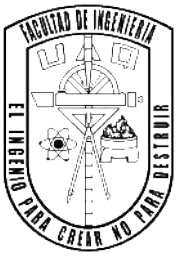

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Ingeniería en automatización

CONSTRUCCIÓN, INSTRUMENTACIÓN Y CONTROL DE UN ROBOT BALANCÍN

TESIS

Que como parte de los requisitos para obtener el grado de
Ingeniero en automatización con línea terminal en electrónica y sistemas embebidos

Presenta:

Juan Pablo Luna Moreno

Dirigido por:

Dr. Roberto Valentín Carrillo Serrano

SINODALES

Dr. Roberto Valentín Carrillo Serrano Presidente	Firma
Dr. Mariano Garduño Aparicio Secretario	Firma
Dr. Suresh Thenozhi Vocal	Firma
MC. Moisés Agustín Martínez Hernández Suplente	Firma
Dr. Víctor Manuel Hernández Guzmán Suplente	Firma

Dr. Manuel Toledano Ayala
Director de la Facultad

Centro Universitario
Querétaro, Qro.
Abril de 2023
México

CONSTRUCCIÓN, INSTRUMENTACIÓN Y CONTROL DE UN ROBOT BALANCÍN

por

Juan Pablo Luna Moreno

RESUMEN

En este trabajo se presenta la construcción, instrumentación y el control de un robot balancín seguidor de línea con un controlador maestro PD y lazos esclavos de velocidad PI, capaz de tomar pendientes de 15 grados con el código embebido en un microcontrolador PIC18F4550, se muestran los trabajos que se han realizado sobre el robot balancín así como los controladores con los que fue controlado, la teoría de control, la teoría del filtro complementario, las convenciones utilizadas para el controlador PD-PI, la configuración de los módulos Xbee PRO S1, la lectura de los ejes del sensor inercial MPU6050 para obtener la posición del robot balancín, la lógica para que el robot balancín pueda seguir línea de trayecto, el diseño de la estructura, el diseño de la tarjeta PCB, la comunicación para obtener los datos con un módulo de radio frecuencia XBee PRO S1 y al final el análisis de los resultados experimentales así como las conclusiones.

(**Palabras clave:** robot balancín, microcontrolador PIC18F4550, control embebido, filtro complementario, controlador PD-PI, XBee PRO S1)

SUMMARY

This work presents construction, instrumentation and control of a line follower balancing robot with a PD master controller and PI speed slave loops, capable of taking 15 degree slopes with the code embedded in a PIC18F4550 microcontroller, the work that has been done is shown since the balancing robot as well as the controllers in which it was controlled, the control theory, the complementary filter theory, the conventions used for the PD-PI controller, the configuration of the Xbee PRO S1 modules, the reading of the axes of the inertial sensor MPU6050 to obtain the position of the balancing robot, the logic for the balancing robot to follow the path line, the design of the structure, the design of the PCB card, the communication to obtain the data with a radio frequency module XBee PRO S1 and finally the analysis of the experimental results as well as the conclusions.

(**Key words:** Balancing robot, PIC18F4550 microcontroller, Embedded control, Complementary filter, PD-PI controller, XBee PRO S1)

A Dios, a mi familia y amigos
—por todo el apoyo brindado.

AGRADECIMIENTOS

- Al profesor Dr. Roberto Valentín Carrillo Serrano por todos los consejos y retroalimentación en la redacción y experimentación de este trabajo, así como su amistad y constante motivación semana con semana.
- A la facultad de ingeniería por ayudarme a obtener los conocimientos necesarios para desarrollar este trabajo.
- Al grupo de profesores sinodales por su tiempo prestado para dar revisión y retroalimentación sobre este trabajo, así como sus consejos brindados.
- A mis compañeros, Becerril Escamilla José María, Gaytán Díaz José Santiago, Hernández Carapia Ramiro y Saucillo Castillo Arturo por la donación de material para construir el robot balancín.
- Al profesor MC. José Luis Avendaño coordinador de ingeniería en automatización por prestarme los módulos XBee PRO S1 y por guiarme en el proceso para presentar este trabajo.
- A mi hermano Marco Antonio Luna Moreno por el apoyo moral y económico brindado durante toda mi carrera universitaria.
- A mis amigas Claudia Beatríz Reséndiz Jurado, Paulina Pozas Yañez y hermana María del Carmen Luna Moreno por apoyarme, brindarme consejos y darme ánimo durante el desarrollo de este trabajo.

ÍNDICE GENERAL

ÍNDICE DE CUADROS

ÍNDICE DE FIGURAS

I. INTRODUCCIÓN

Según Čapek (1920) solemos imaginarnos a un robot basado en su aparición en el cine como una máquina que posee forma antropomórfica (androides), que actúa o parece actuar de forma autónoma y que parece interactuar con las personas (Richards y Smart, 2016), así un concepto tecnológico de robot que pueda abarcar todos los posibles supuestos podría ser el siguiente: un sistema que es capaz de percibir el entorno o contexto en el que se encuentra, que puede procesar la información para planificar una determinada actuación y ejecutarla (Bertolini, 2016). Dentro de las ramas que comprende la robótica se encuentran los denominados robots manipuladores y la robótica móvil (Baturone, 2005). A diferencia de los robots manipuladores, los robots móviles se caracterizan por no estar sujetos o fijos, es decir, pueden desplazarse por la tierra, en el agua o incluso volar libremente. Los aspectos más relevantes de la robótica móvil son los que están relacionados con el desplazamiento autónomo o navegación del robot. La robótica móvil tiene como meta principal lograr que el robot sea cada vez más independiente del operador, de modo que el operador sólo realice tareas de supervisión del comportamiento normal del robot, efectuar ajustes y correcciones (Pulido Fentanes et al., 2012).

La mayoría de los robots móviles terrestres usan tres o más ruedas, debido a que éstos por su naturaleza resultan ser estables, aún sin tener control sobre ellos. Sin embargo, están limitados en su movilidad para desplazarse en un ambiente humano. Un robot de dos ruedas proporciona una movilidad excelente, ya que puede realizar movimientos rápidos y suaves, tales como girar instantáneamente sobre su propio eje (Nozaki y Murakami, 2009). A pesar de ello, este tipo de sistemas son de naturaleza inestable, ya que son sistemas subactuados, es decir, tiene menos actuadores que grados de libertad (Fantoni et al., 2002).

Este tipo de robots corresponden al principio del péndulo invertido, el cual es un problema clásico en la teoría de control de sistemas subactuados. El péndulo invertido sobre

base móvil es uno de los sistemas más conocidos y estudiados por la comunidad de robótica y control automático, ya que además de ser un sistema no lineal es un sistema subactuado y con restricciones holónomas (Li et al., 2012). Por lo anterior, el péndulo invertido sobre base móvil representa un gran reto para los investigadores pasando desde los diferentes enfoques para la obtención de un modelo matemático, la identificación de parámetros, hasta la aplicación de diferentes técnicas de control y observación, dando lugar al desarrollo de nuevos e innovadores vehículos conocidos como el Robot Móvil de Péndulo Invertido (RMPI) los cuales constituyen un desafío para la teoría e ingeniería de control.

I.1. Antecedentes

La movilidad urbana es un tópico de interés actual y en ese sentido los RMPI están jugando un papel muy importante (Segway, N.d.). De ahí, surge el interés por lograr la construcción de un robot auto-balanceable de menor costo que sirva para el transporte personal (Clark et al., 2005), así como de tamaño a escala que sirva para implementar diferentes técnicas de control (García y Montiel, 2014).

En el caso particular de Clark et al. (2005) en Australia, lograron la construcción de un robot móvil auto-balanceable de dos ruedas, al que llamaron ”EDGAR CAR” capaz de transportar a una persona, se inspiraron en diseñar y construir un robot móvil que tuviera características similares a las del Segway Personal Transporter (Segway PT) (ver Figura 1.1), reduciendo el costo del móvil, mejorando el rendimiento de la energía e implementaron diferentes tipos de controladores con la ayuda de un giroscopio. Otro estudio realizado a este tipo de robot es el trabajo presentado por (Draz et al., 2012), donde muestran el diseño, construcción y control del robot, así como el análisis de deflexión del material cuando el móvil está sujeto a una carga de $200\,[N]$, implementando el controlador en un dispositivo ATMega16.

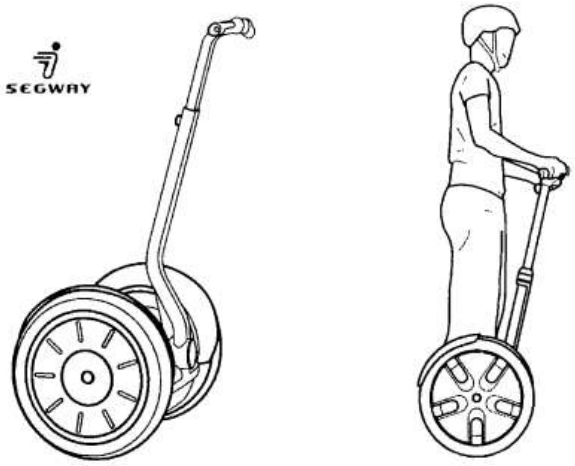

Figura 1.1: Segway Personal Transporter (Martínez y Martínez, 2010).

El robot balancín o $RMPI$ es un sistema altamente complejo, el cual ha sido atacado con diferentes controladores a lo largo de los años, buscando tener una mejor respuesta del sistema en el tiempo, en la Tabla 1.1 y 1.2 en la primer columna se muestran los trabajos de investigación con prototipos experimentales en los cuales se ha logrado mantener en equilibrio el robot balancín, en la segunda columna los controladores que se han aplicado y en la tercer columna los dispositivos en los cuales se han implementado los controladores. Como podemos ver el robot balancín ha sido estudiado con controladores simples y complejos implementados en diferentes dispositivos, algunos dispositivos tienen más baja capacidad de procesamiento que otros como en Bhatti et al. (2015), donde se usa un PIC18F452 para implementar un PID autosintonizado, pero este solo puede mantenerse en equilibrio y no puede seguir línea o tomar pendientes, otro trabajo donde se usa un dispositivo de baja capacidad se muestra en Chee et al. (2006), en el cual se usa un PIC16F877A, donde se implementa un controlador PID y al igual que en Bhatti et al. (2015) solo puede mantenerse en equilibrio, así existen muchos trabajos de investigación donde el robot balancín carece de la capacidad de seguir línea y tomar pendientes.

Tabla 1.1: Trabajos de investigación donde el robot balancín se mantiene en equilibrio.

Trabajo.	Controlador.	Dispositivo.
Chee et al. (2006)	Controlador PID para cada rueda, con un periodo de muestreo de $20\,[ms]$	PIC16F877A
Ruan et al. (2008)	PID con redes neuronales difusas	No definido
Xiaogang et al. (2008)	Controlador LQR	TMS320F2812DSP
Min et al. (2010)	Controlador LQR usando Lagrange	DSP-2812
Miasa et al. (2010)	Controlador lógica difusa	DSPIC30F2010
Nasir et al. (2011)	Comparación de rendimiento entre un controlador por lógica difusa y un controlador PID	Simulación
Feng et al. (2011)	Espacio de estados usando las leyes del movimiento de Newton	ATMEGA128A
Wu y Jia (2011)	Control adaptable por redes neuronales	Simulación
Wu et al. (2012)	Controlador por lógica difusa en cascada con un controlador PD	DSP
Ding et al. (2012)	Controlador de retroalimentación de estado completo con filtro Kalman	Arduino Mega 1280
Wasif et al. (2013)	Controlador PID con filtro complementario	Arduino uno
Adeel et al. (2013)	Controlador PID con representación de espacio de estados	ATmega16
Iwendi et al. (2019)	Controlador PD en cascada con controlador PI	DSP TMS320F2812
Sun et al. (2013)	Comparación entre controlador LQR y controlador LQR-NN	TMS320CDSP2407
Juang y Lum (2013)	Controlador PID y PI-PD con LQR	ATmega2560
Mahler y Haase (2013)	Espacio de estados fusionando las medidas del sensor para calcular el ángulo de inclinación	No definido
Azimi y Koofigar (2013)	Comparación de controlador MPC y controlador PD	Simulación

Tabla 1.2: Trabajos de investigación donde el robot balancín se mantiene en equilibrio.

Trabajo.	Controlador.	Dispositivo.
Fang (2014)	Controlador LQR basado en el algoritmo optimización de enjambres	Simulación
Jamil et al. (2014)	Comparación entre controlador PID y controlador LQR	Simulación
Ruan y Li (2014)	Controlador por lógica difusa con evasor de obstáculos	TMS320F28335
Sun et al. (2014)	Controlador LQR y Filtro Kalman	MK60DN512VLQ10
Gong et al. (2015)	Comparación de controlador PID y controlador LQR	Simulación
Bhatti et al. (2015)	Comparación de controlador PID y controlador PID auto- sintonizado	PIC18F452
Qiu y Huang (2015)	Controlador PID adaptable difuso	TMS320LF2407 DS
Ivoilov et al. (2016)	Control por retroalimentación	STM32F103
Schinstock et al. (2016)	Controlador PID en cascada con un controlador PI	Stm32f4 microcontroller
Martins y Nunes (2017)	Controlador PID	Arduino Pro Mini
Gonzalez et al. (2017)	Controlador LQR en cascada con controlador PI	Arduino Mega 2560
Park y Cho (2018)	Controlador PID con observador	TMS320F2808
Sarathy et al. (2018)	Controlador PID con retroalimentación negativa	myRIO
Anisimov et al. (2018)	Controlador por lógica difusa en cascada con controlador PD	STM32F4
Mai et al. (2019)	Controlador PID por lógica difusa	STM32F4
Ordóñez Cerezo et al. (2019)	Controlador PD	FPGA iCE40HX4K-TQ144
Philippart et al. (2019)	Comparación entre controlador PID y controlador LQR	Arduino due cortex-M3

Tabla 1.3: Trabajos donde el robot balancín sigue línea.

Trabajo.	Controlador.	Dispositivo.	Sigue línea.
Ghani et al. (2011)	Controlador PID	ATMEGA32	Sensor IR
Hsu et al. (2018)	Controlador por lógica difusa	Arduino Mega 2560	Cámara
Hasanah et al. (2018)	Controlador por lógica difusa y controlador en cascada PID-PD	No definido	CCD cámara
Binugroho et al. (2015)	Controlador en cascada PI-PID-PD	ATMega 128	Módulo RGB
Jung et al. (2011)	Controlador PD para control de posición y controlador PID para control de orientación	DSP	Cámara con sensores UV

En la Tabla 1.3 se muestra en la primer columna los trabajos de investigación en donde el robot balancín tiene la capacidad de seguir línea, en la segunda columna tenemos los controladores aplicados, en la tercera columna tenemos los dispositivos donde se implementó el controlador y en la cuarta columna tenemos los sensores que se usaron para poder seguir línea, como podemos ver en Hasanah et al. (2018), Hsu et al. (2018) y Jung et al. (2011) se necesita de un dispositivo con capacidad para procesar imágenes, ya que se usa una cámara. En Ghani et al. (2011) y Binugroho et al. (2015) solo son entradas digitales y analógicas, debido a que se usan sensores con salidas digitales.

Uno de los trabajos de investigación donde el robot balancín sigue línea y es capaz de subir pendientes es Hasanah et al. (2018), en el cual se implementó un controlador de lógica difusa y un PID, pero en su trabajo de investigación no viene en que dispositivo lo implementaron, sin embargo se usó visión para poder subir pendientes, lo que implica gran cantidad de procesamiento. En este trabajo se controla un robot balancín seguidor de línea capaz de tomar

pendientes de 15 grados en un PIC18F4550.

I.2. Descripción del problema

El robot balancín es un sistema complejo como se puede observar en (2.15) que ha sido modelado por diferentes trabajos de investigación como podemos ver en Patete et al. (2011); Sundin y Thorstenson (2012), entre otros.

Así mismo entre los controladores reportados en la literatura se encuentran diferentes técnicas de control, entre los controladores que se han aplicado en el robot balancín se tienen: el controlador PID (Chee et al., 2006), controlador por espacio de estados (Feng et al., 2011), controlador por lógica difusa (Ruan y Li, 2014), controlador LQR (Min et al., 2010), controlador por redes neuronales difusas (Ruan et al., 2008), controlador PID autosintonizado (Bhatti et al., 2015), entre otros.

Para poder implementar estos controladores es necesario un dispositivo que pueda procesar la información y realizar los cálculos necesarios en un tiempo prudente para este sistema, es decir con periodos de muestreo menores a los $20\,[ms]$ por lo que han sido implementados para el robot balancín controladores en dispositivos como FPGA (Ordóñez Cerezo et al., 2019), arduino uno (Wasif et al., 2013), arduino mega (Ding et al., 2012), PIC16F877A (Chee et al., 2006), entre otros. Entre estos trabajos de investigación existen robots del tipo balancín que son capaces de seguir línea como se muestra en la Tabla 1.3 e incluso tomar pendientes (Hasanah et al., 2018), pero utilizan técnicas que requieren procesar mucha información.

En este trabajo se controla un robot balancín seguidor de línea capaz de tomar pendientes de 15 grados con controladores simples para realizar el control, capaz de seguir línea de trayecto con sensores infrarrojos con el fin de disminuir la cantidad de información que va a procesar el microcontrolador, destacando que la estrategia de control está embebida en su totalidad en un PIC18F4550.

I.3. Hipótesis del trabajo

Se puede controlar un robot balancín seguidor de línea que sube pendientes de 15 grados con un PIC18F4550 con control clásico en cascada PD-PI..

I.4. Objetivo

Construir, instrumentar y controlar un robot balancín con un PIC18F4550 para seguir línea de trayecto y subir pendientes de 15 grados.

I.5. Objetivos específicos

1. Construir un robot balancín con placas de acrílico y tornillería galvanizada para usarlo como prototipo experimental.

2. Instrumentar el robot balancín con una unidad inercial y con encoders en los motores para medir la posición y velocidad de las ruedas en un *protoboard*.

3. Realizar una PCB con el método de planchado para reducir el ruido que se genera en las señales de la unidad inercial.

4. Diseñar pista con madera MDF para probar el funcionamiento del robot balancín.

II. FUNDAMENTACIÓN TEÓRICA

II.1. Modelado y reducción de modelo matemático

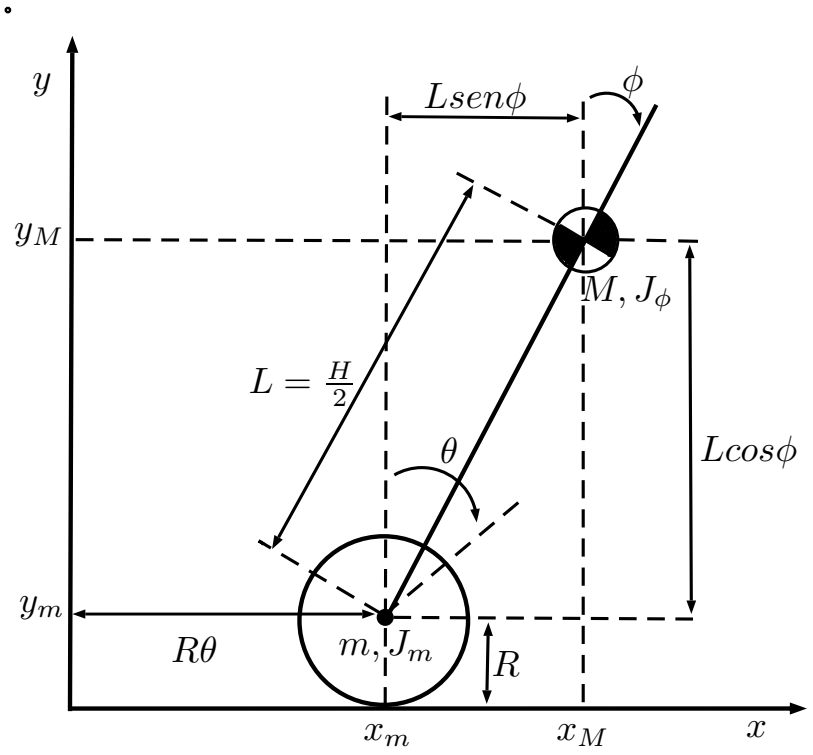

Figura 2.1: Vista lateral del péndulo invertido sobre dos ruedas (Patete et al., 2011).

En esta sección se presenta el modelado matemático del sistema, donde se toma como referencia el modelo utilizado por Patete et al. (2011). En la Figura 2.1, se muestra la vista lateral y el sistema de coordenadas sobre el cual se elabora el modelo matemático para el péndulo invertido sobre dos ruedas. En la Tabla 2.1 se muestran los parámetros del sistema.

Tabla 2.1: Parámetros del Sistema.

