades | Coste Unitario | Coste | TOTAL PARTIDA |
|---|---|---|---|---|---|
| **HORAS INTERNAS** | | | | | 5.000,00 € |
| Ingeniería | horas | 250 | 20,00 € | 5.000,00 € | |
| **AMRTIZACIONES** | | | | | 87,51 € |
| ***Equipos*** | | | | | 10,00 € |
| Ordenador | horas | 200 | 0,05 € | 10,00 € | |
| ***Softwares utilizados*** | | | | | 77,50 € |
| Licencia MATLAB Student R2019a (MATLAB and Simulink Student Suite) | horas | 75 | 0,10 € | 7,50 € | |
| Licencia anual Microsoft Office 365 Hogar y Empresas | horas | 150 | 0,40 € | 60,00 € | |
| Licencia Microsoft Project | horas | 20 | 0,50 € | 10,00 € | |
| ***Herramientas*** | | | | | 0,01 € |
| Crimpadora | horas | 1 | 0,01 € | 0,01 € | |
| **GASTOS** | | | | | 3.076,12 € |
| ***Proyecto anterior*** | | | | | 2.812,07 € |
| Montaje maqueta | | | | 2.812,07 € | |
| ***Hardware Lego Mindstorms*** | | | | | 242,75 € |
| Kit Lego Mindstorms(ladrillo + sensores + servomotores + piezas technic) | | | | 242,75 € | |
| ***Montaje*** | | | | | 21,30 € |
| Cable RJ12 | | 1 | 2,50 € | 2,50 € | |
| Conector RJ12 macho | | 2 | 1,00 € | 2,00 € | |
| Acoplador RJ12 | | 2 | 1,50 € | 3,00 € | |
| Cola blanca | | 1 | 2,80 € | 2,80 € | |
| Lija dura | | 1 | 3,00 € | 3,00 € | |
| Lija blanda | | 1 | 2,00 € | 2,00 € | |
| Cera | | 1 | 4,00 € | 4,00 € | |
| Láminas de cristal | | 4 | 0,50 € | 2,00 € | |
| **SUBTOTAL 1** | | | | | 8.163,63 € |
| Costes indirectos (3%) | | | | | 244,91 € |
| **SUBTOTAL 2** | | | | | 8.408,54 € |
| Imprevistos (10%) | | | | | 840,85 € |
| **TOTAL** | | | | | **9.249,39 €** |

# 9. Conclusiones

En este proyecto se ha implementado un controlador de regulación de velocidad en una maqueta funicular además de haber introducido otras mejoras sobre un trabajo de fin de grado realizado previamente. Tras la realización del proyecto se ha llegado a una serie de conclusiones importantes que se discuten a continuación:

En primer lugar, la suposición de que el sistema es lineal en todo el rango de operación de los motores ha resultado no ser cierta. Las pruebas llevadas a cabo conducen a la conclusión de que el sistema de regulación solo trabaja en régimen lineal en un rango entre el 50 y el 60% de la potencia.

El controlador diseñado funciona correctamente a pesar del ruido introducido por los engranajes internos que realizan la reducción de velocidad interna del motor, lo que se traduce en un ruido superpuesto a la salida en forma de rizado fácilmente identificable en las gráficas de respuesta. Si bien es cierto que debido al material plástico y a la calidad media-baja de los sensores y servomotores estos no proporcionan gran precisión, el set completo NXT - con el ladrillo inteligente, sensores, motores y piezas de LEGO - es muchísimo más barato que el equipamiento habitual de un laboratorio de control y mucho más versátil.

A lo largo del proyecto se ha tenido que hacer frente a diversos problemas como la rotura de las agujas de la bifurcación y se ha podido observar como por un pequeño contratiempo el proyecto puede verse muy afectado provocando retrasos en su ejecución. Además, la singularidad de las piezas de Lego, hace que muchas veces no son compatibles con productos universales que se puedan encontrar en tiendas especializadas de electrónica. En este proyecto, ha sido necesario contar con un cable RJ12 más extenso que el proporcionado por Lego, la creatividad y la innovación han sido competencias imprescindibles para resolver esta incidencia.

Un posible punto de mejora de este proyecto sería la optimización de código. El hecho de haber utilizado un lenguaje de programación nuevo, y unos conocimientos relativamente escasos acerca de él, hacen pensar que el código puede optimizarse y así ocupar menos memoria, a la vez que se conseguiría una más rápida ejecución.