Par.	Descripción.	Unidad.
g	Aceleración de la gravedad	$[\frac{m}{s^2}]$
m	Masa de la rueda	$[kg]$
R	Radio de las ruedas	$[m]$
J_m	Momento de inercia de las ruedas	$\frac{MR^2}{2}[kg.m^2]$
M	Masa del cuerpo del péndulo	$[kg]$
H	Altura del cuerpo del péndulo	$[m]$
L	Distancia del centro de masa al eje de las ruedas	$\frac{H}{2}[m]$
J_ϕ	Momento de inercia del péndulo	$\frac{ML^2}{3}[kg.m^2]$
J_m	Momento de inercia del motor DC	$[kg.m^2]$

Para obtener las ecuaciones de movimiento de Lagrange es necesario definir las coordenadas generalizadas del sistema, así como el lagrangiano. Las coordenadas generalizadas del sistema son:

- θ : Ángulo de rotación de las ruedas,
- ϕ : Ángulo de inclinación del cuerpo del péndulo,
- (x_i, y_i) : Posición de la rueda izquierda,
- (x_d, y_d) : Posición de la rueda derecha,
- (x_M, y_M) : Posición del péndulo invertido.

El sistema de coordenadas bidimensional sobre el que se realiza el modelo para la rueda derecha se muestra en (2.2) y para la rueda izquierda se muestra en (2.1).

$$(x_i, y_i) = (x_i, y_i) = (R\theta, R) \tag{2.1}$$

$$(x_d, y_d) = (x_i, y_i) = (R\theta, R) \tag{2.2}$$

La posición del péndulo invertido está vinculada con la posición de sus ruedas, por lo tanto $x_i = x_d = x_M \, e \, y_i = y_d = y_M$, así:

$$\begin{aligned}(x_M, y_M) &= (x_m + Lsin\phi, y_m + Lcos\phi) \\ &= (R\theta) + Lsin\phi, R + Lcos\phi)\end{aligned} \tag{2.3}$$

La energía cinética traslacional es:

$$T_1 = \frac{1}{2}m\dot{x}_i^2 + \frac{1}{2}m\dot{x}_d^2 + \frac{1}{2}m\dot{x}_M^2 + \frac{1}{2}m\dot{y}_M^2 \tag{2.4}$$

$$T_1 = \frac{1}{2}(R\dot{\theta}_i)^2 + \frac{1}{2}(R\dot{\theta}_d)^2 + \frac{1}{2}M(R\dot{\theta} + L\dot{\phi}cos\phi)^2 + \frac{1}{2}M(-L\dot{\phi}sin\phi)^2 \tag{2.5}$$

La energía cinética rotacional es:

$$T_2 = \frac{1}{2}J_m\dot{\theta}_d^2 + \frac{1}{2}J_m\dot{\theta}_i^2 + \frac{1}{2}J_m\dot{\phi}^2 \tag{2.6}$$

La energía potencial es:

$$\begin{aligned}U &= mgy_m + mgy_m + Mgy_M \\ &= 2mgR + Mg(R + Lcos\phi)\end{aligned} \tag{2.7}$$

El lagrangiano es definido como:

$$\begin{aligned}lag &= T_1 + T_2 - U \\ lag &= R\dot{\theta}^2 + \frac{1}{2}M(R\dot{\theta} + L\dot{\phi}cos\phi)^2 \\ &\quad + \frac{1}{2}M(-L\dot{\phi}sin\phi)^2 + J_m\dot{\theta}^2 \\ &\quad + \frac{1}{2}J_\phi\dot{\phi}^2 - 2mgR \\ &\quad - Mg(R + Lcos\theta)\end{aligned} \tag{2.8}$$

Las ecuaciones de Lagrange son:

$$\frac{d}{dt}\left(\frac{\partial Lag}{\partial \dot{\theta}}\right) - \frac{\partial Lag}{\partial \theta} = F_\theta \tag{2.9}$$

$$\frac{d}{dt}\left(\frac{\partial Lag}{\partial \dot{\phi}}\right) - \frac{\partial Lag}{\partial \phi} = F_\phi \tag{2.10}$$

Aplicando las Ecuaciones (2.10) y (2.9) a (2.8) se obtienen las ecuaciones de movimiento de Lagrange para el péndulo invertido sobre dos ruedas.

$$[(2m+M)R^2 + 2J_m]\ddot{\theta} + (MLRcos\phi)\ddot{\phi} - MLR\dot{\phi}^2 sin\phi = F_\theta \tag{2.11}$$

$$(MLRcos\phi)\ddot{\theta} + (ML^2 + J_\phi)\ddot{\phi} - MgLsin\phi = F_\phi \tag{2.12}$$

Tomando como referencia el modelado del péndulo invertido sobre un carro, realizado por Ogata y Sanchez (1987), se toma como fuerza externa el voltaje aplicado a las ruedas, el cual es la señal de control u.

Con estas modificaciones las ecuaciones de movimiento de Lagrange, para el péndulo invertido sobre dos ruedas, son:

$$[(2m+M)R^2 + 2J_m]\ddot{\theta} + (MLRcos\phi)\ddot{\phi} - MLR\dot{\phi}^2 sin\phi = u \tag{2.13}$$

$$(MLRcos\phi)\ddot{\theta} + (ML^2 + J_\phi)\ddot{\phi} - MgLsin\phi = 0 \tag{2.14}$$

Despejando $\ddot{\theta}$ y $\ddot{\phi}$ de (2.13) y (2.14) se obtiene:

$$\begin{aligned}
\ddot{\theta} &= \frac{(ML^2 + J_\phi)(u + MLR\dot{\phi}^2 sin\phi)}{\Delta} - \frac{(ML)^2 sin\phi cos\phi}{\Delta} \\
\ddot{\phi} &= -\frac{(MLRcos\phi)(u + MLR\dot{\phi}^2 sin\phi)}{\Delta} + \frac{MgLsin\phi[(2m+M)R^2 + 2J_m]}{\Delta}
\end{aligned} \tag{2.15}$$

con $\Delta = [(2m+M)R^2 + 2J_m](ML^2 + J_\phi) - (MLRcos\phi)^2$.

Sean las variables de estado:

$$x_1 = \phi,\ x_2 = \dot{\phi},\ x_3 = \theta,\ x_4 = \dot{\theta} \tag{2.16}$$

y escribiendo las ecuaciones del sistema no lineal en variables de estado, se tiene la siguiente representación:

$$x_1 = x_2 \tag{2.17}$$

$$x_2 = -\frac{(MLRcosx_1)(u + MLRx_2^2 sinx_1)}{\Delta} + \frac{MgLsinx_1[(2m + M) + 2J_m]}{\Delta} \tag{2.18}$$

$$x_3 = x_4 \tag{2.19}$$

$$x_4 = \frac{(ML^2 + J_\phi)(u + MLRx_2^2 sinx_1)}{\Delta} \tag{2.20}$$

con $A = \Delta = [(2m + M)R^2 + 2J_m](ML^2 + J_\phi) - (MLRcosx_1)^2$.

Se observa que el modelo obtenido es un modelo de cuarto orden, como era de esperarse. Esto debido a las dinámicas de posición y velocidad asociadas al péndulo y la posición y velocidad asociadas a la plataforma móvil (formada por las ruedas).

II.2. Filtro complementario

El filtro complementario es sencillo de implementar en un microcontrolador y proporciona la posición angular gracias a la combinación de la señal del giroscopio y del acelerómetro.

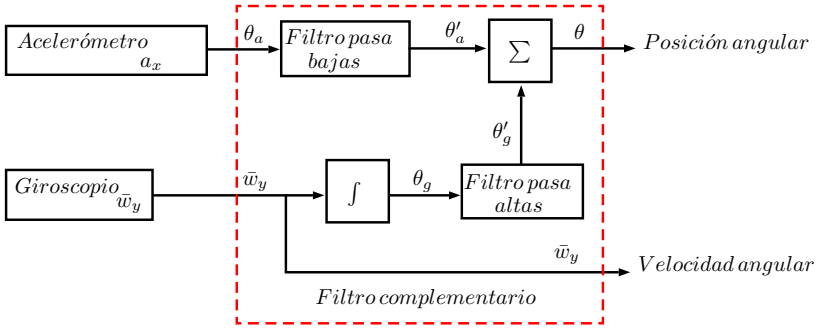

Figura 2.2: Diagrama de bloques del algoritmo del filtro complementario, donde obtenemos la velocidad angular y la posición angular del sensor MPU6050 (Palacios-Hurtado, 2017).

La respuesta de éste muestra algunos errores debidos a las señales del acelerómetro y del giroscopio, además de que se ven distorsionados por cualquier fuerza externa. El pro-

blema en el giroscopio es la deriva a lo largo del tiempo y el no retorno a cero producidos por la integración de la señal.

La combinación de las derivas del acelerómetro y del giroscopio se solucionan en el filtro complementario mediante la integración y el filtrado de la velocidad angular y una compensación utilizando la señal filtrada del acelerómetro (Palacios-Hurtado, 2017; Sundin y Thorstenson, 2012). La ecuación del filtro complementario utilizada se muestra en (2.21).

$$\phi_g = 0.999\left[\phi_g(k-1) + g_y T\right] + 0.001\phi_{gax} \tag{2.21}$$

$$\phi = (\phi_g)\left(\frac{\pi}{180}\right) \tag{2.22}$$

II.3. Controlador PID

Este controlador tiene una función de transferencia (Dorf y Bishop, 2011):

$$G_c(s) = K_P + \frac{K_I}{s} + K_D s \tag{2.23}$$

La ecuación para la salida en el dominio del tiempo es:

$$u(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \tag{2.24}$$

donde e en el robot balancín es $\phi - \phi_d$.

El controlador de tres términos se denomina controlador PID porque contiene un controlador proporcional, un integral, y un término derivativo ponderado por K_P, K_I y K_D, respectivamente. La función de transferencia del término derivativo es en realidad:

$$G_d(s) = \frac{K_D s}{\tau_d s + 1} \tag{2.25}$$

Pero τ_d suele ser mucho más pequeño que las constantes de tiempo del propio proceso, por lo que es descuidado.

Si establecemos $K_D = 0$, entonces tenemos el controlador proporcional integral (PI):

$$G_c(s) = K_P + \frac{K_I}{s} \tag{2.26}$$

Cuando $K_I = 0$, tenemos:

$$G_c(s) = K_P + K_D s \tag{2.27}$$

que se denomina controlador proporcional derivativo (PD).

El controlador PID también se puede ver como un controlador en cascada de los controladores PI y PD. Considere el controlador PI:

$$G_{PI}(s) = \hat{K}_P + \frac{K_I}{s} \tag{2.28}$$

y el controlador PD:

$$G_{PD}(s) = \bar{K}_P + \bar{K}_D s \tag{2.29}$$

donde $\hat{K}_P$ y $\hat{K}_I$ son las ganancias del controlador PI y $\bar{K}_P$ y $\bar{K}_D$ son las ganancias del controlador PD. En cascada los dos controladores (es decir, colocándolos en serie) se obtiene:

$$\begin{aligned} G_c(s) &= G_{PI}(s) G_{PD}(s) \\ &= \left(\hat{K}_P + \frac{\hat{K}_I}{s} \right) (\bar{K}_P + \bar{K}_D s) \\ &= (\bar{K}_P \hat{K}_P + \hat{K}_I \bar{K}_D) + \hat{K}_P \bar{K}_D s + \frac{\hat{K}_I \bar{K}_D}{s} \\ &= K_P + K_D s + \frac{K_I}{s} \end{aligned} \tag{2.30}$$

donde tenemos las siguientes relaciones entre las ganancias de los controladores PI, PD y PID:

$$\begin{aligned} K_P &= \bar{K}_P \hat{K}_P + \hat{K}_I \bar{K}_D \\ K_D &= \hat{K}_P \bar{K}_D \\ K_I &= \hat{K}_I \bar{K}_D \end{aligned} \tag{2.31}$$

Considere el controlador PID:

$$\begin{aligned} G_c(s) &= K_P + \frac{K_I}{s} + K_D s = \frac{K_D s^2 + K_P s + K_I}{s} \\ &= \frac{K_D(s^2 + as + b)}{s} = \frac{K_D(s + z_1)(s + z_2)}{s} \end{aligned} \tag{2.32}$$

donde $a = \frac{K_P}{K_D}$ y $b = \frac{K_I}{K_D}$. Por lo tanto, un controlador PID introduce una función de transferencia con un polo en el origen y dos ceros que se pueden ubicar en cualquier lugar en el plano (Dorf y Bishop, 2011).

II.4. Controlador y convenciones usadas

Partiendo del modelo propuesto en el Capítulo 2.1 se llegó a las convenciones de la Figura 2.3.

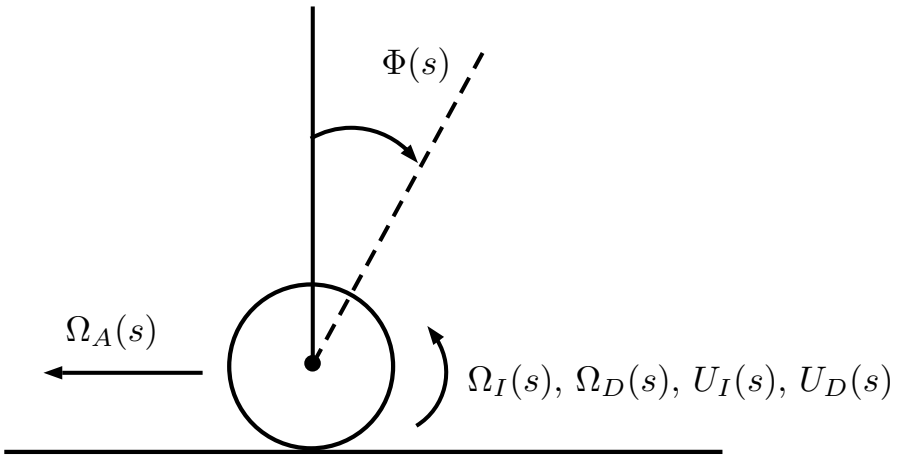

Figura 2.3: Convenciones propuestas.

Tabla 2.2: Nomenclatura de la Figura 2.3.

Expresión.	Descripción.
$U_I(s)$	Voltaje aplicado al motor izquierdo
$U_D(s)$	Voltaje aplicado al motor izquierdo
$\Omega_A(s)$	Velocidad de avance
$\Omega_D(s)$	Velocidad de arranque motor derecho
$\Omega_I(s)$	Velocidad de arranque motor izquierdo
$\Phi(s)$	Ángulo del balancín

Con las convenciones de la Figura 2.3 se diseñó el controlador para el robot balancín. El controlador que se diseñó es un controlador maestro PD en cascada con dos controladores esclavos PI.

Controlador maestro PD

El controlador maestro PD que se diseñó para controlar el ángulo del robot balancín se puede ver en la Figura 2.4. El controlador maestro PD tiene como entrada el ángulo deseado del robot balancín $\Phi_d(s)$, como retroalimentación el ángulo del robot balancín $\Phi(s)$ y como salida la velocidad deseada de los lazos de velocidad ($\Omega_{dI}(s)$ y $\Omega_{dD}(s)$). En este caso se necesita que el robot balancín avance, por esta razón se suma $\Omega_A(s)$ a la salida del controlador maestro PD para que los motores respondan cuando $\Phi_d(s)$ esta cerca de $\Phi(s)$.

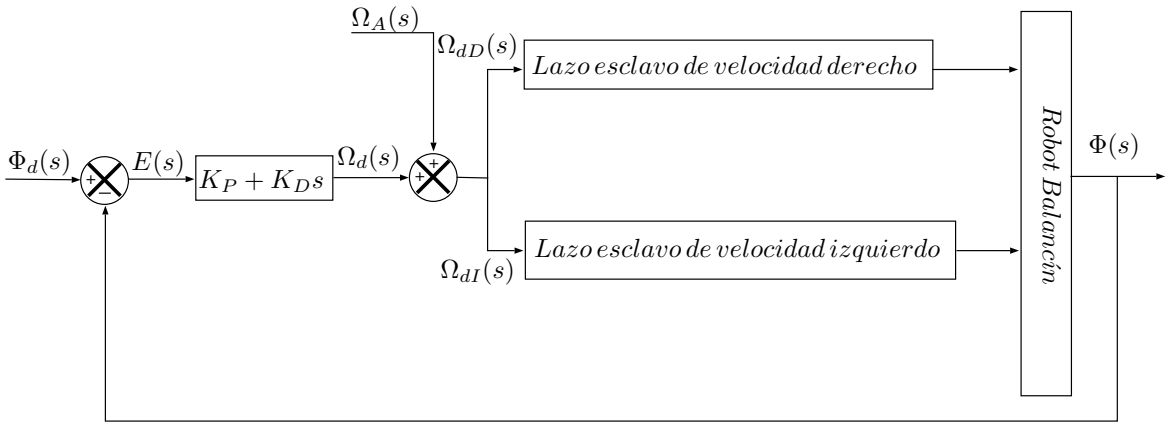

Figura 2.4: Diagrama de bloques controlador maestro PD.

Tabla 2.3: Nomenclatura de la Figura 2.4.

Expresión.	Descripción.	Unidad.
$\Phi_d(s)$	Ángulo deseado del robot balancín	$[rad]$
$E(s)$	Error maestro	$[rad]$
$\Omega_d(s)$	Esfuerzo de control maestro	$[\frac{rad}{s}]$
$\Omega_A(s)$	Velocidad de avance	$[\frac{rad}{s}]$
$\Omega_{dI}(s)$	Velocidad deseada rueda izquierda	$[\frac{rad}{s}]$
$\Omega_{dD}(s)$	Velocidad deseada rueda derecha	$[\frac{rad}{s}]$
$\Omega_D(s)$	Velocidad rueda derecha	$[\frac{rad}{s}]$
$\Phi(s)$	Ángulo del balancín	$[rad]$

Controladores esclavos PI

Con la velocidad de cada rueda y la velocidad deseada del controlador maestro se diseñaron los controladores esclavos PI, la entrada del lazo derecho de velocidad PI es la velocidad deseada de la rueda derecha $\Omega_{dD}(s)$, entra al controlador PI derecho para determinar el voltaje $U_D(s)$ del motor derecho $M_D(s)$ y así mover la rueda derecha $R_D(s)$ para obtener la velocidad de la rueda derecha $\Omega_D(s)$ y controlar el ángulo del robot balancín, este mismo procedimiento se aplicó para la rueda izquierda (ver Figura 2.5).

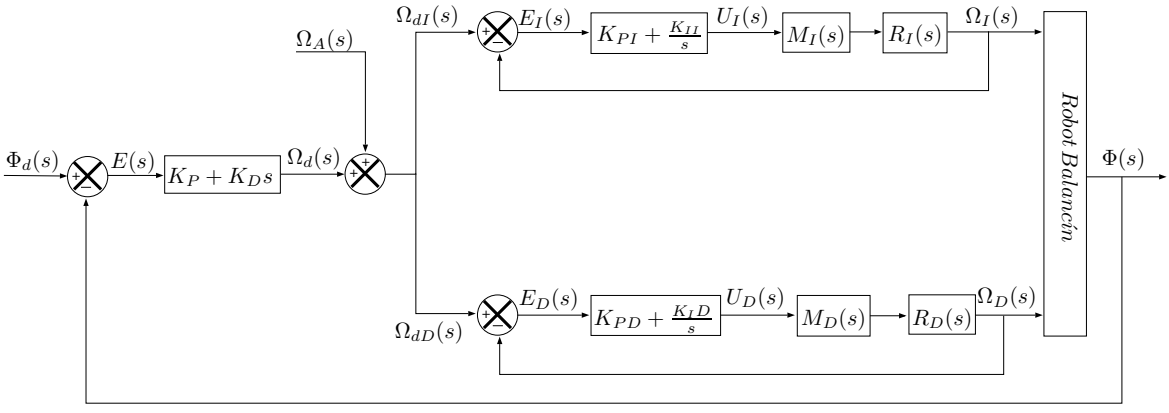

Figura 2.5: Diagrama de bloques del controlador maestro PD y lazos esclavos de velocidad PI.

Tabla 2.4: Nomenclatura de la Figura 2.5.

Expresión.	Descripción.	Unidad.
$\Phi_d(s)$	Ángulo deseado del robot balancín	$[rad]$
$E(s)$	Error maestro	$[rad]$
$\Omega_d(s)$	Esfuerzo de control maestro	$[\frac{rad}{s}]$
$\Omega_A(s)$	Velocidad de arranque	$[\frac{rad}{s}]$
$\Omega_{dI}(s)$	Velocidad deseada rueda izquierda	$[\frac{rad}{s}]$
$E_I(s)$	Error esclavo izquierdo	$[\frac{rad}{s}]$
$U_I(s)$	Voltaje aplicado a motor izquierdo	$[V]$
$M_I(s)$	Motor izquierdo	s/u
$R_I(s)$	Rueda izquierda	s/u
$\Omega_I(s)$	Velocidad rueda izquierda	$[\frac{rad}{s}]$
$\Omega_{dD}(s)$	Velocidad deseada rueda derecha	$[\frac{rad}{s}]$
$E_D(s)$	Error esclavo derecho	$[\frac{rad}{s}]$
$U_D(s)$	Voltaje aplicado a motor derecho	$[V]$
$M_D(s)$	Motor derecho	s/u
$R_D(s)$	Rueda derecha	s/u
$\Omega_D(s)$	Velocidad rueda derecha	$[\frac{rad}{s}]$
$\Phi(s)$	Ángulo del balancín	$[rad]$

Por último para poder modificar la velocidad deseada de cada rueda se agregaron las siguientes variables; KVD es la ponderación que modifica la velocidad deseada de la rueda derecha y KVI es la ponderación que modifica la velocidad deseada de la rueda izquierda (ver Figura 2.6). La lógica que se siguió para modificar KVD y KVI se puede ver más adelante en 3.2.6.

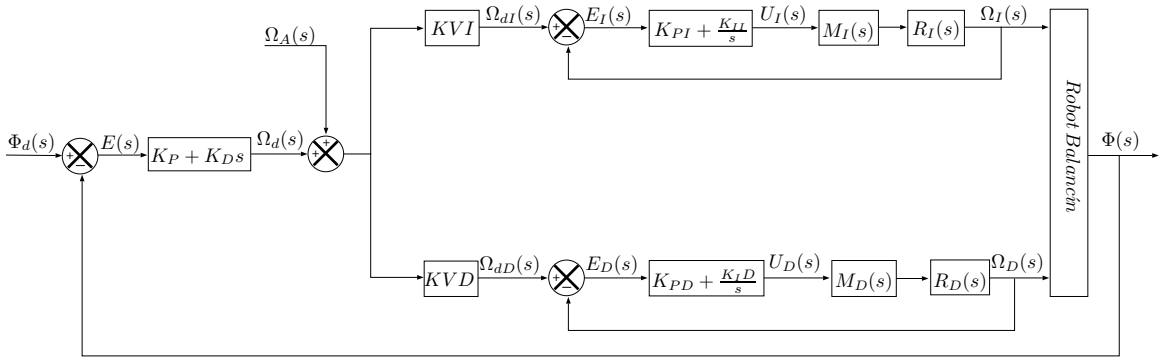

Figura 2.6: Diagrama de bloques del controlador en cascada PD-PI que se aplicó al robot balancín.

III. MATERIALES Y MÉTODOS

III.1. Materiales

Tabla 3.1: Equipo de cómputo.

Dispositivo.	Características.
Laptop HP pavilion G4	Procesador AMD A10-4600M RAM instalada 8.00 GB Frecuencia del procesador 2.3 GHz Windows 10 Home

Tabla 3.2: Programas.

Programa.	Versión.
PIC C Compiler	5.0
Dev-C++	5.5
Matlab	2017b
DIGI XCTU	s/v
Pickit 3	3.10
Texmaker	5.0.1
LaTeXDraw	3.3
Solidworks	2018
Autocad	online
Easy EDA	6.4.25

Tabla 3.3: Componentes eléctricos y electrónicos .

Cantidad.	Descripción.
1	Pickit 3
1	Cargador de baterias LI-PO MK3 PRO
1	PIC18F4550
1	MPU6050
2	XBee PRO S1
1	Adaptador XBee a USB
1	Adaptador XBee a protoboard
1	Puente H-L298N
1	LM7805
5	Sensores infrarrojo TCRT5000
1	Crystal $20\,[Mhz]$
2	Capacitores $15\,[pF]$
2	Resistencias $10\,[k\Omega]$
1	Resistencia $1\,[k\Omega]$
3	Tiras de pines hembra
10	Dupones hembra macho $15\,[cm]$
1	Cable para protoboard $0.5\,[m]$
1	Clema $5\,[mm]$-2
3	Switches ON-OFF
1	Batería LI-PO $7.4\,[V]$, $2200\,[mA]$
2	Motores de $6\,[V]$ a $2\,[A]$ con encoder 34:1 de $5\,[V]$

Tabla 3.4: PCB.

Cantidad.	Descripción.
1	Placa fenólica $10x10\,[cm]$
1	Cloruro férrico $0.5\,[l]$
1	Plancha
1	Hoja de papel transferencia
1	Cinta adhesiva transparente $12\,[mm]$

Tabla 3.5: Estructura del robot balancín.

Cantidad.	Descripción.
4	Espárragos de $8\,[cm]$ de largo y $5\,[mm]$ de diámetro
12	Tornillos de $1.27\,[cm]$ largo y $3\,[mm]$ de diámetro
2	Tornillos de $2\,[cm]$ largo y $2\,[mm]$ de diámetro
2	Tornillos de $4\,[cm]$ largo y $3\,[mm]$ de diámetro
16	Tuercas para espárragos de $5\,[mm]$
14	Tuercas para tornillos de $3\,[mm]$
2	Tuercas para tornillos de $2\,[mm]$
2	Acopladores para eje de motor $3\,[mm]$ diámetro
2	Sujetadores de motor
2	Llantas de caucho $7\,[cm]$ de diámetro
2	Placas de acrílico $5\,[mm]$ con $15\,[cm]$ largo y $7\,[cm]$ ancho
1	Llave de tuercas tipo perica
1	Desarmador de relojero

Tabla 3.6: Pista de pruebas.

Cantidad.	Descripción.
6	Hojas de papel tamaño carta
1	Cinta métrica
1	Compás
1	Resistol adhesivo $100\,[g]$
1	Segueta
1	Madera triplay de $10\,[mm]$ espesor y $1\,[m]$ de largo por $0.6\,[m]$ ancho
1	MDF de $5\,[mm]$ espesor y $0.3\,[m]$ de largo por $0.3\,[m]$ ancho
1	Bote de pintura en aerosol color negro mate de $0.4\,[l]$
1	Pistola de silicona
5	Barras de silicona

III.2. Métodos

III.2.1 Programa principal

Para poder controlar el robot balancín se diseño un programa en PIC C Compiler 5.0 (ver Apéndice A), siguiendo los siguientes pasos:

1. Se configuró el reloj interno del PIC18F4550 con PLL a $48\,[Mhz]$.

2. Se realizaron las configuraciones en el bus i2c para comunicarnos con el sensor inercial MPU6050, se configuró como maestro con una frecuencia de transmisión de $100\,[Mhz]$.

3. Se realizaron las configuraciones en puerto RS232 para obtener los resultados experimentales del robot balancín, se configuró a 115200 baudios, sin paridad y 8 bits de datos.

4. Se inicializó el sensor inercial MPU6050 con los valores de la Tabla 3.7.

5. Se configuró el timer 2 como modo captura compara para el pwm del motor derecho e izquierdo. Se configuraron ccp1 y ccp2 con una frecuencia de $2929.68\,[Hz]$ y un ciclo máximo en cuentas de 255.

6. Se habilitó la interrupción del timer 0 y la interrupción de la parte alta del puerto B, la interrupción del timer 0 se hace cada $5\,[ms]$ como periodo de muestreo y la interrupción de la parte alta del puerto B se usa para leer los encoders de las ruedas derecha e izquierda.

7. Para poder obtener la aceleración en el eje x (a_x) y velocidad angular en el eje y ($\bar{w}_y$), se manda llamar la función $mpu6050datos()$ en el ciclo while del programa principal.

8. Para seguir línea se manda llamar la función $Linea()$ que hace el mapeo de los sensores S_{DH}, S_{DL}, S_{IL} y S_{IH}, para detectar la rampa se creo la función $Rampa()$ que hace el mapeo del sensor S_P que detecta la pendiente y se mandan llamar al inicio de la interrupción del periodo de muestreo cada $5\,[ms]$.

9. Dentro de la interrupción del timer 0 primero se mandan llamar las funciones de seguidor de $Linea()$ y $Rampa()$, después se hace el cálculo del filtro complementario y seguido de esto se aplica el controlador con los arreglos para seguir línea y detectar la pendiente de 15 grados. Al final de la función se mandan los datos por el puerto serial al XBee PRO S1 para recibirlos en el programa en DevC++(ver Apéndice B) y ser graficados en matlab (ver Apéndice C).

III.2.2 Carga de programa a microcontrolador

La programación del microcontrolador se realizó en PIC C Compiler 5.0, en este compilador se crearon los códigos para controlar el robot balancín.

Para cargar el código en la memoria flash del PIC18F4550 se usó el programador Pickit 3 de microchip (ver Figura 3.1) con su programa Pickit 3.10.

Figura 3.1: Programador Pickit 3 con cable USB.

Los pasos que se siguieron para grabar el código en la memoria flash del microcontrolador se muestran a continuación:

1. Se creó el código en PIC C Compiler 5.0 con el programa para controlar el robot balancín, después se compiló y verificó que no hubiera errores (ver Figura 3.2).

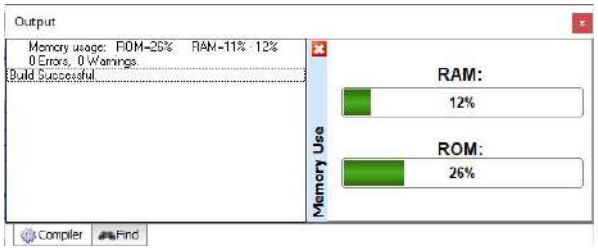

Figura 3.2: Salida de PIC C Compiler 5.0.

2. Se realizaron las conexiones del Pickit 3 al microcontrolador (ver Figura 3.3).

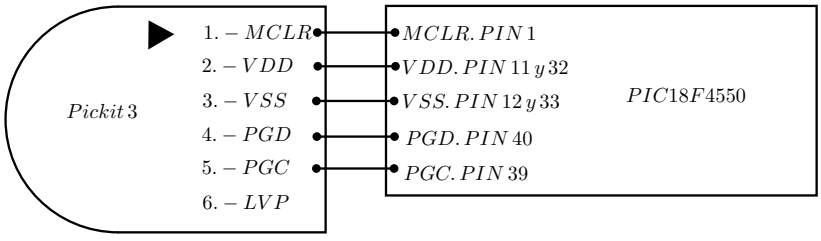

Figura 3.3: Conexiones del Pickit 3 a microcontrolador.

3. Se conectó el Pickit 3 a la computadora, en el programa Pickit 3.10 se seleccionó la pestaña tools y luego check comunications para detectar el microcontrolador (ver Figura 3.4).

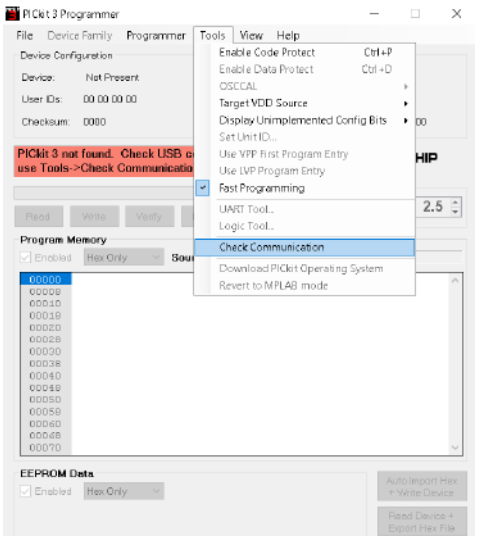

Figura 3.4: Ventana de Pickit 3.10 para detectar el microcontrolador.

4. Para grabar el código en hexadecimal de PIC C Compiler 5.0 se seleccionó el botón Auto Import Hex + Write Device (ver Figura 3.5), una vez que se seleccionó se eligió el archivo en hexadecimal y seleccionamos aceptar.

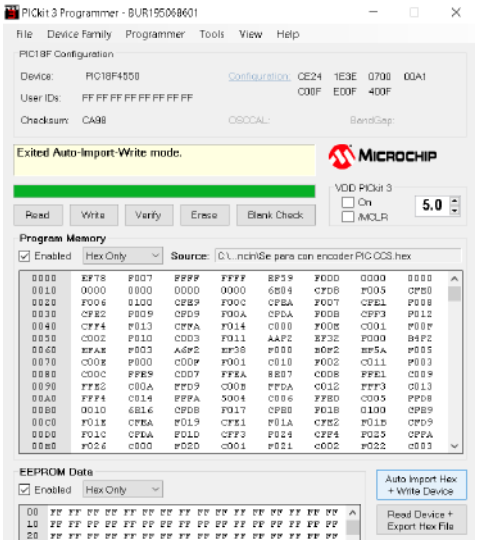

Figura 3.5: Ventana de Pickit 3.10 con botón Auto Import Hex + Write Device.

III.2.3 Configuración de módulos XBee PRO S1

La comunicación inalámbrica para obtener la respuesta en el tiempo se realizó con dos módulos XBee PRO S1 (ver Figura 3.6).

Figura 3.6: Módulos XBee PRO S1.

La configuración de los módulos XBee PRO S1 fue necesaria para realizar la comunicación entre los módulos XBee PRO S1. La comunicación se realizó punto a punto, se colocó un Xbee PRO S1 en el robot balancín y otro en la computadora.

Para conectar los módulos Xbee PRO S1 se usaron adaptadores de niveles lógicos de voltaje, en la conexión del módulo XBee PRO S1 a la computadora se usó un adaptador USB a Ft232rl (ver Figura 3.7 primera imagen) y para conectar el módulo XBee PRO S1 al PIC18F4550 se usó un adaptador XBee Sparkfun (ver Figura 3.7 segunda imagen).

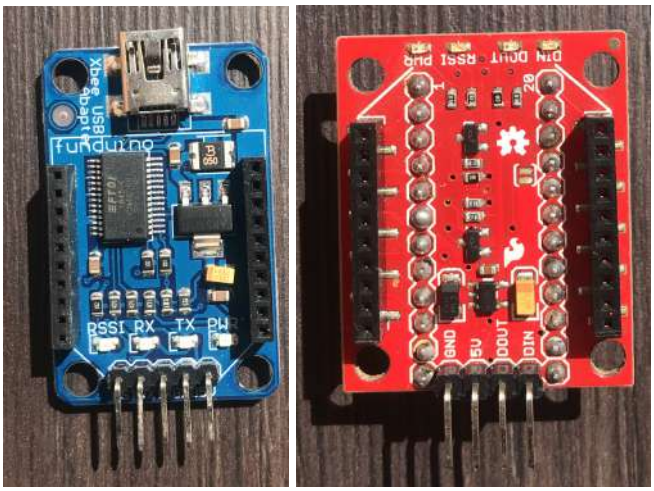

Figura 3.7: Adaptadores de niveles lógicos de voltaje XBee.

Para configurar el módulo XBee PRO S1 que se colocó en el robot balancín se siguieron los siguientes pasos:

1. Se conectó el módulo XBee PRO S1 a la computadora con el adaptador USB a Ft232rl.

2. En el programa DIGI XCTU se seleccionó el botón para agregar un nuevo módulo de radio (ver Figura 3.8), en esta ventana se seleccionó la opción de escribir el nombre del puerto manualmente y se colocó el nombre del puerto en el que esta conectado el módulo XBee PRO S1.

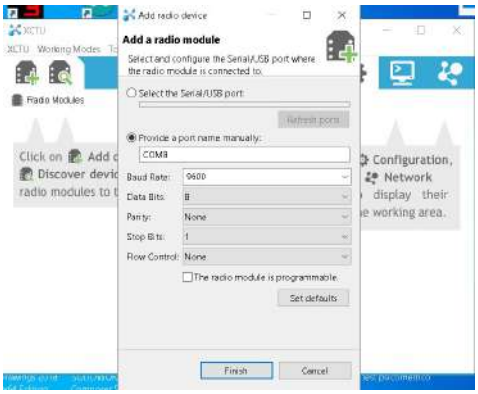

Figura 3.8: Ventana de DIGI XCTU para agregar un nuevo módulo XBee.

3. Seleccionamos el módulo XBee PRO S1 para ver las configuraciones que tiene actualmente, en esta ventana seleccionamos el botón actualizar programa y configuramos el

módulo XBee PRO S1. Elegimos la familia de nuestro módulo que en este caso es XBP24, la función que va a desempeñar que en este caso es adaptador RS232 y la versión del programa de nivel lógico que va a tener internamente el módulo XBee PRO S1 que en este caso es 11e8 (ver Figura 3.9).

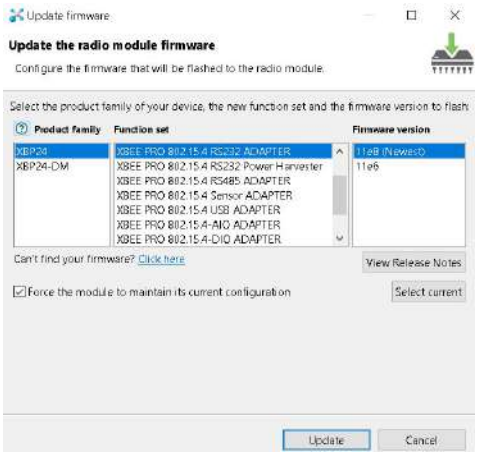

Figura 3.9: Ventana de configuración de familia, función y programa de XBee del COORDINADOR en DIGI XCTU.

4. En configuración de red y seguridad se colocó la dirección del módulo XBee PRO S1 al cual se va a conectar, se divide en parte alta (DH) de 32 bits y parte baja (DL) de 32 bits, parte alta es $13A200$ y la dirección baja es 40776938. Se eligió la opción (CE) para ser COORDINADOR y se cambio el nombre del identificador de nodo (NI) a COORDINADOR (ver Figura 3.10).

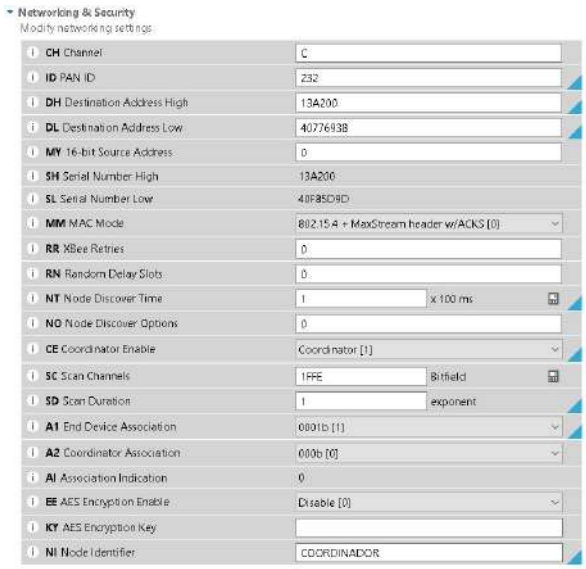

Figura 3.10: Ventana de configuración de red en DIGI XCTU.

5. La configuración de modo dormir se desactivo (ver Figura 3.11).

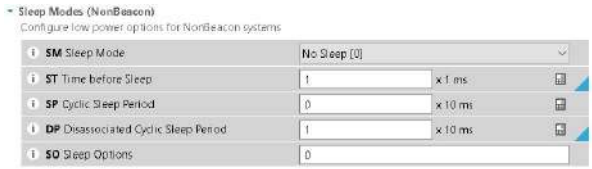

Figura 3.11: Ventana de configuración de modo dormir en DIGI XCTU.

6. La configuración serial (BD) se configuró a 115200 baudios, sin paridad (NB), se desactivó para que no almacene bytes en el buffer y se envíen los datos sin retraso (RO), de desactivó la configuración API (AP) y se activó la resistencia PULL-UP (PR).

7. Se desactivaron las entradas analógicas(ver Figura 3.12).

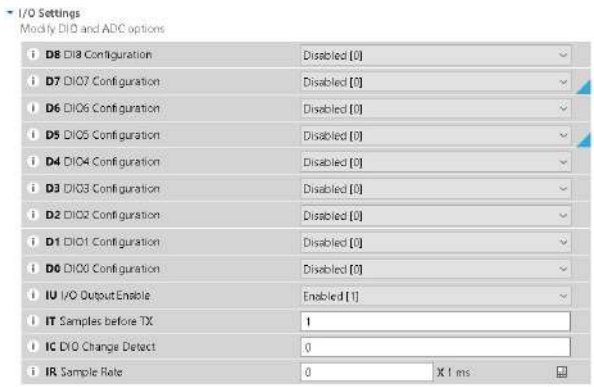

Figura 3.12: Ventana de salidas analógicas DIGI XCTU.

8. Se desactivaron las salidas de PWM (ver Figura 3.13), las ventanas de diagnostico y comandos AT se dejaron con las configuraciones por defecto.

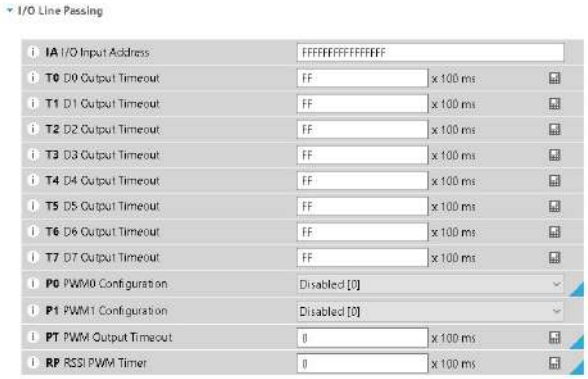

Figura 3.13: Ventana de salidas PWM DIGI XCTU.