# Bibliografía

[1] Mindstorms LEGO.com. Recuperado de https://www.lego.com/es-es/mindstorms

[2] Bricx Command Center. Recuperado de http://bricxcc.sourceforge.net/

[3] Benedettelli, D. (2007). *Programming LEGO NXT Robots using NXC*.

[4] Hansen, J. (2013). NXC: NXC Programmer's Guide. Recuperado de http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html

[5] de la Herrán Prado, A. (2018). *Construcción y Control de una maqueta funicular* (Trabajo Fin de Grado). Escuela de Ingeniería de Bilbao.

[6] LEGO® MINDSTORMS & LEGO Technic. Recuperado de http://www.philohome.com/mindstorms.htm

[7] Oh, P. (2011). *Lego Programming - NXC Motor Velocity and Data Acquisition* [PDF]. Hands-on-Lab.

[8] Sensors and Sensing. Recuperado de http://cs.brown.edu/people/tdean/courses/cs148/02/sensors.html

[9] Marcos, M., Iriondo, N., Cabanes, I., & Zubizarreta, A. (2017). *Apuntes de Clase: Automática y Control*. Escuela de Ingeniería de Bilbao.

[10] Escuela de Ingeniería de Bilbao. (2019). *Apuntes de Clase: Control por Computador*.

[11] Correa, N., Remiro, C., & Costela, L. (2015). Sensor lego a butia - Proyecto Butiá. Recuperado de https://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/Sensor_lego_a_butia

[12] MATLAB Documentation. (1994). Recuperado de https://www.mathworks.com/help/

[13] HiTechnic Products. (2001). Recuperado de https://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NSK1042

[14] Nieves Molina, G. (2008). *Estudio de las posibilidades didácticas en ingeniería de control del LEGO Mindstorms NXT*. Universidad Politécnica de Cartagena.

[15] López Robles, V. (2012). *Maqueta de Lego para el seguimiento de un objetivo móvil. Interfaz de usuario mediante GUI de Matlab*. Universidad Politécnica de Cartagena.

# Anexo 1: Esquema

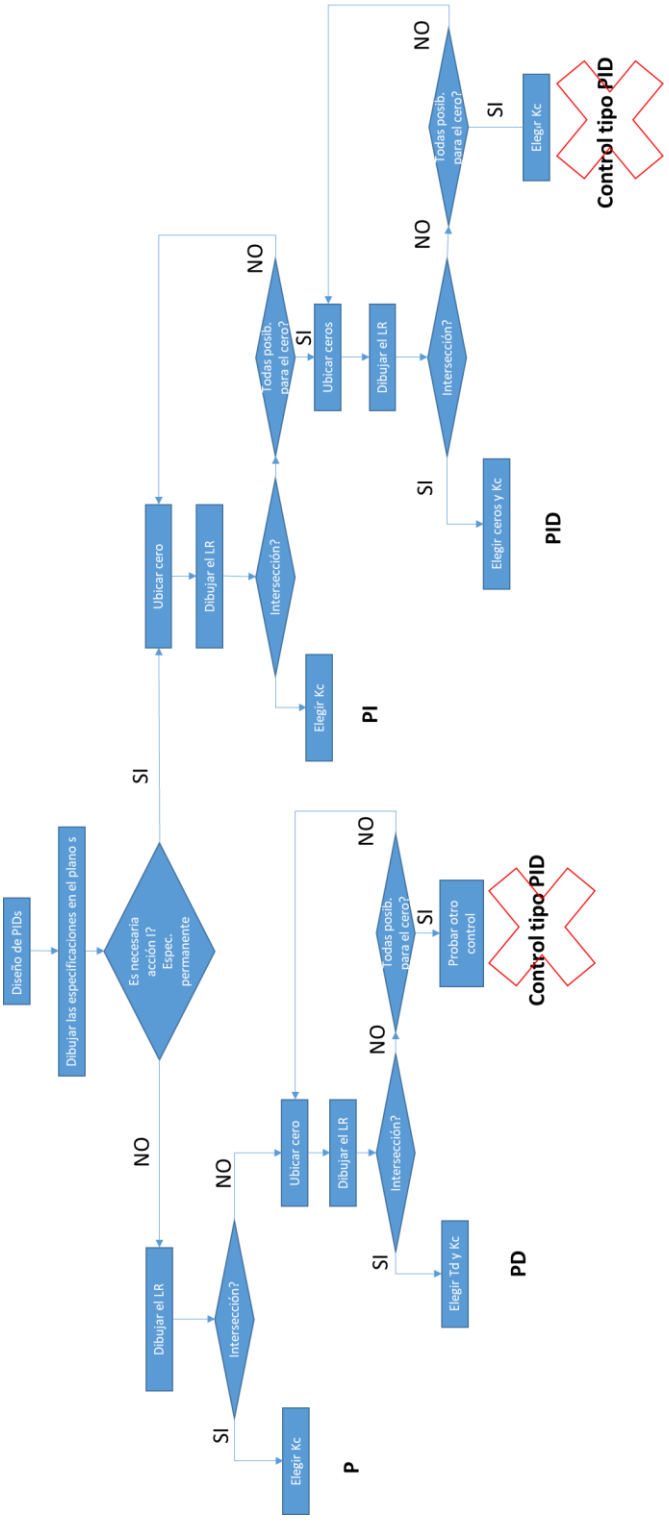



**Ilustración 58: Esquema de selección de controlador**

# Anexo 2: Códigos de programa

## Programa para el registro de las velocidades

El siguiente programa es utilizado para la recogida de datos de la velocidad del funicular. Este programa ha sido utilizado para la recogida de datos tanto una vez implementado el controlador como antes de implementarlo.

```
#define MOTOR OUT_A // definir el motor en el puerto A
#define DEG2RPM 166.667 // deg/msec to RPM

// Global variables
unsigned int result;
byte filehandle; // manejar el archivo de datos
short fileSize; // tamaño del archivo
short bytesWritten; // número de bytes escritos en el archivo
string fileHeader; // encabezado de columna para los datos en el
archivo
int fileNumber, filePart; // enteros para dividir los nombres de los
archivos de datos
string fileName; // nombre de archivo
string strFileNumber; // número de archivo, ej: myDataFile 1, 2, 3
string strFilePart; // parte del archivo, ej: myDataFile1-1, 1-2, 1-3
string text; // cadena a escribir en el archivo, es decir,
valores de datos

// Crear e inicializar un archivo
void InitWriteToFile() {
fileNumber = 0; // establece el primer archivo de datos en cero
filePart = 0; // establecer la primera parte del primer archivo de
datos a cero
fileName = "myMotorSpeed.csv"; // nombre archivo
result=CreateFile(fileName, 1024, filehandle);


 // Comprobar si existe el archivo
while (result==LDR_FILEEXISTS) // LDR_FILEEXISTS se marca si el
archivo ya existe
{
CloseFile(filehandle);
fileNumber = fileNumber + 1; // Si el archivo de datos ya existe, cree
uno nuevo.
fileName=NumToStr(fileNumber);
fileName=StrCat("myMotorSpeed", fileName, ".csv");
result=CreateFile(fileName, 1024, filehandle);
} // end while
// Suena un tono cada vez que un nuevo archivo es creado
PlayTone(TONE_B7, 5);
```

```
 fileHeader = "Tick, Motor Speed"; // titulo del archivo de datos
 WriteLnString(filehandle, fileHeader, bytesWritten);
} // end InitWriteToFile


void WriteToFile(string strTempText) {
 // almacena el texto (es decir, ticks y motorRpm que se
escribirán en el archivo
 // escribir cadena en el archivo
 result=WriteLnString(filehandle, strTempText, bytesWritten);
 // si se llega al final del archivo, cierre el archivo y cree una
nueva parte
 if (result==LDR_EOFEXPECTED) // LDR_EOFEXPECTED se marca cuando
finliza el archivo
 {
 // cerrar archivo actual
 CloseFile(filehandle);

 // crear el proximo nombre de archivo
 filePart = filePart + 1; strFileNumber =
 NumToStr(fileNumber); strFilePart =
 NumToStr(filePart);
 fileName = StrCat("myMotorSpeed", strFileNumber,"-", strFilePart
,".csv");
 // borrar archivo si existiese
 DeleteFile(fileName);

 // crear nuevo archivo
 CreateFile(fileName, 1024, filehandle);
 // Suena un tono cada vez que un nuevo archivo es creado
 PlayTone(TONE_B7, 5);
 WriteLnString(filehandle, strTempText, bytesWritten);
 } // end if
} // end WriteToFile


// Cerrar el archivo
void StopWriteToFile() {
 CloseFile(filehandle);
} // end StopWriteToFile

task main() {

 long prevAngleInDegrees; // marcador de posición para el grado
leído por el encoder del motor
 long curAngleInDegrees; // Ángulo actual del motor [DEG]
 long deltaAngleInDegrees; // Variación del ángulo del motor[DEG]
 long prevTick;