Las configuraciones del módulo XBee PRO S1 que se colocó en la computadora fueron casi las mismas que el módulo XBee PRO S1 que se colocó en el robot balancín, soló cambiaron las configuraciones de dos ventanas.

La primer ventana que cambió fue en donde se configura la familia, función y programa del XBee PRO S1. En esta ventana cambió la función a adaptador USB y la versión del programa de nivel lógico que va a tener internamente el módulo XBee PRO S1 a 14ef (ver Figura 3.14).

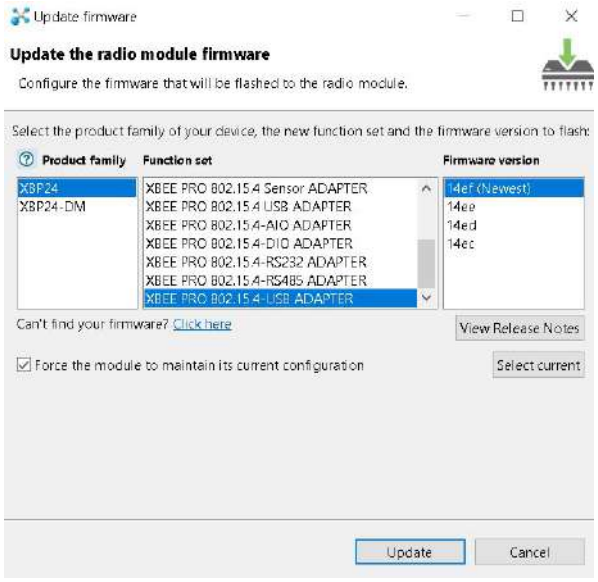

Figura 3.14: Ventana de configuración de familia, función y programa de XBee del DISPOSITIVO FINAL en DIGI XCTU.

La segunda ventana que cambió fue la de configuración de red, en esta ventana cambió la dirección baja (DL) a $40F85D9D$, la opción de (CE) se cambió a DISPOSITIVO FINAL y la opción de identificador de nodo se cambió a DISPOSITIVO FINAL (ver Figura 3.15).

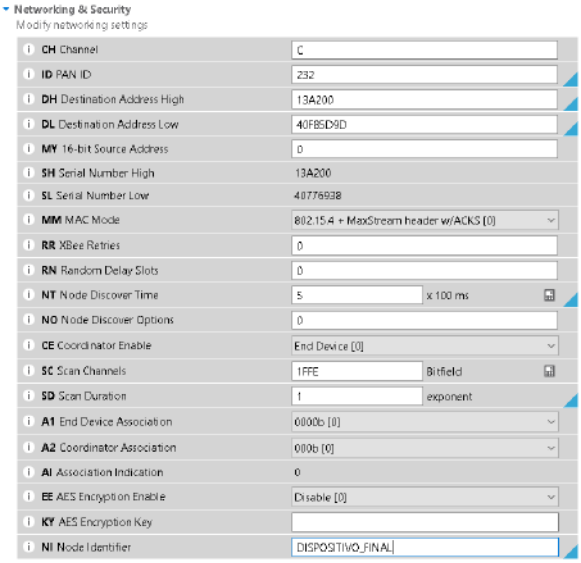

Figura 3.15: Ventana de configuración de red DIGI XCTU.

III.2.4 Posición del robot balancín

Para obtener el ángulo de inclinación del robot balancín se usó el filtro complementario y el sensor inercial MPU6050 (ver Figura 3.16), el cual puede leer la aceleración en los ejes x, y y z, la velocidad angular en los ejes x, y y z y la temperatura ambiente. Para poder leer los datos primero se configuró el sensor inercial MPU6050, para lo cual se usaron las direcciones de memoria de la Tabla 3.7.

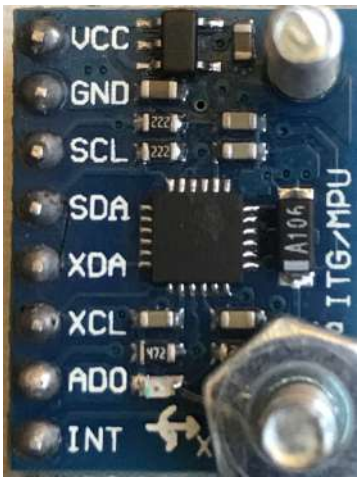

Figura 3.16: Sensor inercial MPU6050.

Tabla 3.7: Direcciones de memoria del sensor MPU6050.

Registro.	valor.
$0x1A$	0b00000000
$0x1B$	0b01000000
$0x1C$	0b10000000
$0x6B$	0b00000000

La dirección $0x1A$ se usó para desactivar el filtro pasa bajas interno del sensor inercial MPU6050, los primeros $3\,bits$ se igualaron a cero.
La dirección $0x1B$ se usó para configurar la escala de la velocidad angular, para configurar la escala se usaron los $bits\,3$ y 4, los cuales se pusieron en cero para tener una escala de $\pm\,250\,[\frac{^\circ}{s}]$. En este registro se configura la velocidad angular que se va a usar en los $bits\,6,\ 7$ y 8, se igualó el $bit\,7$ a 1, para configurar la velocidad angular en el eje y.

La dirección $0x1C$ se uso para configurar la escala de la aceleración, para configurar la escala se igualaron a cero los $bits\,3$ y 4, para tener una escala de $\pm\,2\,[g]$. En este registro se usó PLL en el eje x con el $bit\,8$ igualado a 1.

La dirección $0x6B$ se usó para configurar el reloj interno, la lectura de temperatura, el modo reiniciar y el modo dormir. Para la configuración del reloj se igualaron los últimos $3\,bits$ a cero para tener un reloj interno de $8\,[Mhz]$, la lectura de temperatura se desactivó igualando el $bit\,3$ a cero, el modo reiniciar y el modo dormir se desactivaron igualando los $bits\,6$ y 7 a cero.

La posición del sensor inercial en el robot balancín es muy importante, ya que determina que ejes puedes usar para aplicar el filtro complementario, la orientación que se usó se muestra en la Figura 3.17.

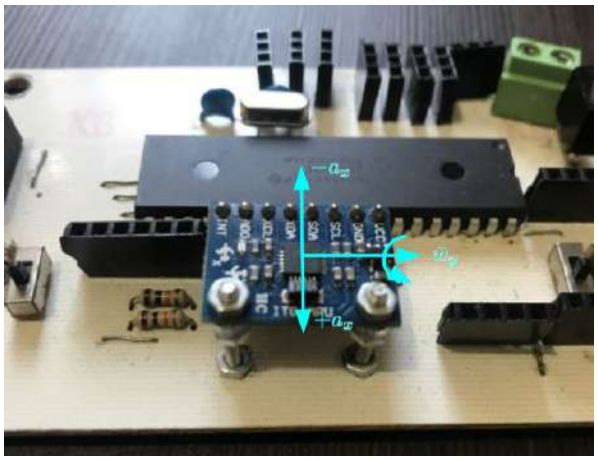

Figura 3.17: Posición del sensor inercial en el robot balancín.

Una vez que se configuró el sensor inercial MPU6050 se siguió el diagrama de la Figura 3.18 para obtener el ángulo de inclinación del robot balancín, primero se obtuvo la velocidad angular en el eje y ($\bar{w}_y$) y la aceleración en el eje x (a_x), después se divide la velocidad angular entre 131.0 para obtener $\pm\,250\,[\frac{^\circ}{s}]$ y la aceleración se multiplica por $\frac{-90.0}{16902.0}$ para obtener la posición en un rango de $\pm\,90\,[^\circ]$, estos valores entran al filtro complementario para obtener el ángulo de inclinación del robot balancín Φ (ver Capítulo 2).

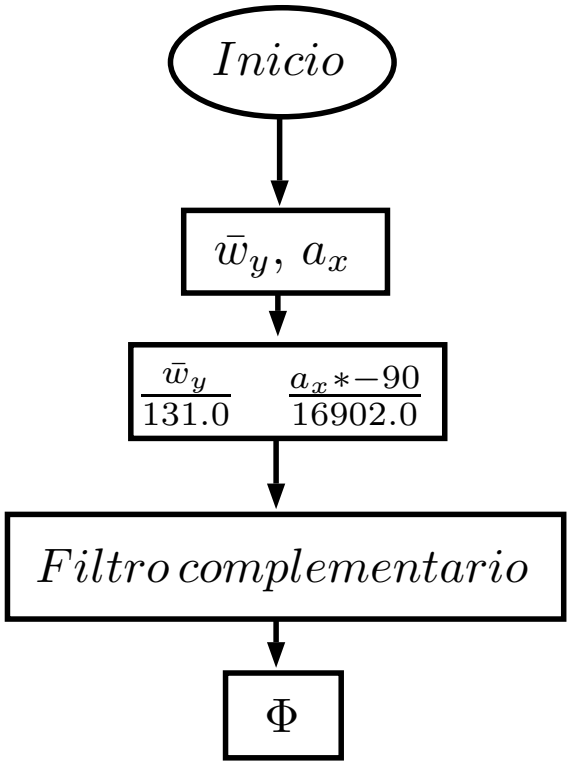

Figura 3.18: Diagrama para obtener el ángulo de inclinación del robot balancín.

III.2.5 Envió de información para graficar

Para la adquisición de información de control del robot balancín se siguió el diagrama de la Figura 3.19. Se enviaron un total de 10 $bytes$, en el primer $byte$ se envió una bandera para identificar el inicio de datos en la recepción, en el segundo byte se envió el ángulo deseado del robot balancín, en el tercer $byte$ se envió ángulo actual del robot balancín, en el cuarto $byte$ se envió la velocidad deseada de la rueda derecha, en el quinto $byte$ se envió la velocidad de la rueda derecha, en el sexto $byte$ se envió el voltaje aplicado al motor derecho, en el séptimo $byte$ se envió la velocidad deseada de la rueda izquierda, en el octavo $byte$ se envió la velocidad de la rueda izquierda, en el noveno $byte$ se envió el voltaje aplicado a la rueda izquierda y en el décimo $byte$ se envió el estado lógico de los sensores.

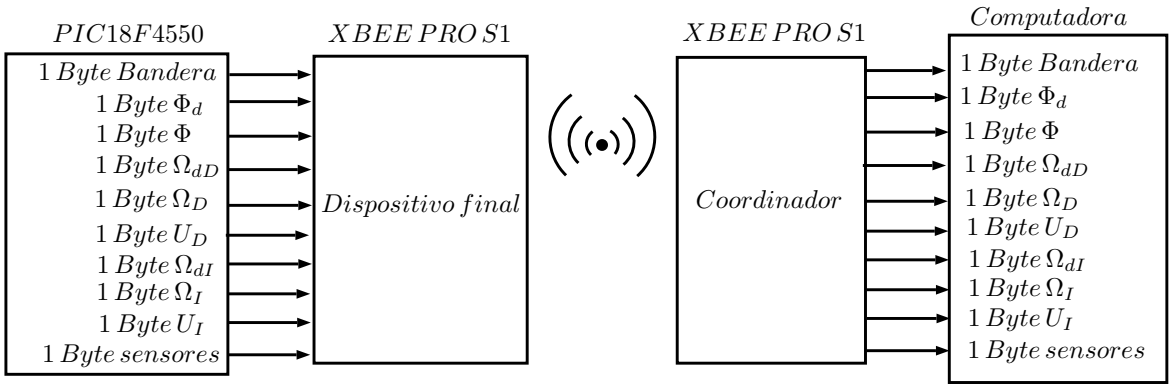

Figura 3.19: Diagrama de comunicación inalámbrica para obtener datos.

III.2.6 Seguidor de línea

Los pasos para que el robot balancín pudiera seguir línea se muestran a continuación:

1. Se eligieron los sensores TCRT5000 (ver Figura 3.20), ya que se pueden ajustar con el trimpot para que no se vean afectados por la luz del día.

Figura 3.20: Sensor infrarrojo TCRT5000

2. La ubicación de los sensores seguidores de línea se muestran en la Figura 3.21. S_{IH}, S_{IL}, S_{DH} y S_{DL} son los sensores que se usaron para seguir la línea de trayecto, RD es la rueda derecha y RI es la rueda izquierda.

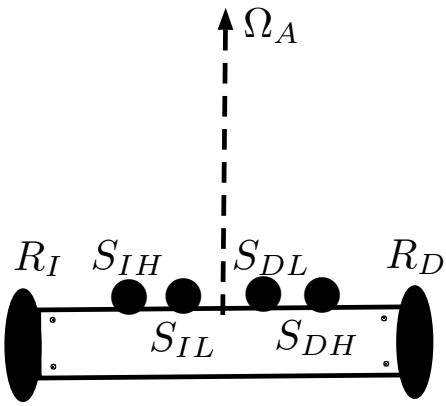

Figura 3.21: Diagrama de ubicación de sensores seguidores de línea.

3. Para colocar los sensores en el robot balancín se diseñó una pieza en Solidworks 2018 (ver Figura 3.22 y apéndice D), se mando imprimir en impresora 3D en material PLA color negro. Para poder atornillar la pieza al robot balancín se dejaron dos agujeros los cuales se pueden ver encerrados en color rojo, se dejó un espacio para poder pasar los leds de los sensores la cual se puede ver encerrada en color verde y se dejó un espacio para poder atornillar los sensores a la pieza el cual se puede ver encerrado en color amarillo.

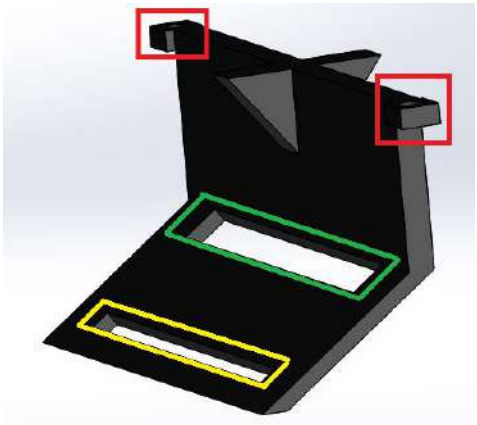

Figura 3.22: Diseño de pieza de sensores en Solidworks 2018.

4. Los sensores se ajustaron con el trimpot y un desarmador de cruz tipo relojero, estando colocados en el robot balancín y sobre la pista de pruebas.

5. La lógica que se usó se muestra en la Figura 3.23, primero se evalúan los sensores más cercanos a la línea S_{IL} y S_{DL} y si el sensor detecta se disminuye la velocidad deseada de la rueda a $40\,[\,\%]$, después se evalúan los sensores más alejados a la línea S_{IH} y S_{DH} donde si el sensor detecta se disminuye la velocidad deseada de la rueda a $20\,[\,\%]$.

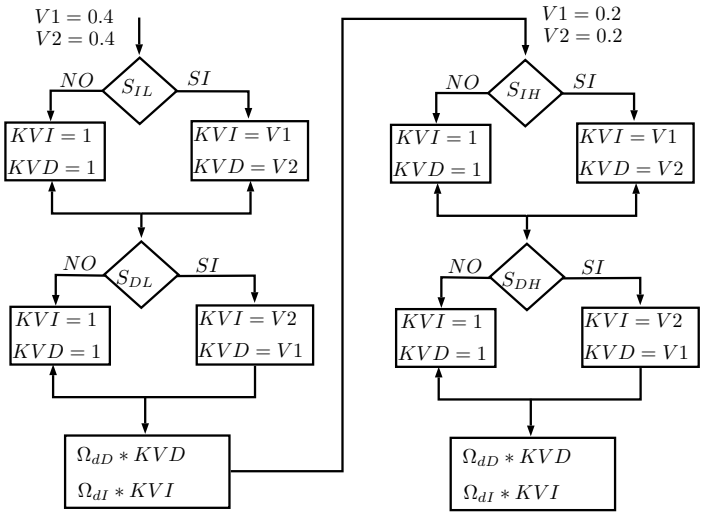

Figura 3.23: Lógica de sensores seguidores de línea.

6. El controlador general del sistema agregando las constantes KVD y KVI se puede ver en la Figura 2.6, se agregó KVD en el lazo de velocidad derecho y KVI en el lazo de velocidad izquierdo.

III.2.7 Plano inclinado

Los pasos que se siguieron para que el robot balancín fuera capaz de subir la pendiente de 15 grados se muestra a continuación:

1. El esquema de ubicación del sensor S_P que detecta la pendiente se puede ver en la Figura 3.24.

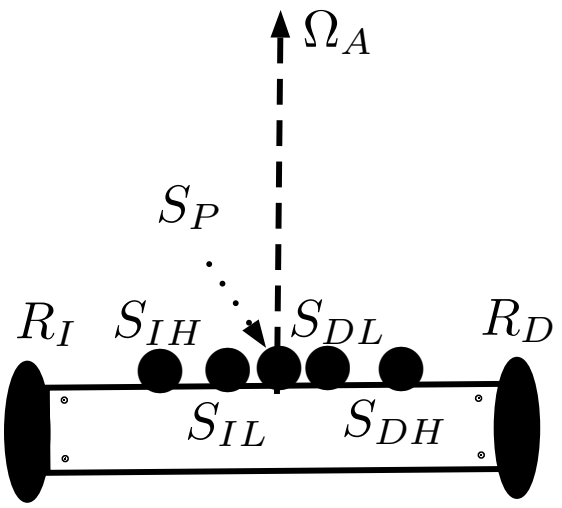

Figura 3.24: Esquema de ubicación de del sensor detecta pendiente SP.

2. Primero se colocó un sensor TCRT5000 en medio de los sensores seguidores de línea para poder detectar el plano inclinado.

3. El sensor se calibró con el trimpot y un desarmador de relojero, se ajustó la distancia de detención colocando el robot balancín justo donde inicia el plano inclinado (ver Figura 3.25).

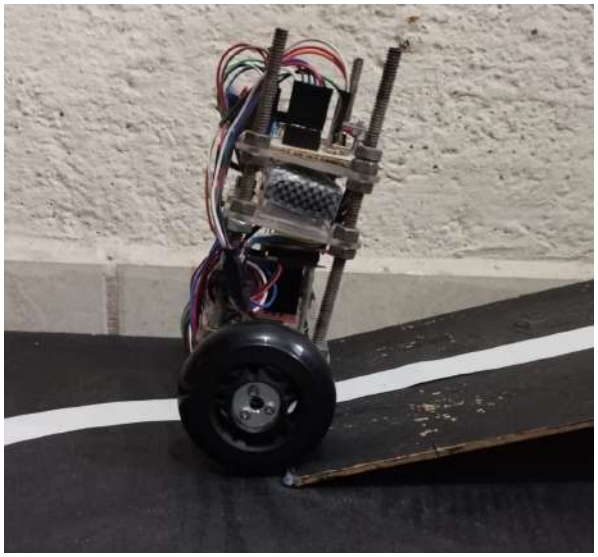

Figura 3.25: Robot balancín en posición de subir la pendiente de 15 grados.

4. Cuando el sensor detecta la pendiente de 15 grados se cambia Ω_A y Φ_d (ver Figura 3.26).

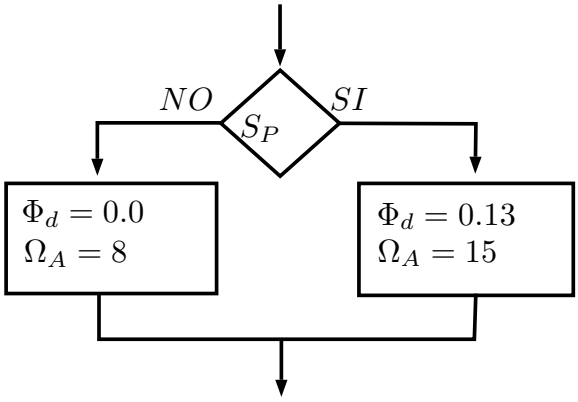

Figura 3.26: Diagrama de bloques cuando el robot balancín detecta la pendiente.