 long curTick; // Valor actual de tiempo
 long deltaT; // Para calcular el tiempo transcurrido entre ticks
```

```
 float elapsedTimeInSeconds; // tiempo en segundos
 string strElapsedTimeInSeconds; // representación en cadena del
tiempo transcurrido

 float motorRpm; // velocidad del motor [RPM]
 string strMotorRpm; // almacenar el valor entero de motorRpm como
cadena
 string strDeltaT; // almacenar el valor de deltaT como cadena
 string strDeltaAngleInDegrees; // almacenar el valor de
deltaAngleInDegrees como una cadena

 //variables del controlador
 float kp,ki; float
 desiredRpm; float
 error;
 float  prevError; float
 deltaError; float
 integralOfError;
 float motorCommand; // velocidad de referencia


 //Crea un nuevo archivo que captura el tiempo y la velocidad del
motor
 InitWriteToFile();
 prevAngleInDegrees = 0; //motor inicialmente inmóvil por lo que el
ángulo establecido a cero
 elapsedTimeInSeconds = 0.0; //establecer el tiempo transcurrido a
cero

 deltaError=0;
 prevError=0;
 integralOfError=0;
 deltaError=0;


 kp=1.3497;
 ki=12.335;
 desiredRpm=62.5;
 motorCommand=50;

  //saturación del actuador
 if(motorCommand>=100)
 {motorCommand=100;}
 if (motorCommand<=0)
 {motorCommand=0;}  prevTick

 = CurrentTick();

 OnFwd(MOTOR,motorCommand);

//iniciar bucle
```

```
 while (true<>ButtonPressed(BTNCENTER,false))

 curAngleInDegrees = MotorRotationCount(MOTOR); // obtener
posición relativa
 deltaAngleInDegrees = curAngleInDegrees - prevAngleInDegrees;
 strDeltaAngleInDegrees = FormatNum("deltaAngle = %ld",
deltaAngleInDegrees);
 curTick = CurrentTick(); // leer la hora
 deltaT = curTick - prevTick; //medir el tiempo transcurrido
entre lecturas de ángulo
 strDeltaT = FormatNum("deltaT = %ld", deltaT);
 elapsedTimeInSeconds = elapsedTimeInSeconds + (deltaT/1000.0);
// en segundos

 motorRpm = deltaAngleInDegrees * DEG2RPM / deltaT;
 strMotorRpm = FormatNum("%5.3f", motorRpm);
 //Mostrar en NXT Brick
 TextOut(0, LCD_LINE2, strDeltaT);
 TextOut(0,LCD_LINE4,strDeltaAngleInDegree);
 TextOut(0,LCD_LINE6, strMotorRpm);

 error=desiredRpm-motorRpm;
 deltaError=error-prevError;
 integralOfError=prevError+error;

 //señal de control

 motorCommand=kp*error+ki*integralOfError;

 //reinicializar variables
 prevTick = curTick;
 prevAngleInDegrees = curAngleInDegrees;
 prevError=error;

 Wait(5); // actualizar la pantalla cada 5 milisegundos

 //escribir el tiempo transcurrido en segundos y la velocidad del
motor en RPM al archivo
 strElapsedTimeInSeconds = FormatNum("%5.3f",
elapsedTimeInSeconds);
 text=StrCat(strElapsedTimeInSeconds, ",", strMotorRpm,"," );
 // escribe el texto en un archivo
 WriteToFile(text);
 } // end of while
 // close the file
 StopWriteToFile();
} // end of main
```

## Programa para el funcionamiento del funicular

El siguiente programa para el funcionamiento del funicular incluye todas las subrutinas ya mencionadas a lo largo del proyecto: parada al llegar a la estación, parada de emergencia, parada suave, detección de fallo en los frenos, parada accidental y control PI. Además, en la función principal se encuentra el código para el movimiento principal del funicular.

```
 //PARADA FUNCTION
inline void parada()
{
          Wait(700);
        TextOut(30,LCD_LINE3,"PARADA",TRUE);
        Off(OUT_A);
        Wait(1300);
        ClearScreen();
}
```