III.2.8 Estructura del robot balancín

La estructura se diseñó siguiendo las reglas del concurso robouaq 2020, las cuales especifica que el robot balancín debe de ser capaz de ser contenido en un cubo de $20\,[cm]$ y tener un peso menor a $1\,[kg]$.
Siguiendo estas dos reglas se siguieron los siguientes pasos para el diseñar la estructura:

1. Se diseñó la estructura con tres niveles (ver Figura 3.27).

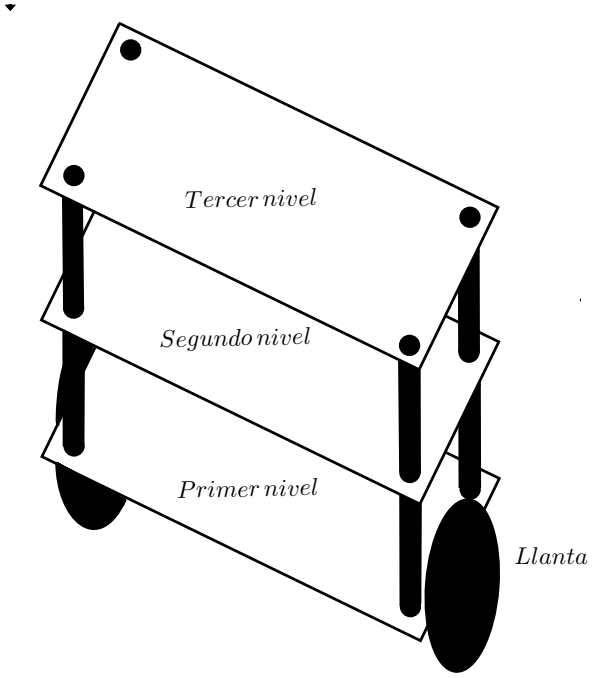

Figura 3.27: Esquema de estructura del robot balancín.

2. Las dimensiones de cada nivel se diseñaron de $15\,[cm]$ de largo y $7\,[cm]$ de ancho en acrílico de $5\,[mm]$ de espesor. En la Figura 3.28 se puede ver el diseño del primer nivel, encerrado con color rojo están los cuatro orificios que se crearon para atornillar el Puente H-L298N (ver Figura), encerrado con color verde están los 8 orificios que se crearon para atornillar los sujetadores de cada motor, encerrado en color morado están dos espacios que se crearon para pasar los cables de los encoders y sensores infrarrojo seguidores de línea y encerrado en color azul esta los 4 orificios que se crearon para pasar los espárragos.

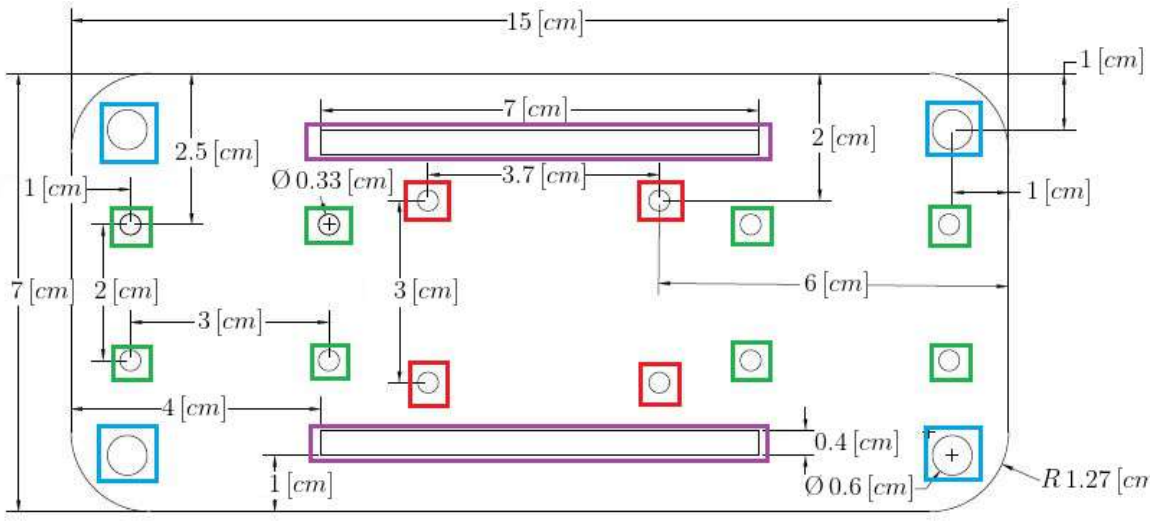

Figura 3.28: Diseño del primer nivel del robot balancín en AutoCAD online.

3. Se diseñó el segundo nivel de la estructura del robot balancín (ver Figura 3.29), encerrado en color azul están los 4 orificios para los espárragos, encerrado en color morado están los 2 orificios que se crearon para pasar los cables de los encoders y sensores infrarrojos seguidores de línea y encerrado en color naranja están los 2 orificios que se crearon para pasar los cables del Puente H- L298N.

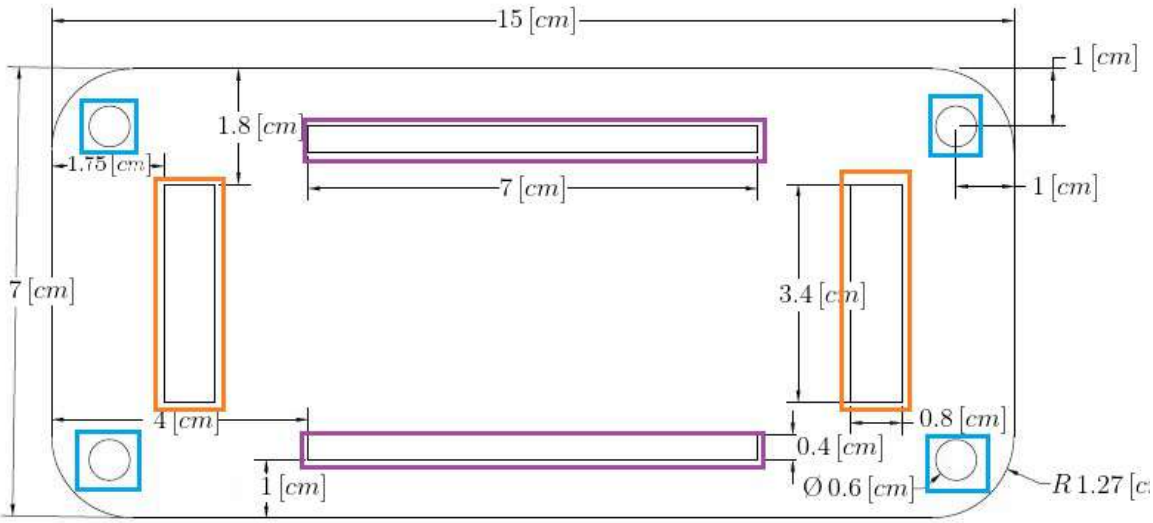

Figura 3.29: Diseño del segundo nivel del robot balancín en AutoCAD online.

4. Se diseñó el tercer nivel del robot balancín ver en (ver Figura 3.30), encerrado en color amarillo esta los 4 orificios que se crearon para atornillar la tarjeta PCB, encerrado en color naranja están los 2 orificios que se crearon para pasar los cables del Puente H-L298N y en color morado están los 2 orificios que se crearon para pasar los cables de los encoders y sensores infrarrojo seguidores de línea.

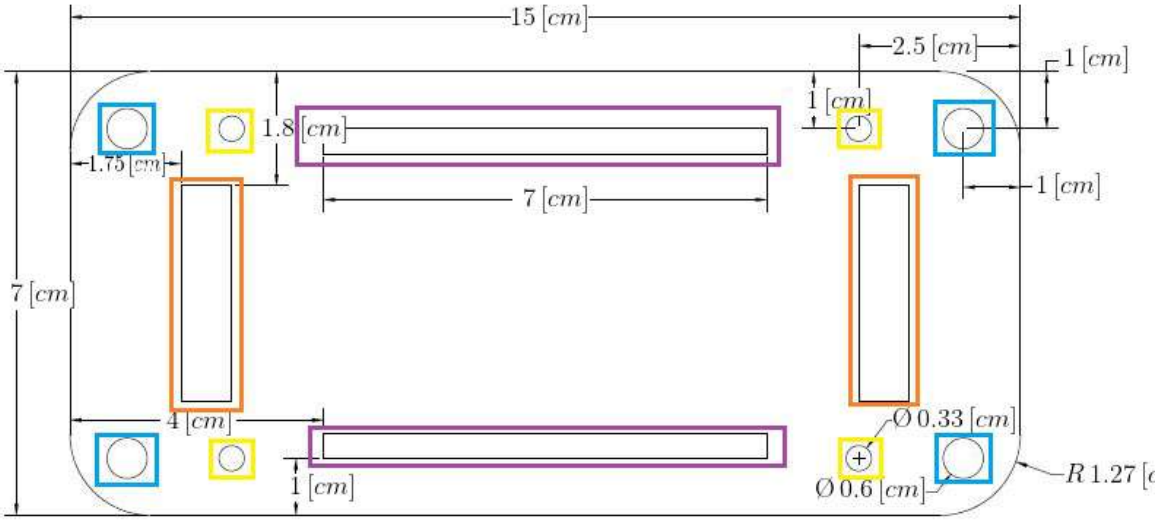

Figura 3.30: Diseño del Tercer nivel del robot balancín en AutoCAD online.

Como postes de la estructura se usaron 4 espárragos de acero (ver Figura 3.31), con un diámetro de $5\,[mm]$ y una longitud de $15\,[cm]$.

Figura 3.31: Espárragos de acero.

Para sujetar cada nivel en los espárragos se usaron 16 tuercas de acero, con diámetro interno de $5\,[mm]$ (ver Figura 3.32).

Figura 3.32: Tuercas de acero.

En cada motor se usaron acopladores (ver Figura 3.33) en los ejes para acoplar las llantas, las llantas que se utilizaron tienen un diámetro de $7\,[cm]$ con $2\,[cm]$ de ancho (ver Figura 3.34).

Figura 3.33: Acoplador de llantas.

Figura 3.34: Llantas robot balancín.

Se usaron sujetadores en cada motor para sujetarlos a la estructura (ver Figura 3.35).

Figura 3.35: Sujetador de motores POLOLU.

III.2.9 Diseño de PCB

La tarjeta PCB se diseñó siguiendo los siguientes pasos:

1. Se creó el circuito eléctrico del robot balancín (ver Figura 3.36).

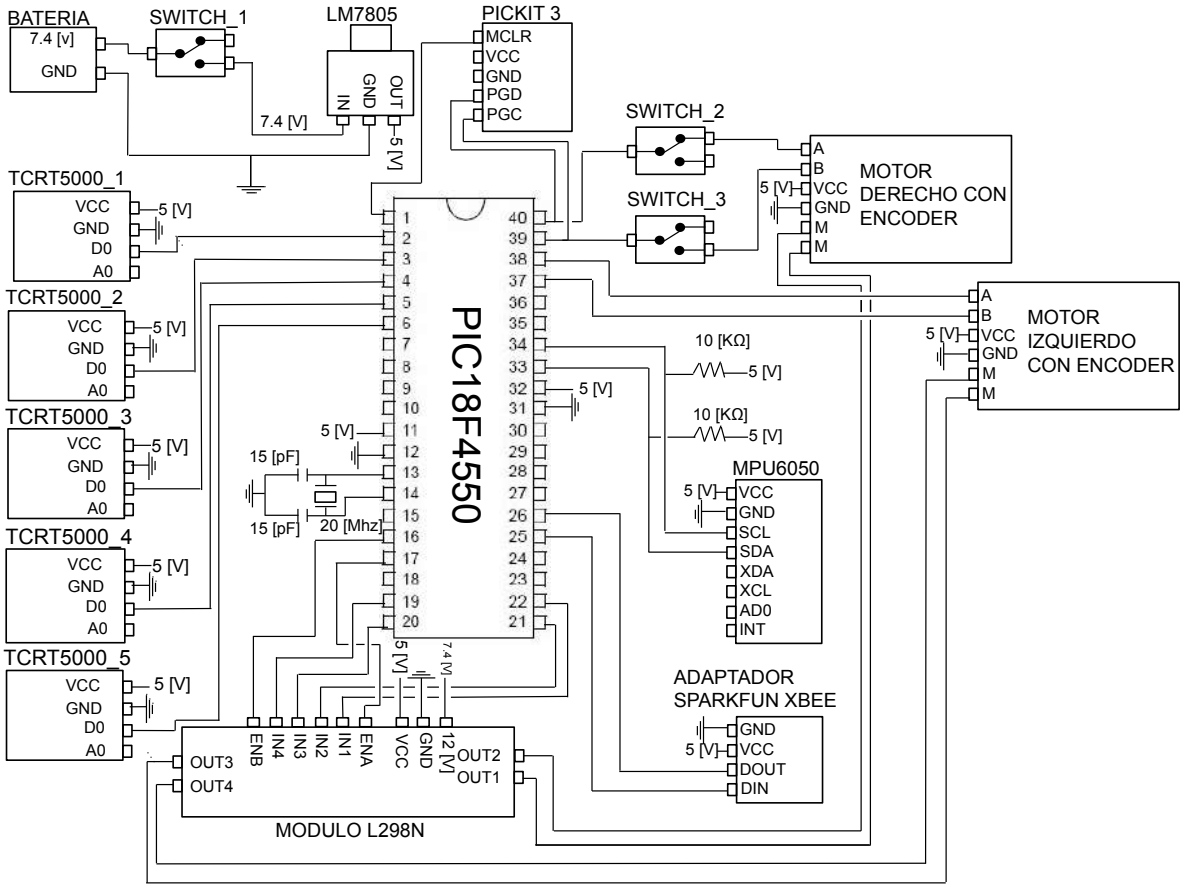

Figura 3.36: Diagrama eléctrico del robot balancín.

2. Siguiendo el circuito eléctrico de la Figura 3.36 se creó uno similar en el programa Easy EDA para hacer la PCB (ver Figura 3.37).

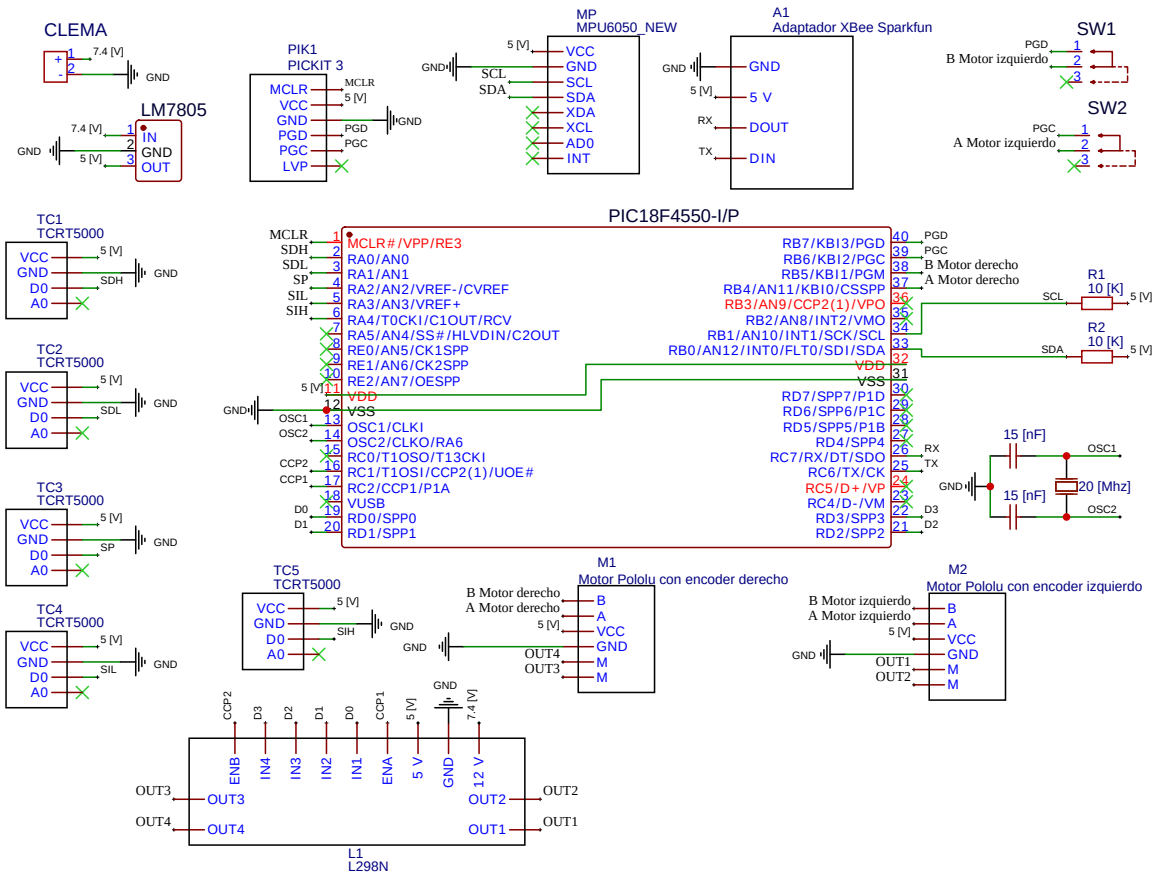

Figura 3.37: Diagrama eléctrico del robot balancín en EASY EDA.

3. Se acomodaron los componentes comunes en paralelo (ver Figura 3.38) en el programa Easy EDA.

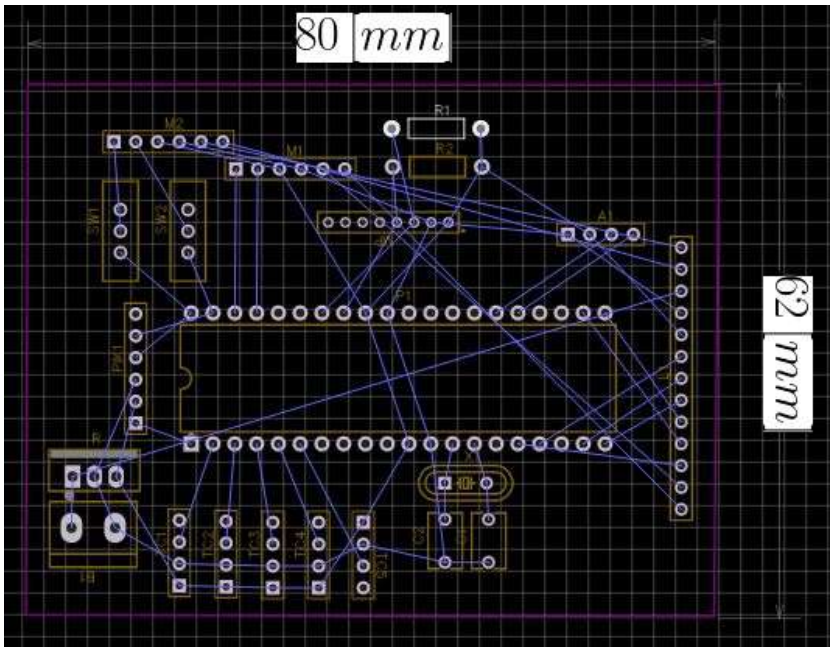

Figura 3.38: Componentes acomodados en Easy EDA.

4. Se enrutaron las conexiones (ver Figura 3.39), el ancho de las pistas de enrutado es de $0.8\,[mm]$.

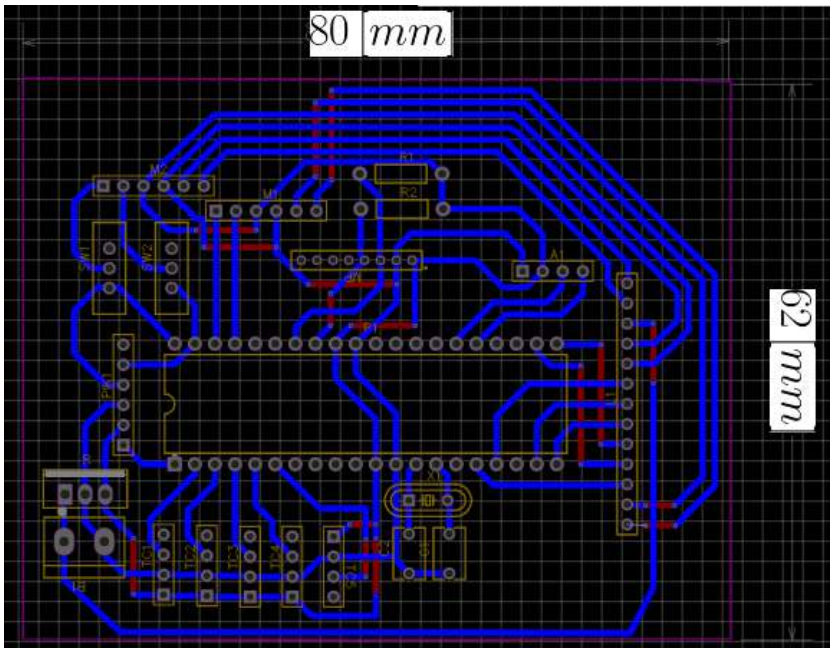

Figura 3.39: Componentes enrutados en EASY EDA.