---

```
inline void fallofrenos(bool fwd,bool rev)
{
 int dataoff[];
 int j;
 j=0;
 ArrayInit(dataoff,0,2000);
 SetSensorUltrasonic(IN_3);

while (MotorPower(OUT_A)==0 && ButtonPressed(BTNRIGHT,
TRUE)==FALSE)
{
 dataoff[j]=SensorUS(IN_3);

 TextOut(0,LCD_LINE7,"dist=");
 NumOut(30,LCD_LINE7,dataoff[j]);

    if (j>60
    && dataoff[j]!=dataoff[j-30]
    && dataoff[j]!=255
    && dataoff[j-60]!=255)
        {
        ClearScreen();
        TextOut(16,LCD_LINE1,"FALLO FRENOS");
        TextOut(8,LCD_LINE2,"FRENO AUXILIAR");
        TextOut(32,LCD_LINE3,"ACTIVADO");
        TextOut(0,LCD_LINE5,"Para arrancar...");
        TextOut(8,LCD_LINE7,"tecla derecha");
        TextOut(0,LCD_LINE8,"dist=");
        NumOut(30,LCD_LINE8,SensorUS(IN_3));
        if (ButtonPressed(BTNRIGHT,TRUE)==TRUE)
        {
```

```
        do{
        Off(OUT_A);
        }
        until(ButtonPressed(BTNRIGHT,TRUE)==TRUE)
        {
       if (fwd==TRUE) {OnFwd(OUT_A,50);}
       if (rev==TRUE) {OnRev(OUT_A,50);}
       ClearScreen();
       }
    }
    }
j=j+1;
Wait(5);
}//end while
}//end task
```

---

```
inline void paradaemergencia(bool fwd,bool rev)
  {
     if (ButtonPressed(BTNCENTER,FALSE)==TRUE)
     {
              ClearScreen(); TextOut(0,LCD_LINE2,"PARADA
              EMERGENCIA");
              TextOut(0,LCD_LINE4,"Para continuar...");
              TextOut(0,LCD_LINE6,"tecla derecha");
              TextOut(0,LCD_LINE8,"dist=");
              NumOut(30,LCD_LINE8,SensorUS(IN_3));
              do{
              Off(OUT_A);
              fallofrenos(fwd,rev);
                }
              until(ButtonPressed(BTNRIGHT,TRUE)==TRUE)
                {
                if (fwd==TRUE) {OnFwd(OUT_A,50);}
                if (rev==TRUE) {OnRev(OUT_A,50);}
                ClearScreen();
                }
     }
  }
```

---

```
inline void paradasuave(bool fwd,bool rev)
  {
     if (ButtonPressed(BTNLEFT,FALSE)==TRUE)
     {
              ClearScreen();
              TextOut(0,LCD_LINE2,"PARADA");
              TextOut(0,LCD_LINE4,"Para continuar...");
              TextOut(0,LCD_LINE6,"tecla derecha");
              TextOut(0,LCD_LINE8,"dist=");
```

```
                NumOut(30,LCD_LINE8,SensorUS(IN_3));
                do{
                Coast(OUT_A);
                fallofrenos(fwd,rev);
                  }
                until(ButtonPressed(BTNRIGHT,TRUE)==TRUE)
                  {
                  if (fwd==TRUE) {OnFwd(OUT_A,50);}
                  if (rev==TRUE) {OnRev(OUT_A,50);}
                  ClearScreen();
                  }
    }
  }
```

---

```
inline void paradaaccidental(bool fwd,bool rev)
{
ClearScreen(); TextOut(0,LCD_LINE2,"PARADA
ACCIDENTAL");
TextOut(0,LCD_LINE4,"Para continuar...");
TextOut(0,LCD_LINE6,"tecla derecha");
TextOut(0,LCD_LINE8,"dist=");
NumOut(30,LCD_LINE8,SensorUS(IN_3));
do{
  Off(OUT_A);
  }
until(ButtonPressed(BTNRIGHT,TRUE)==TRUE)
  {
  if (fwd==TRUE) {OnFwd(OUT_A,50);}
  if (rev==TRUE) {OnRev(OUT_A,50);}
  ClearScreen();
  }
}
```

---

```
inline void controlPI(float motorCommand,bool fwd,bool rev)
 {
  //variables para el control PI
  long prevAngleInDegrees; // marcador de posición para el grado
 leído por el encoder del motor
  long curAngleInDegrees; // ángulo actual del motor [DEG]
  long deltaAngleInDegrees; // variación del ángulo del motor [DEG]

  long prevTick;
  long curTick; // valor actual del tiempo
  long deltaT; // variación del tiempo transcurrido

  float motorRpm; // velocidad del motor [RPM]
  string strMotorRpm; // almacenar el valor entero de motorRpm como
```

*cadena*

```
 float kp,ki; float
 desiredRpm; float
 error; float
 prevError; float
 deltaError;
 float integralOfError;

 //inicializar variables PI
 prevAngleInDegrees = 0; // motor inicialmente inmóvil por lo que el
ángulo establecido a cero
 prevError=0;
 integralOfError=0;
 deltaError=0;

 kp=1.3497;
 ki=12.335;
 desiredRpm=62.5;
 motorCommand=50;
  //saturación del actuador
 if(motorCommand>=100)
 {motorCommand=100;}
 if (motorCommand<=0)
 {motorCommand=0;}

 prevTick = CurrentTick();

 //Para almacenar las medidas del sensor ultrasónico
 int data[];
 int i;
 i=0;
 ArrayInit(data,0,2000);
 SetSensorUltrasonic(IN_3);


while (Sensor(IN_2)<40
        && Sensor(IN_1)!=1
        && ButtonPressed(BTNCENTER,FALSE)==FALSE
        && ButtonPressed(BTNLEFT,FALSE)==FALSE
        && MotorPower(OUT_A)!=0)
 {
 curAngleInDegrees = MotorRotationCount(OUT_A); // obtener la
posición relativa
 deltaAngleInDegrees = curAngleInDegrees - prevAngleInDegrees;

 curTick = CurrentTick(); // lee el valor actual del reloj
 deltaT = curTick - prevTick; //medir el tiempo transcurrido
entre lecturas de ángulo

 motorRpm = deltaAngleInDegrees * 166.667 / deltaT;
```

```
  strMotorRpm = FormatNum("%5.3f", motorRpm);

  // Mostrar en NXT Brick
  TextOut(0,LCD_LINE6, "vel=");
  TextOut(38, LCD_LINE6, strMotorRpm);

  error=desiredRpm-motorRpm;
  deltaError=error-prevError;
  integralOfError=prevError+error;

  //señal de control
  motorCommand=kp*error+ki*integralOfError;

  //Reinicializar variables de control
  prevTick = curTick;
  prevAngleInDegrees = curAngleInDegrees;
  prevError=error;

  data[i]=SensorUS(IN_3);

  TextOut(0,LCD_LINE7,"dist=");

  NumOut(38,LCD_LINE7,data[i]);


  if (i>30
       && data[i]==data[i-30]
       && data[i]!=255
       && data[i-30]!=255
       && data[i]>7)
  {paradaaccidental(fwd,rev);}

  i=i+1;

  Wait(5); // actualizar la pantalla cada 5 milisegundos
   }
 }
```

---

```
task main()
 {
 //Inicializar tipo y modo de sensor

           SetSensor(IN_1,SENSOR_TOUCH);
           SetSensorLight(IN_2);
           SetSensorUltrasonic(IN_3);

           int umbral1=50;
           float motorCommand=50;
```

```
Wait(50); //Tiempo para que los sensores se detecten
correctamente
PlayTone(440,1000); //Señal que indica comienzo de
programa


while(TRUE)
    {
    repeat(1)
    {
do
    {

    OnFwd(OUT_A,motorCommand);
    paradasuave(TRUE,FALSE);
    paradaemergencia(TRUE,FALSE);
    controlPI(motorCommand,TRUE,FALSE);

    if (Sensor(IN_2)>umbral1)
        {
         Wait(500);
         OnFwd(OUT_C,10);
         OnRev(OUT_B,20);
         Wait(700);
         Off(OUT_BC);
         Wait(1000);

         }
    }
until(Sensor(IN_1)==1);
OnRev(OUT_A,15);
parada();
do

    {

    OnRev(OUT_A,motorCommand);
    paradasuave(FALSE,TRUE);
    paradaemergencia(FALSE,TRUE);
    controlPI(motorCommand,FALSE,TRUE);


    if (Sensor(IN_2)>umbral1)
       {
       Wait(500);
       OnRev(OUT_C,10);
       OnFwd(OUT_B,20);
       Wait(700);
       Off(OUT_BC);
       Wait(1000);
       }
   }
```

```
            until(Sensor(IN_1)==1);

            OnFwd(OUT_A,motorCommand);
            paradasuave(TRUE,FALSE);
            paradaemergencia(TRUE,FALSE);
            controlPI(motorCommand,TRUE,FALSE);

            parada();
}
```

# Anexo 3: Manual de usuario

En este apartado se especifican las instrucciones para el funcionamiento de la maqueta.