5. Se agregó el plano a tierra (ver Figura 3.40) para tener un mejor retorno de corriente de cada componente, con espaciado de $0.25\,[mm]$ entre pistas. Este es el diseño final de la tarjeta PCB

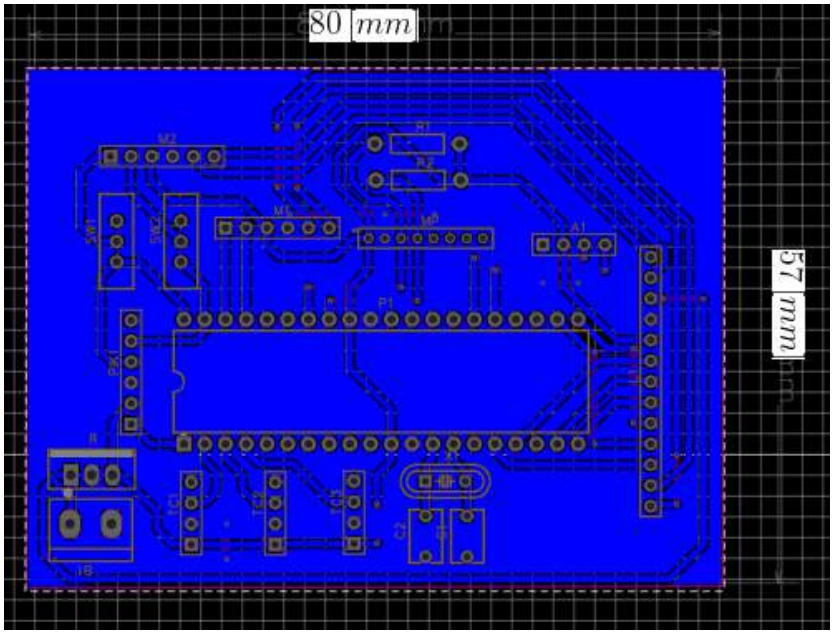

Figura 3.40: Plano a tierra de PCB en EASY EDA.

III.2.10 Pista de pruebas

Los pasos que se siguieron para el diseño de la pista se muestran a continuación:

1. Se creo una pendiente con un ángulo de 15 grados con una línea blanca de $0.003\,[m]$ en el centro, en Solidworks (ver Figura 3.41).

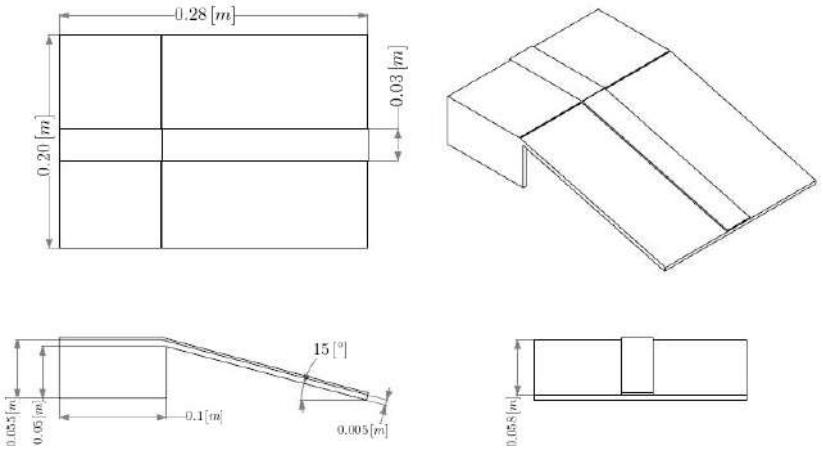

Figura 3.41: Rampa con línea en Solidworks.

2. Se diseñaron dos curvas con $0.15\,[m]$ de diámetro que conectan con la pendiente de 15 grados (ver Figura 3.42).

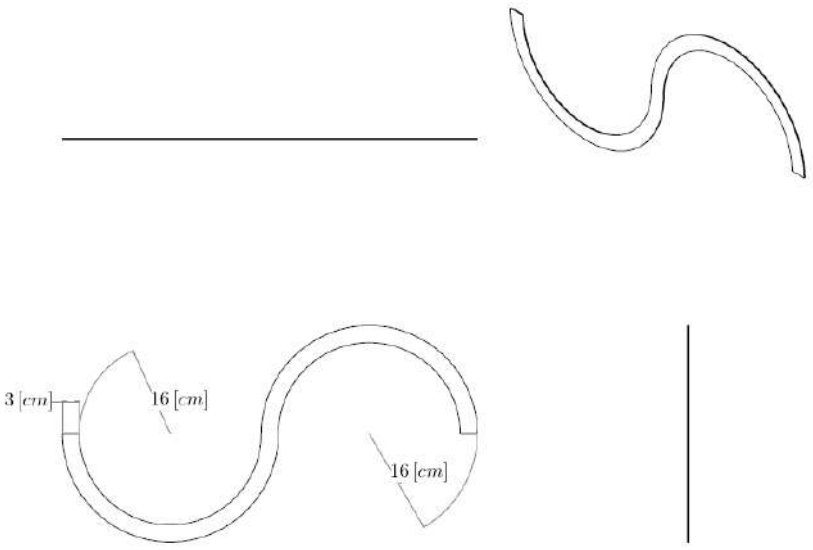

Figura 3.42: Curvas de pista de pruebas en Solidworks.

3. Se diseñó la base de la pista de pruebas con $1\,[m]$ de largo, $40\,[m]$ de ancho y $0.005\,[m]$ de espesor (ver Figura 3.43).

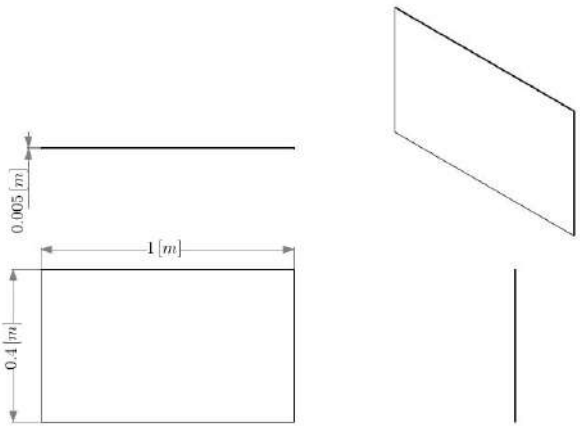

Figura 3.43: Base de pista de pruebas en Solidworks.

4. La pista de pruebas se puede ver en la Figura 3.44.

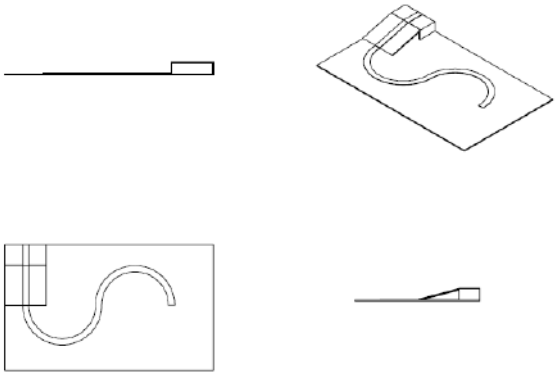

Figura 3.44: Pista de pruebas en Solidworks.

IV. RESULTADOS Y DISCUSIÓN

IV.1. Estructura robot balancín

Figura 4.1: Estructura del robot balancín sin componentes electrónicos.

En la Figura 4.1 se puede ver la estructura diseñada para el robot balancín, con tres niveles de acrílico y los espárragos que sujetan la estructura, el primer nivel se agregaron los orificios para colocar los motores, el Puente H-L298N y la estructura de los sensores, el segundo nivel tiene sólo los orificios para pasar los cables a la parte superior y en el tercer nivel se agregaron 4 orificios para atornillar la tarjeta PCB y los orificios para pasar los cables de los niveles inferiores.

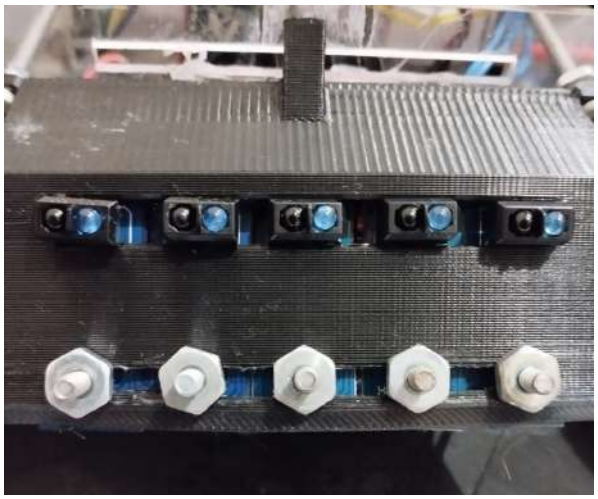

Figura 4.2: Sensores seguidores de línea colocados en la pieza impresa en material PLA color negro.

En la Figura 4.2 se puede ver la pieza impresa en material PLA con los sensores seguidores de línea y el sensor que detecta la pendiente, el sensor del centro es el que detecta la pendiente de 15 grados S_P, los de los lados son S_{DH}, S_{DL}, S_{IL} y S_{IH}, se puede ver que están atornillados a la pieza que se diseñó.

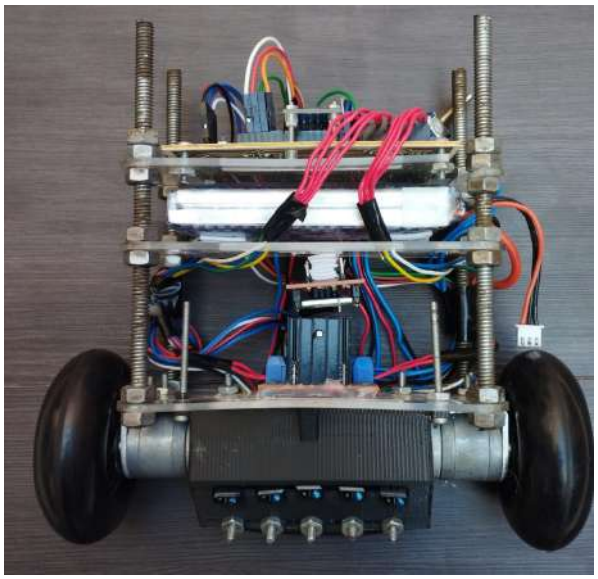

Figura 4.3: Robot balancín con todos los componentes.

En la Figura 4.3 se puede ver la estructura del robot balancín con todos los componentes integrados, en el primer nivel se encuentra el puente H-L298N con las conexiones

hacia la tarjeta PCB y los motores atornillados en la parte baja, en el segundo nivel se encuentra la batería de LI-PO colocada con cinta doble cara y abajo se encuentra el XBee PRO S1 colocado con cinta doble cara, en el tercer nivel se encuentra la tarjeta PCB atornillada a la estructura con los componentes electrónicos.

IV.2. PCB

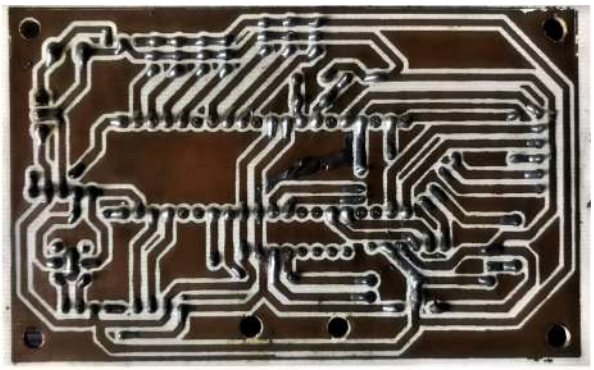

Figura 4.4: Vista inferior de PCB.

En la Figura 4.4 se muestra la vista inferior de la tarjeta PCB con componentes soldados, se pueden ver las cuatro perforaciones para sujetar la PCB a la estructura del robot balancín y las dos para sujetar el sensor inercial MPU6050.

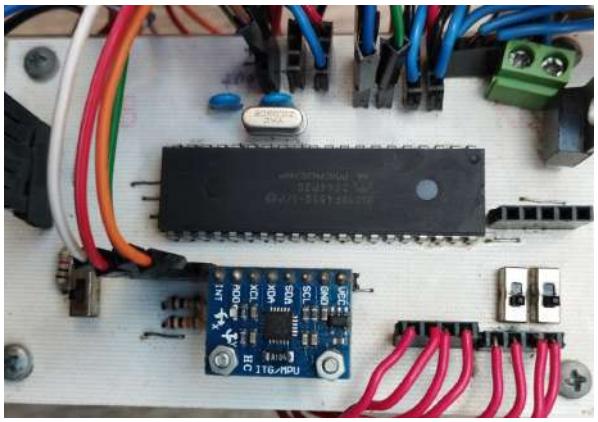

Figura 4.5: Vista superior de tarjeta PCB colocada en el robot balancín.

En la Figura 4.5 se puede ver la vista superior de la tarjeta PCB con las conexiones de cada componente, para conectar los encoders en la parte alta del puerto B se usó cable de

color rojo, para conectar el módulo Xbee PRO S1 se usaron dupones hembra-macho, para conectar la batería se usó una clema para atornillar las conexiones de la batería, para conectar los sensores se usó cable de protoboard y para conectar lo pines de control del puente H-L298N se usaron dupones hembra-macho.

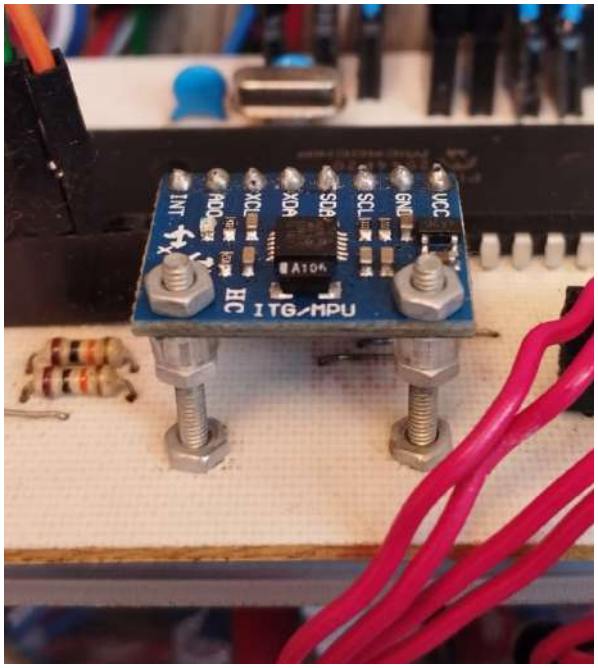

Figura 4.6: Posición de sensor inercial MPU6050 en PCB.

En la Figura 4.6 se muestra el sensor inercial MPU6050 colocado en la tarjeta PCB, el cual tiene dos tornillos que sirven para ajustar la inclinación y sujetar el sensor.

IV.3. Pista de pruebas

Figura 4.7: Pista de pruebas experimentales del robot balancín.

En la Figura 4.7 se puede apreciar la pista que se usó para que el robot balancín siguiera línea y tomara la pendiente de 15 grados, la base de la pista es de triplai de $10\,[mm]$ y la pendiente se hizo con MDF con un ángulo de inclinación de 15 grados, la línea mide $3\,[cm]$ de ancho con curvas de $16\,[cm]$ de radio.

IV.4. Respuesta en el tiempo del sistema embebido en el PIC18F4550

En la Figura 4.8 se muestra un recorrido completo de la pista de la Figura 4.7 por el robot balancín.

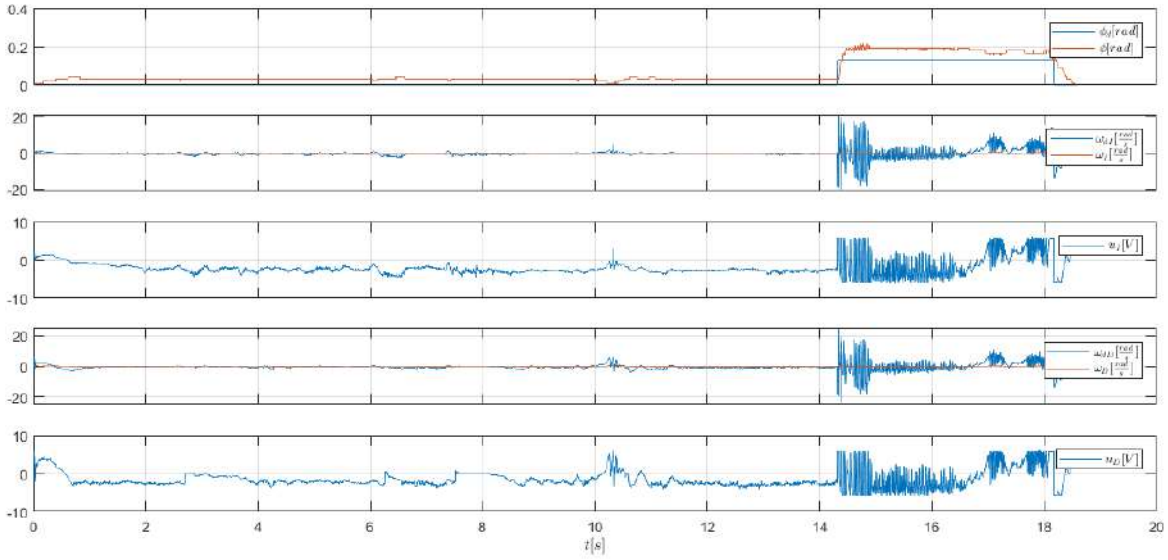

Figura 4.8: Respuesta en el tiempo del controlador PD-PI.

En la Figura 4.8 se muestra la respuesta en el tiempo del controlador PD-PI, en el primer canal se muestra en color rojo la posición del robot balancín ϕ y en color azul la posición deseada del robot balancín ϕ_d, en el segundo canal en color rojo se muestra la velocidad de la rueda izquierda ω_I y en color azul la velocidad deseada de la rueda izquierda ω_{dI}, en el tercer canal en color azul se muestra el esfuerzo de control de la rueda izquierda u_I, en el cuarto canal en color rojo se muestra la velocidad de la rueda derecha ω_D y en color azul se muestra la velocidad deseada de la rueda derecha ω_{dD}, en el quinto canal en color azul se muestra el esfuerzo de control de la rueda derecha u_D.

En todas las gráficas con dos señales se aprecia el seguimiento al valor de consigna, sin embargo, cuando el robot detecta la pendiente en $t = 14.32\,[s]$, se cambia el valor de ϕ_d. Cuando el robot sube el plano inclinado las señales deseadas de velocidad a los motores y los voltajes aplicados a los motores presentan ruido, sin embargo los motores y el propio robot balancín filtraron este ruido y el robot balancín logró subir el plano inclinado sin caerse y observando solo un error en estado estacionario entre ϕ y ϕ_d en esta última.

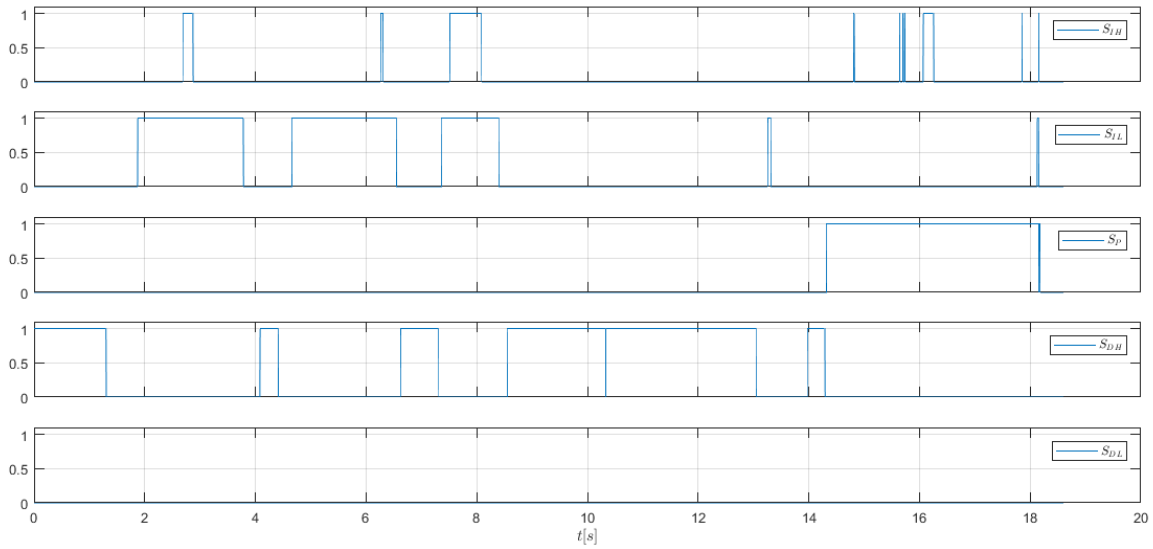

Figura 4.9: Respuesta en el tiempo de los sensores seguidores de línea y sensor que detecta la pendiente.

En la Figura 4.9 se muestra la respuesta en el tiempo de los sensores seguidores de línea y el sensor que detecta la pendiente, en el primer canal se muestra la respuesta de S_{IH} que es el sensor izquierdo más alejado de la línea blanca, en el segundo canal se muestra la respuesta de S_{IL} que es el sensor izquierdo más cercano a la línea blanca, en el tercer canal se muestra la respuesta de S_P el cual detecta la pendiente, en el cuarto canal se muestra la respuesta de S_{DL} que es el sensor derecho más cercano a la línea blanca y en el quinto canal se muestra la respuesta de S_{DH} que es el sensor derecho más alejado de la línea blanca.