1. / Antes de encender el NXT debe asegurarse de que los motores y los sensores están en los puertos correspondientes a cada uno. En la tabla 16 se detalla la correspondencia:

**Tabla 16: Correspondencia de motores y sensores con cada puerto de alimentación**

| Motores | | Sensores | |
|---|---|---|---|
| **Motor** | **Puerto** | **Sensor** | **Puerto** |
| **Estación superior** | A | de contacto | 1 |
| **Aguja superior** | B | de luz | 2 |
| **Aguja inferior** | C | ultrasónico | 3 |

Además, debe asegurarse que los hilos de las vías estén en paralelo, para así evitar que se enreden.

2. / Una vez conectados, se puede proceder a encender el NXT. Encienda el NXT pulsando el botón Intro color naranja, que se encontrará en el centro del ladrillo.

3. / Aparecerá el menú principal, tal como se muestra en la imagen del NXT. Utilice las teclas de dirección izquierda y derecha para explorar las distintas opciones NXT. Pulse el botón Intro naranja para seleccionar.

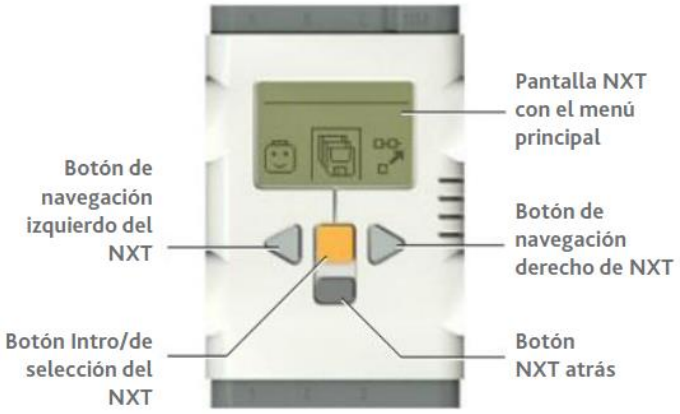



**Ilustración 59: Ladrillo NXT**

El ladrillo funciona de forma similar a un teléfono móvil. Se maneja a través de un menú y los archivos se guardan en carpetas.

4. / Para poner en funcionamiento el funicular debe acceder al programa que lo pone en marcha. Este se encuentra en la siguiente dirección: My files/Software files/funicular (2).

5. / Una vez en la carpeta del programa, para ejecutarlo pulse Run.

Tras seguir estos pasos el funicular comenzará su marcha.

Las funciones que podrá realizar con esta maqueta son las siguientes:

- Si desea realizar una parada de emergencia→ Pulse el botón central
- Si desea realizar una parada normal en mitad del recorrido→ Pulse la tecla izquierda

Para volver a la marcha tras las paradas se deberá pulsar la tecla derecha.

En caso de que estando el motor en marcha los vagones no avancen saltará una alarma en la pantalla que indique que ha habido una parada accidental y el motor se parará. Si desea volver a la marcha pulse la tecla derecha.

En caso de que estando el motor parado el funicular siga avanzando saltará una alarma en pantalla que indique que ha habido un fallo en los frenos por el que se han activado los frenos auxiliares. Si desea volver a la marcha pulse la tecla derecha.

Para dejar de ejecutar el programa pulse el botón inferior. Si desea apagar el NXT mantenga pulsado el botón inferior hasta que el ladrillo se apague.

 ADVERTENCIAS:

ADVERTENCIA 1: Para el funcionamiento del funicular el NXT debe tener batería. Para recargar las baterías del NXT se debe usar el cargador 10V DC LEGO® Education.

ADVERTENCIA 2: Debe evitarse la exposición de foco de luz dirigido al sensor de luz, debido a su foto sensibilidad.

ADVERTENCIA 3: Para un correcto funcionamiento la maqueta debe estar sobre una superficie lisa recta, de modo que esté equilibrada.