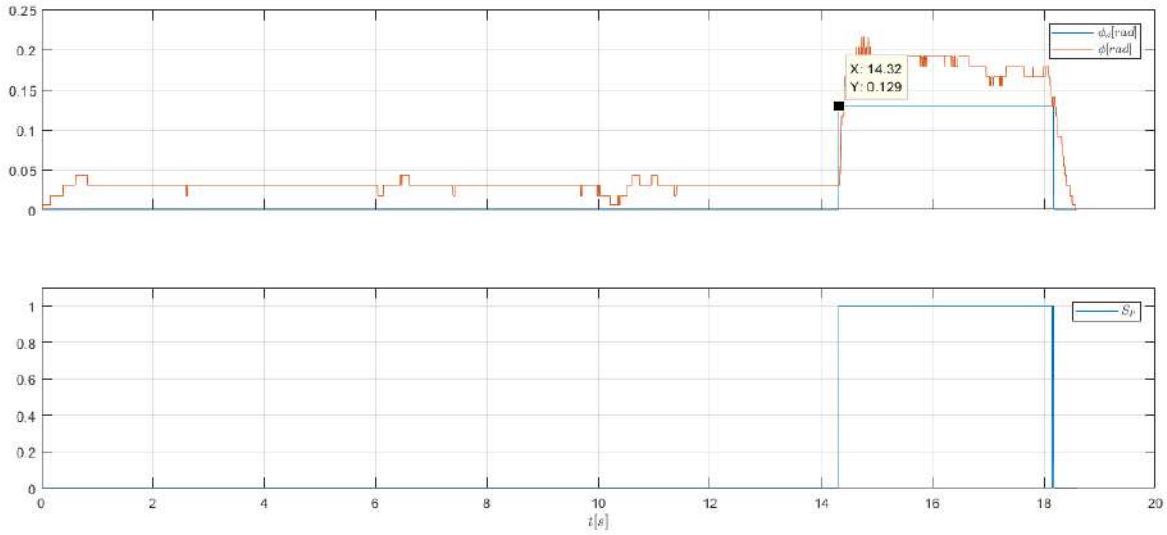

Figura 4.10: Respuesta en el tiempo de ϕ, ϕ_d y S_P.

En la Figura 4.10 se muestra con más detalle el cambio de posición cuando el robot balancín detecta la pendiente, podemos ver que ϕ_d es cero hasta que en el segundo 14.32 se hace el cambio a $0.129\,[rad]$ que es el momento en que S_P hace el cambio de 0 a 1, esto significa que ha detectado la pendiente, se puede observar que existe un desface de ϕ_d con respecto a ϕ a lo largo del tiempo y esto es por la suma del avance ω_A a la salida del controlador PD maestro.
Durante el plano horizontal ω_A es 8.0, ϕ_d es 0.0 y cuando S_P es 1 cambia ω_A a 15.0, ϕ_d a 0.13, las constantes que se obtuvieron empíricamente para el controlador maestro PD son $K_P = 250$, $K_D = 7.3$ y para los lazos esclavos de velocidad PI son $K_{PD} = 1.0$, $K_{ID} = 1.0$, $K_{PI} = 1.0$ y $K_{DI} = 1.0$, estas ganancias se usaron para el plano horizontal e inclinado.

Cabe resaltar que estas gráficas se pudieron realizar gracias a que el programa embebido en el microcontrolador manda un $byte$ al puerto serial por cada variable a graficar, donde esta información llega a la laptop gracias a al puente de comunicación inalámbrica XBee evitando la perturbación de un alambre de comunicación pegado al robot balancín durante el recorrido experimental de la pista.

Con lo anterior se demuestra que la estrategia propuesta embebida en un PIC18F4550

permite al robot balancín balancearse, avanzar siguiendo línea de la pista y subir el plano inclinado del final del trayecto.

V. CONCLUSIONES

En este trabajo se pueden concluir los siguientes puntos:

El primero es la estructura, se diseñó en base a los componentes que tiene el robot balancín y fue funcional para el trabajo, desde el primer nivel en el cual se colocó el Puente H-L298N, el segundo nivel donde se colocó la batería de LI-PO y el tercer nivel donde se colocó la PCB con los componentes electrónicos. Pero a pesar de que se cumplió el objetivo hay dos puntos que se puede mejorar, el primero es el tipo de material, ya que durante la experimentación el robot balancín sufría caídas debido a malas configuraciones lo que ocasionó que el acrílico se estrellara, sería una buena opción crear los niveles con otro material para para este tipo de prototipos experimentales.

Otro punto de la estructura que se puede mejorar es el peso del robot balancín, ya que los espárragos son más pesados que las placas de acrílico y al tener mucho peso los motores requieren un mayor suministro de corriente para poder responder ante las perturbaciones y esto ocasiona el desgaste de la batería.

El segundo punto del cual se puede concluir es acerca de seguir la línea blanca, se logró que el robot balancín siguiera línea y los sensores elegidos funcionaron adecuadamente, pero es importante mencionar que previamente se experimentó sólo con dos sensores y el robot balancín se salia de la línea ya que por inercia las llantas siguen avanzando con $40\,[\,\%]$ de velocidad deseada y se agregaron otros dos para reducir la velocidad deseada a $20\,[\,\%]$. Previamente se experimentó con otro modelo de sensor infrarrojo el cual tenía los diodos emisor receptor demasiado expuestos a la luz del día y se veían afectados, los TCR-5000 funcionaron muy bien, ya que se puede variar la distancia a la cual detectan la línea y se ven menos afectados por la luz solar.

El tercer punto a concluir es la PCB, el diseñó de la PCB permitió que hubiera menos vibraciones en los pines de los componentes, ya que al iniciar las experimentaciones se tenía el circuito en una protoboard lo que ocasionaba que los componentes se desconectaran con el movimiento del robot balancín, ya que a veces se movían los pines del sensor inercial MPU6050 y el PIC18F4550 se quedaba esperando los $bits$ de respuesta de la comunicación i2c y ocasionaba que el robot balancín perdiera el equilibrio.
Otro punto importante con respecto a la PCB es que al realizar las conexiones de los encoders a la parte alta del puerto B ocasionaba interferencia con los pines PGD y PGC del Pickit 3 y no permitía la programación del PIC18F4550, por esta razón se agregaron dos switch en estos dos pines lo que facilitó en gran medida la programación en cada prueba y evitó estar desconectando y conectando los encoders.

El cuarto punto que se puede concluir es acerca del filtro complementario que se usó para obtener el ángulo de inclinación del robot balancín, ya que originalmente el filtro complementario utiliza los tres ejes coordenados del sensor inercial MPU6050 y se usa un arcotangente para obtener el ángulo de inclinación pero en este caso sólo se buscaba controlar la posición en el eje x, por esta razón sólo se usó la aceleración en el eje x (a_x) y la velocidad angular en el eje y ($\bar{w}_y$). Esto redujo la carga en la comunicación i2c y también en las operaciones matemáticas ya que con esto se evitó hacer el arcotangente, si bien la respuesta es buena y adecuada para los requerimientos de este trabajo es un filtro que se ve muy afectado por el periodo de muestreo, ya que se estuvieron haciendo pruebas con un periodo de muestreo de $20\,[ms]$ y el robot balancín no respondía adecuadamente, después de estar experimentando se logró que se mantuviera en equilibrio pero no era capaz de seguir línea, por esta razón se bajo el periodo de muestreo a $5\,[ms]$ y en este periodo de muestreo se obtuvo una mejor respuesta y el PIC18F4550 logró realizar las otras tareas adecuadamente.

El quinto punto es la pista de pruebas, ya que se diseñó en base al objetivo pero durante la fase de experimentación se observó que el MDF de $5\,[mm]$ no estaba completamente plano lo que provocaba que el robot balancín no pudiera avanzar adecuadamente durante todo el trayecto, por esta razón la base de la pista de pruebas se cambió por triplay de $10\,[mm]$ con

lo que se vio una mejor respuesta en el avance del robot balancín.

El sexto punto es el PIC18F4550, conocer las capacidades que tiene este PIC es muy importante ya que se logró probar la hipótesis y cumplir el objetivo, uno de los punto clave para hacer el control fue colocar el controlador PD-PI en la interrupción del timer 0 para que no se afectara por la lectura de los encoders que se realizó en la parte alta del puerto B, ya que la prioridad del timer 0 es mayor a la del puerto B, otro de los puntos clave fue la comunicación i2c con el sensor MPU6050 ya que inicialmente se estaba haciendo por interrupción lo que ocasionaba que se realizará una lectura erronea de los encoders y para corregirlo se colocó en el main del programa.
Es importante mencionar que previamente se intentó hacer el control del robot balancín en un PIC16F877A y se cargó el mismo código que en el PIC18F4550 pero la memoria ROM no fue suficiente para realizar las operaciones matemáticas y adquirir los datos de los sensores, ya que aparecía un error al intentar contener las mismas operaciones en la interrupción, ademas se intentó distribuir las operaciones en diferentes funciones para reducir la carga en la interrupción pero fue en vano, el problema radicaba principalmente al hacer el control de los lazos esclavos ya que era cuando marcaba el error de exceso de memoria ROM.

El séptimo punto son los resultados experimentales, el controlador PD-PI funcionó adecuadamente y la respuesta en el tiempo es estable, aunque al realizar la experimentación y ver los resultados se debe mejorar la sintonía de los controladores, especialmente del controlador maestro ya que las señales de velocidad deseada y los voltajes aplicados a los motores presentan ruido. Debido a que se terminó el tiempo destinado a este trabajo, se deja esta mejora como un trabajo futuro.

Por último es importante mencionar que se intentó realizar el control de manera remota, para estas pruebas inicialmente se probó que se realizará el control correctamente con un TTL por medio de cables y después con los módulos XBee PRO S1 y XBee PRO S1 PRO pero no se logró realizar la comunicación bidirecional en el periodo requerido para controlar el Robot balancín, para esto se hicieron pruebas a $20\,[ms]$ y se hacia el envió pero llegaban datos erró-

neos lo que ocasionaba que el robot Balancín perdiera el equilibrio, también se probó reducir la velocidad de transmisión de datos a 9600 baudios para descartar que fuera por la velocidad aunque se obtuvo el mismo resultado. En este punto es importante mencionar que también se hicieron pruebas con módulos Bluethooth hc-05, hc-06 y hc-08 v4.0 para realizar el control remoto y al igual que los módulos Xbee PRO S1 llegaban datos erróneos. Para trabajos futuros sería bueno buscar nuevas tecnologías que puedan realizar la transmisión bidireccional de datos en periodos por debajo de los $10\,[ms]$ y sin errores para poder controlar el robot balancín.

BIBLIOGRAFÍA

Adeel, Umar, KS Alimgeer, Omair Inam, Ayesha Hameed, Mehmood Qureshi and Mehmood Ashraf. 2013. Autonomous dual wheel self balancing robot based on microcontroller. Journal of Basic and Applied Scientific Research 3(1):843–848.

Anisimov, DN, Thai Son Dang et al. 2018. Development of a microcontroller-based adaptive fuzzy controller for a two-wheeled self-balancing robot. Microsystem Technologies 24(9):3677–3687.

Azimi, Mohammad Mahdi and Hamid Reza Koofigar. 2013. Model predictive control for a two wheeled self balancing robot. 2013 First RSI/ISM International Conference on Robotics and Mechatronics (ICRoM). :152–157.

Baturone, Aníbal Ollero. 2005. Robótica: manipuladores y robots móviles. Marcombo.

Bertolini, Andrea. 2016. Insurance and risk management for robotic devices: Identifying the problems. Global Jurist 16(3):235.

Bhatti, Omer Saleem, Khalid Mehmood-ul Hasan and Muhammad Anas Imtiaz. 2015. Attitude control and stabilization of a two-wheeled self-balancing robot. Journal of Control Engineering and Applied Informatics 17(3):98–104.

Binugroho, Eko Henfri, Derry Pratama, Akhmad Zackarya Rizqy Syahputra and Dadet Pramadihanto. 2015. Control for balancing line follower robot using discrete cascaded PID algorithm on ADROIT V1 education robot. 2015 International Electronics Symposium (IES). :245–250.

Čapek, Karel. 1920. RUR (Rossums Universal Robots): A Play in Introductory Scene and Three Acts.

Chee, Ong Yin et al. 2006. Design and development of two wheeled autonomous balancing robot. 2006 4th Student Conference on Research and Development. IEEE :169–172.

Clark, MA, JB Field, SG McMahon, PS Philps and BS Cazzolatto. 2005. EDGAR: A self balancing scooter. Project Final Report. University of Adelaide, Australia .

Ding, Ye, Joshua Gafford and Mie Kunio. 2012. Modeling, simulation and fabrication of a balancing robot. Harvard University, Massachusettes Institute of Technology 151.

Dorf, C. and H. Bishop. 2011. Modern Control Systems. 12 ed. Prentice Hall.

Draz, Muhammad Umar, Mohsin Shaheer Ali, Maryam Majeed, Umair Ejaz and Umer Izhar. 2012. Segway electric vehicle. 2012 International Conference of Robotics and Artificial Intelligence. IEEE :34–39.

Fang, Jian. 2014. The LQR controller design of two-wheeled self-balancing robot based on the particle swarm optimization algorithm. Mathematical Problems in Engineering 2014.

Fantoni, Isabelle, Rogelio Lozano and Rogelio Lozano. 2002. Non-linear control for underactuated mechanical systems. Springer Science & Business Media.

Feng, Tao, Tao Liu, Xu Wang, Zhao Xu, Meng Zhang and Sheng-chao Han. 2011. Modeling and implementation of two-wheel self-balancing robot equipped with supporting arms. 2011 6th IEEE Conference on Industrial Electronics and Applications. :713–718.

García, R and Manuel Arias Montiel. 2014. Desarrollo de un robot tipo ballbot para aplicaciones de control. Huajuapan de León, México .

Ghani, Nor Maniha Abdul, Faradila Naim and Tan Piow Yon. 2011. Two wheels balancing robot with line following capability. World Academy of Science, Engineering and Technology 55(7):1401–1405.

Gong, Yulei, Xiao Wu and Huijiao Ma. 2015. Research on Control Strategy of Two-Wheeled Self-Balancing Robot. 2015 International Conference on Computer Science and Mechanical Automation (CSMA). :281–284.

Gonzalez, C, I Alvarado and D Muñoz La Peña. 2017. Low cost two-wheels self-balancing robot for control education. IFAC-PapersOnLine 50(1):9174–9179.

Hasanah, Nurul, Ali Husein Alasiry and Bambang Sumantri. 2018. Two Wheels Line Following Balancing Robot Control using Fuzzy Logic and PID on Sloping Surface. 2018 International Electronics Symposium on Engineering Technology and Applications (IESETA). :210–215.

Hsu, Chun-Fei, Chien-Ting Su, Wei-Fu Kao and Bore-Kuen Lee. 2018. Vision-Based Line-Following Control of a Two-Wheel Self-Balancing Robot. 2018 International Conference on Machine Learning and Cybernetics (ICMLC). 1 :319–324.

Ivoilov, Andrey, Vadim Zhmud, Vitaly Trubin and Lubomir Dimitrov. 2016. Detection of unrevealed non-linearities in the layout of the balancing robot. 2016 International Siberian Conference on Control and Communications (SIBCON). IEEE :1–9.

Iwendi, Celestine, Mohammed A Alqarni, Joseph Henry Anajemba, Ahmed S Alfakeeh, Zhiyong Zhang and Ali Kashif Bashir. 2019. Robust navigational control of a two-wheeled self-balancing robot in a sensed environment. IEEE Access 7:82337–82348.

Jamil, Osama, Mohsin Jamil, Yasar Ayaz and Khubab Ahmad. 2014. Modeling, control of a two-wheeled self-balancing robot. 2014 International Conference on Robotics and Emerging Allied Technologies in Engineering (iCREATE). IEEE :191–199.

Juang, Hau-Shiue and Kai-Yew Lum. 2013. Design and control of a two-wheel self-balancing robot using the arduino microcontroller board. 2013 10th IEEE International Conference on Control and Automation (ICCA). :634–639.

Jung, Taehwa, Hyun Wook Kim and Seul Jung. 2011. Implementation and control of balancing line tracer robot using vision. 2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI). :858–862.

Li, Zhijun, Chenguang Yang and Liping Fan. 2012. Advanced control of wheeled inverted pendulum systems. Springer Science & Business Media.

Mahler, Bernhard and Jan Haase. 2013. Mathematical model and control strategy of a two-wheeled self-balancing robot. IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society. :4198–4203.

Mai, The Anh, Thai Son Dang, D.N. Anisimov and Ekaterina Fedorova. 2019. Fuzzy-PID Controller for Two Wheels Balancing Robot Based on STM32 Microcontroller. 2019 International Conference on Engineering Technologies and Computer Science (EnT). :20–24.

Martínez, Sergio Becerril and Sergio Becerril Martínez. 2010. Propuesta de sistema de estacionamiento autónomo aplicado a un vehículo eléctrico auto-balanceado tipo péndulo invertido.

Martins, Ricardo Santos and Francisco Nunes. 2017. Control system for a self-balancing robot. 2017 4th Experiment@International Conference (exp.at'17). :297–302.

Miasa, Samer, Mohammad Al-Mjali, Anas Al-Haj Ibrahim and Tarek A. Tutunji. 2010. Fuzzy control of a two-wheel balancing robot using DSPIC. 2010 7th International Multi- Conference on Systems, Signals and Devices. :1–6.

Min, Hyung-Gi, Ji-Hoon Kim, Ju-Han Yoon, Eun-Tae Jeung and Sung-Ha Kwon. 2010. A control of balancing robot. Journal of Institute of Control, Robotics and Systems 16(12):1201–1207.

Nasir, Ahmad Nor Kasruddin, Mohd. Ashraf Ahmad, Riduwan Ghazali and Nasrul Salim Pakheri. 2011. Performance Comparison between Fuzzy Logic Controller (FLC) and PID Controller for a Highly Nonlinear Two-Wheels Balancing Robot. 2011 First International Conference on Informatics and Computational Intelligence. :176–181.

Nozaki, Kohei and Toshiyuki Murakami. 2009. A motion control of two-wheels driven mobile manipulator for human-robot cooperative transportation. 2009 35th Annual Conference of IEEE Industrial Electronics. IEEE :1574–1579.

Ogata, Katsuhiko and Guillermo Lopez Portillo Sanchez. 1987. Dinámica de sistemas. Ed. Prentice Hall Hispanoamericana.

Ordóñez Cerezo, Juan, Encarnación Castillo Morales and José María Cañas Plaza. 2019. Control system in open-source FPGA for a self-balancing robot. Electronics 8(2):198.

Palacios-Hurtado, Manuel. 2017. Control de sistema de realidad aumentada mediante gestos de los dedos. Master's thesis Universidad Politécnica de Valencia.

Park, Ji-Hyun and Baek-Kyu Cho. 2018. Development of a self-balancing robot with a control moment gyroscope. International Journal of Advanced Robotic Systems 15(2):1729881418770865.

Patete, Anna, Iãki Aguirre and Sánchez Héctor. 2011. Control de un péndulo invertido basado en un modelo reducido. INGENIERÍA UC 18(1):12–22.

Philippart, Vincent Y., Kristian O. Snel, Antoine M. de Waal, Jeedella S. Y. Jeedella and Esmaeil Najafi. 2019. Model-Based Design for a Self-Balancing Robot using the Arduino Micro-Controller Board. 2019 23rd International Conference on Mechatronics Technology (ICMT). :1–6.

Pulido Fentanes, Jaime et al. 2012. Exploración y reconstrucción tridimensional de entornos mediante robots móviles.

Qiu, Congying and Yibin Huang. 2015. The design of fuzzy adaptive PID controller of two-wheeled self-balancing robot. International Journal of Information and Electronics Engineering 5(3):193.

Richards, Neil M and William D Smart. 2016. How should the law think about robots? Robot law. Edward Elgar Publishing.

Ruan, Xiaogang, Jianxian Cai and Jing Chen. 2008. Learning to Control Two-Wheeled Self-Balancing Robot Using Reinforcement Learning Rules and Fuzzy Neural Networks. 2008 Fourth International Conference on Natural Computation. 4 :395–398.

Ruan, Xiaogang and Wangbo Li. 2014. Ultrasonic sensor based two-wheeled self-balancing robot obstacle avoidance control system. 2014 IEEE International Conference on Mechatronics and Automation. :896–900.

Sarathy, Shyamala, M M Mariyam Hibah, S Anusooya and S. Kalaivani. 2018. Implementation of Efficient Self Balancing Robot. 2018 International Conference on Recent Trends in Electrical, Control and Communication (RTECC). :65–70.

Schinstock, Dale, Kyle McGahee and Shane Smith. 2016. Engaging students in control systems using a balancing robot in a mechatronics course. 2016 American Control Conference (ACC). :6658–6663.

Segway. N.d. Segway is the worldwide leader in personal transportation.
URL: *https://www.segway.com/*

Sun, Chenxi, Tao Lu and Kui Yuan. 2013. Balance control of two-wheeled self-balancing robot based on Linear Quadratic Regulator and Neural Network. 2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP). IEEE :862–867.

Sun, Fengxin, Zhen Yu and Haijiao Yang. 2014. A design for two-wheeled self-balancing robot based on Kalman filter and LQR. 2014 International Conference on Mechatronics and Control (ICMC). :612–616.

Sundin, Christian and Filip Thorstenson. 2012. Autonomous balancing robot: Design and construction of a balancing robot. Master's thesis Chalmers University Of Technology.

Wasif, Ammar, Danish Raza, Waqas Rasheed, Zubair Farooq and Syed Qaseem Ali. 2013. Design and implementation of a two wheel self balancing robot with a two level adaptive control. Eighth International Conference on Digital Information Management (ICDIM 2013). :187–193.

Wu, Junfeng and Shengwei Jia. 2011. T-S adaptive neural network fuzzy control applied in two-wheeled self-balancing robot. Proceedings of 2011 6th International Forum on Strategic Technology. 2 :1023–1026.

Wu, Junfeng, Wanying Zhang and Shengda Wang. 2012. A two-wheeled self-balancing robot with the fuzzy PD control method. Mathematical Problems in Engineering 2012.

Xiaogang, Ruan, Liu Jiang, Di Haijiang and Li Xinyuan. 2008. Design and LQ Control of a two-wheeled self-balancing robot. 2008 27th Chinese Control Conference. :275–279.

A. PROGRAMA EN PIC CCS DEL ROBOT BALANCÍN

A continuación se muestra el código en PIC CCS del robot balancín embebido en el PIC18F4550.

```
#include  <18F4550.h>//Librería del PIC18F4550
#include  <math.h>//Librería para funciones matemáticas
#fuses    HSPLL,NOPROTECT,NOWDT,PLL5,CPUDIV1,NOMCLR//Configuraciones generales
#use      delay(clock=48M)//Clock a 48 [Mhz] con PLL y crystal externo de 20 [Mhz]
#use      rs232(baud=115200,XMIT=PIN_C6,RCV=PIN_C7,BITS=8,PARITY=N)//Configuración de puerto RS232
#use      i2c(master, sda=PIN_B0, scl=PIN_B1,fast=100000)//Configuración de puerto i2c
//*****Direciones de puertos PIC18F4550*****//
#Byte PORTA = 0xF80            // Dirección del puerto A
#Byte PORTB = 0xF81            // Dirección del puerto B
#Byte PORTC = 0xF82            // Dirección del puerto C
#Byte PORTD = 0xF83            // Dirección del puerto D
#Byte PORTE = 0xF84            // Dirección del puerto E
//*****Bits de sensores*****//
#bit PA0 = 0xF80.0             // Dirección sensor SDH
#bit PA1 = 0xF80.1             // Dirección sensor SDL
#bit PA2 = 0xF80.2             // Dirección sensor SP
#bit PA3 = 0xF80.3             // Dirección sensor SIL
#bit PA4 = 0xF80.4             // Dirección sensor SIH
#bit PA5 = 0xF80.5             // No conectado
#bit PA6 = 0xF80.6             // No conectado
#bit PA7 = 0xF80.7             // No conectado
//*****Bits de control puente H L298N*****//
#bit PD0 = 0xF83.0             // Dirección motor izquierdo
#bit PD1 = 0xF83.1             // Dirección motor izquierdo
#bit PD2 = 0xF83.2             // Dirección motor derecho
#bit PD3 = 0xF83.3             // Dirección motor derecho
//*****Direcciones de memoría MPU6050*****//
#define   W_DATA         0xD0 //Dirección para escribir
#define   R_DATA         0xD1 //Dirección para leer
#define   AX_H           0x3B //Dirección ax parte alta
#define   AX_L           0x3C //Dirección ax parte baja
#define   GY_H           0x45 //Dirección gy parte alta
#define   GY_L           0x46 //Dirección gy parte baja
//*****Prototipos de funciones*****//
```

```
void Linea(); //Función sensores seguidores de línea
void Rampa(); //Función sensor rampa
void mpu6050_write(int add, int data); //Función para escribir en las direcciones de memoria MPU6050
void mpu6050_init(); //Función para inicializar el sensor inercial MPU6050
void mpu6050_datos(); //Función para obtener la aceleración y la velocidad angular
//*****Constantes controladores*****//
#define Ts   0.005 //Constante periodo de muestreo
#define iTs  1/Ts //Inversa del periodo de muestreo
#define VM  6.0   //Voltaje máximo motores
#define VN 5.7    //Voltaje nominal máximo motores
#define escENCODER 2*pi/(4*1632.0)//Cuentas a posición 2*Pi/4*PPR
#define escPWM 255.0/6.0 //Escalamiento de esfuerzo de control a PWM
#define M1 255.0/180.0 //Escalamiento de 0 a 180 grados en un Byte
#define M2 255.0/50.0 //Escalamiento de OmegaD a un Byte
#define M3 255.0/12.0 //Escalamiento del esfuerzo de control a un Byte
//*****Variables de filtro complementario*****//
signed int16 aX=0,gY=0;//Aceleración y girosocpio para el Filtro complementario
char aDATA[2],gDATA[2];//Obtener las lecturas del sensor MPU6050
float angulo=0.0,angulo_1=0.0,acel=0.0;//Variables del filtro complementario
//*****Variables lectura de encoders*****//
int8 posI=0,posD=0,AB,AB_1,BC,BC_1,aux,puerto;
//******Variables controlador PD maestro*****//
float phi=0.0,phid=0.00,phi_1=0.0;
float p=0.0,d=0.0,e=0.0,omega_d=0.0,omegaA=0.0;
float KP=250.0,KD=7.3;
//******Variables controlador motor derecho*****//
signed int posDER=0;
float posDER_1=0.0,e_posDER=0.0,posDERF=0.0;
float omegadD=0.0,omegaD=0.0,eD=0.0,pD=0.0,iD=0.0,uD=0.0,KVD=1.0;
float KPD=1.0,KID=5.0;
unsigned int PWMD=0,PWMFD=0;
//******Variables controlador motor izquierdo*****//
signed int posIZQ=0;
float posIZQ_1=0.0,e_posIZQ=0.0,posIZQF=0.0;
float omegadI=0.0,omegaI=0.0,eI=0.0,p_I=0.0,iI=0.0,uI=0.0,KVI=1.0;
float KPI=1.0,KII=5.0;
unsigned int PWMFI=0,PWMI=0;
//*****Variables sensores seguidores de línea*****//
int8 SDL=0,SIL=0,SDH=0,SIH=0;
//*****Variable sensor detecta pendiente*****//
int8 SP=0;
//*****Variables que filtran el sensor de pendiente
int8 SP_TRUE=0;
char SPN[14];
//*****Bandera de espera por 5 [s] antes de iniciar*****//
```

```
int flag=0;
//*****Datos para gráficar en matlab*****//
int phi_G;
int phid_G;
int omegadD_G;
int omegaD_G;
int uD_G;
int omegadI_G;
int omegaI_G;
int uI_G;
int sensores;
//*****Timer para periodo de muestreo a 5 [ms]*****//
#int_timer0
void timer0_isr()
{
   //*****Escalamiento de datos para filtro complementario*****//
   acel=-(aX*90.0)/16902.0;//Escalamiento de -90 a +90
   gY=gY/131.0;//Escalamiento de la velocidad angular
   //*****Filtro complementario*****//
   angulo = 0.999*(angulo_1+gY*(0.005)) + (0.001*acel);//filtro complementario
   phi_G=(angulo+90.0)*M1;///////////////Enviar para graficar////////////////
   angulo_1=angulo;
   if(flag==1)//Después de 5 [s] empieza con el control
   {
      Linea();//Función de mapeo de sensores seguidores de línea
      Rampa();//Función de mapeo de sensor que detecta la pendiente
      if(SP_TRUE==1)//Cuando detecta la pendiente
      {
         phid=0.13;
         KP=250.0;
         KD=7.3;
         KPD=1.0;
         KID=5.0;
         KPI=1.0;
         KII=5.0;
         if (e>0.0 && e<0.01)
         {
            omegaA=15;
         }
      }
      else//Cuando esta en el plano recto
      {
         phid=0.00;
         KP=250.0;
         KD=7.3;
```

```
   KPD=1.0;
   KPI=1.0;
   KID=5.0;
   KII=5.0;
   if (e>0.0 && e<0.01)
   {
       omegaA=8;
   }
}
phid_G=(((phid*180.0)/3.1415)+90.0)*M1;///////////////Enviar para graficar///////////////
//*****Inicia el controlador maestro PD*****//
phi=angulo*(3.1415/180.0);//se convierte el ángulo a radianes
e=phid -phi;//error maestro
p=KP*e;//proporcional maestro
d=(phi_1-phi)*KD*iTs;//derivativo maestro
omega_d=p+d+omegaA;
phi_1=phi;
omegadD=(float)omega_d;//velocidad deseada rueda derecha
omegadI=(float)omega_d;//velocidad deseada rueda izquierda
if(SDL==1&&SIL==0)
{
   if (SP_TRUE==0)
   {
      KVI=0.4;
      KVD=1.0;
   }
   else if (SP_TRUE==1)
   {
      KVI=1.0;
      KVD=1.0;
   }
}
else if(SDL==0&&SIL==1)
{
   if (SP_TRUE==0)
   {
      KVI=1.0;
      KVD=0.4;
   }
   else if(SP_TRUE==1)
   {
      KVI=1.0;
      KVD=1.0;
   }
}
```

```
else
{
   if(SP_TRUE==0)
   {
      KVI=1.0;
      KVD=1.0;
   }
   else{
      KVI=1.0;
      KVD=1.0;
   }
}
//Al terminar se hace el ajuste de velocidad para
//Después ver que sensores altos estan activos SDH y SIH
omegadI=(float)omegadI*KVI;//Se multiplica por el factor de velocidad
omegadD=(float)omegadD*KVD;//Se multiplica por el factor de velocidad
/****Final sensores bajos****/
/******************inicio sensores altos*******************/
//********SDH y SIH*********//
if(SDH==1&&SIH==0)
{
   if (SP_TRUE==0)
   {
      KVI=0.2;
      KVD=1.0;
      iI=0.0;
   }
   else if (SP_TRUE==1)
   {
      KVI=1.0;
      KVD=1.0;
   }
}
else if(SDH==0&&SIH==1)
{
   if (SP_TRUE==0)
   {
      KVI=1.0;
      KVD=0.2;
      iD=0.0;
   }
   else if(SP_TRUE==1)
   {
      KVI=1.0;
      KVD=1.0;
```

```
   }
}
else
{
   if(SP_TRUE==0)
   {
      KVI=1.0;
      KVD=1.0;
   }
   else{
      KVI=1.0;
      KVD=1.0;
   }
}
omegadI=(float)omegadI*KVI;//Se multiplica por el factor de velocidad
omegadD=(float)omegadD*KVD;//Se multiplica por el factor de velocidad

omegadD_G=(omegadD+25.0)*M2;///////////////Enviar para graficar///////////////
omegadI_G=(omegadI+25.0)*M2;///////////////Enviar para graficar///////////////
//*****Inicia el controlador esclavo PI del motor derecho*****//
posDER_1=posDERF;//posición anterior rueda derecha
posDER=posD;//asignación de numero de cuentas
posDERF=(float)(escENCODER*posDER);//escalamiento de cuentas a posición
e_posDER=posDERF;//error de posición rueda derecha
posD=0;
omegaD=(e_posDER)/Ts;//Velocidad rueda derecha
omegaD_G=(omegaD+25.0)*M2;///////////////Enviar para graficar///////////////
eD=omegadD - omegaD;//error esclavo derecho
pD=KPD*eD;//proporcional rueda derecha
if((iD<VM)&&(iD>-VM))//rango de voltaje máximo
   iD=iD+KID*Ts*eD;//integral rueda derecha
else//si se pasa del rango de voltaje máximo la integral
{
   if(iD>=VM)
      iD=VN;
   if(iD<=(-VM))
      iD=-VN;
}
uD=pD+iD;//voltaje aplicado al motor derecho
if(uD>VM)//si se pasa del voltaje máximo se iguala al voltaje nominal máximo//
   uD=VN;
if(uD<-VM)
   uD=-VN;
uD_G=(uD+6.0)*M3;///////////////Enviar para graficar///////////////
//*****Inicia el controlador esclavo PI del motor izquierdo*****//
```

```
posIZQ_1=posIZQF;//posición anterior rueda izquierda
posIZQ=posI;//asignación de numero de cuentas
posIZQF=(float)(escENCODER*posIZQ);//escalamiento de cuentas a posición
e_posIZQ=posIZQF;//error de posición rueda derecha
posI=0;
omegaI=(e_posIZQ)/Ts;//Velocidad rueda derecha
omegaI_G=(omegaI+25.0)*M2;///////////////Enviar para graficar///////////////
eI=omegadI - omegaI;//error esclavo derecho
p_I=KPI*eI;//proporcional rueda derecha
if((iI<VM)&&(iI>-VM))//rango de voltaje máximo
   iI=iI+KII*Ts*eI;//integral rueda derecha
else//si se pasa del rango de voltaje máximo la integral
{
   if(iI>=VM)
      iI=VN;
   if(iI<=(-VM))
      iI=-VN;
}
uI=p_I+iI;//voltaje aplicado al motor derecho
if(uI>VM)//si se pasa del voltaje máximo se iguala al voltaje nominal máximo//
   uI=VN;
if(uI<-VM)
   uI=-VN;
uI_G=(uI+6.0)*M3;///////////////Enviar para graficar///////////////
//*****Termina el controlador esclavo PI del motor derecho*****//
//*****Sensores seguidores de línea*****//
//*****SDL y SIL*****//

/*****************Final sensores altos*****************/
uI=(float)escPWM*uI;//Se escala al ciclo de trabajo del PWM
uD=(float)escPWM*uD;//Se escala al ciclo de trabajo del PWM
//*****Control de direción de motor izquierdo*****//
if(uI<0)
{
   pwmI=abs(uI);
   PD2=0;
   PD3=1;
}
else
{
   pwmI=abs(uI);
   PD2=1;
   PD3=0;
}
//*****Control de dirección de motor derecho*****//
```

```
      if(uD<0)
      {
         pwmD=abs(uD);
         PD0=0;
         PD1=1;
      }
      else
      {
         pwmD=abs(uD);
         PD0=1;
         PD1=0;
      }
      //*****Aplicación de pwm a puente H*****//
      PWMFI=(unsigned)pwmI;//Quitamos signo
      PWMFD=(unsigned)pwmD;//Quitamos isgno
      set_pwm2_duty (PWMFI);//Pwm izquierdo
      set_pwm1_duty (PWMFD);//Pwm derecho
      //***************************Envio de datos a devc*******************//
      sensores=0;//Se hace cero para actualizar los sensores
      sensores=SIH;//Se iguala para que SD quede el bit 1 (0b0000000"SIH")
      sensores=sensores<<1|SIL;//Se hace un corrimiento desplazando SD(0b000000"SIH""SIL")
      sensores=sensores<<1|SP_TRUE;//Se hace un corrimiento desplazando SD(0b00000"SIH""SIL""SP")
      sensores=sensores<<1|SDL;//Se hace otro corrimiento desplazando SD(0b0000"SIH""SIL""SP""SDL")
      sensores=sensores<<1|SDH;//Se hace otro corrimiento desplazando SD(0b000"SIH""SIL""SP""SDL""SDH")
      putc(0xAA);//Envio de bandera
      putc(phi_G);//Envio de phi
      putc(phid_G);//Envio de phid
      putc(omegadD_G);//Envio de velocidad deseada derecha
      putc(omegaD_G);//Envio de velocidad derecha
      putc(uD_G);//Envio de esfuerzo de control derecho
      putc(omegadI_G);//Envio de velocidad deseada izquierda
      putc(omegaI_G);//Envio de velocidad izquierda
      putc(uI_G);//Envio de esfuerzo de control izquierdo
      putc(sensores);//Envio de sensores SIH,SIL,SP,SDL,SDH
   }
   set_timer0(65300);//20 [ms] 64597//15 [ms] 64831 10 [ms] 65066 5 [ms] 65300
}
//*****Interrupción puerto B para lectura de encoders*****//
#int_rb
void rb_isr()
{
   puerto=PORTB;
   AB=((puerto)&(0x30))>>4;
   aux=AB^AB_1;
   if(aux!=0)
```

```
      if(aux!=3)
         if(((AB_1<<1)^AB)&(0x02))
            posI++;
         else
            posI--;
   AB_1=AB;
   BC=((puerto)&(0xC0))>>6;
   aux=BC^BC_1;
   if(aux!=0)
      if(aux!=3)
         if(((BC_1<<1)^BC)&(0x02))
            posD++;
         else
            posD--;
   BC_1=BC;
}
//*****Función principal*****//
void main()
{
   set_tris_a(0b11111111);
   set_tris_b(0b11110011);
   set_tris_c(0b01000110);
   set_tris_d(0b00000000);
   set_tris_e(0b00000000);
   mpu6050_init();//Se incicializa el sensor MPU6050
   setup_timer_2(T2_DIV_BY_16,255,1);//Se configura el timer 2 para el PWM
   setup_ccp1(CCP_PWM);//Se configura ccp1 como pwm
   setup_ccp2(CCP_PWM);//Se configura ccp2 como pwm
   setup_timer_0(T0_INTERNAL|T0_DIV_256);//Se configura el timer 0 para el periodo de muestreo
   set_timer0(65300);//Se inicializa el timer
   //*****Se habilitan las interrupciones*****//
   enable_interrupts(int_timer0);
   enable_interrupts(int_rda);
   enable_interrupts(int_rb);
   enable_interrupts(global);
   delay_ms(5000);//Delay para acomodar el robot balancín en posición de 5 [s]
   flag=1;//Bandera para empezar el control
   while(true)
   {
      mpu6050_datos();//Se obtienen datos del sensor MPU6050
   }
}
 void mpu6050_write(int add, int data)
{
   i2c_start();//Inicia la comunicación i2c
```

```
   i2c_write(W_DATA);//Manda la dirección para escribir
   i2c_write(add);//Agrega la dirección a la cual queremos acceder
   i2c_write(data);//Agrega el valor que queremos poner
   i2c_stop();//Termina la comunicación
}
void mpu6050_init(){
   mpu6050_write(0x19,0b00000001);//Divisor de frecuencia de muestreo
   mpu6050_write(0x6B,0b00000001);//Reset, dormir, ciclo, temperatura, clock
   mpu6050_write(0x1A,0b00000000);//Sincronización externa, filtro
   mpu6050_write(0x1B,0b01000000);//Configuración de giroscopios gY, gX, gZ
   mpu6050_write(0x1C,0b10000000);//Configuración de acelerómetros aX, aY, aZ
}
void mpu6050_datos()
{
   i2c_start();//Inicia la comunicación i2c
   i2c_write(W_DATA);//Manda la dirección para escribir
   i2c_write(AX_H);//Manda la dirección de la aceleración parte alta
   i2c_start();//Inicia la comunicación i2c
   i2c_write(R_DATA);//Mandamos la dirección para leer
   aDATA[0]=i2c_read(0);//Adquirimos la aceleración parte alta
   i2c_stop();//Termina la comunicación

   i2c_start();//Inicia la comunicacion i2c
   i2c_write(W_DATA);//Manda la dirección para escribir
   i2c_write(AX_L);//Manda la dirección de la aceleración parte baja
   i2c_start();//Inicia la comunicacion i2c
   i2c_write(R_DATA);//Mandamos la dirección para leer
   aDATA[1]=i2c_read(0);//Adquirimos la aceleración parte baja
   i2c_stop();//Termina la comunicación

   i2c_start();//Inicia la comunicacion i2c
   i2c_write(W_DATA);//Manda la dirección para escribir
   i2c_write(GY_H);//Manda la dirección de la velocidad angular parte alta
   i2c_start();//Inicia la comunicacion i2c
   i2c_write(R_DATA);//Mandamos la dirección para leer
   gDATA[0]=i2c_read(0);//Adquirimos la velocidad angular parte alta
   i2c_stop();//Termina la comunicación

   i2c_start();//Inicia la comunicacion i2c
   i2c_write(W_DATA);//Manda la dirección para escribir
   i2c_write(GY_L);//Manda la dirección de la velocidad angular parte baja
   i2c_start();//Inicia la comunicacion i2c
   i2c_write(R_DATA);//Mandamos la dirección para leer
   gDATA[1]=i2c_read(0);//Adquirimos la velocidad angular parte baja
   i2c_stop();//Termina la comunicación
```

```
   //*****Se unen los 8 bytes parte alta y baja de aX y gY
   aX=make16(aDATA[0],aDATA[1])-10590;//Se resta -10590 para tener en cero la aceleración
   gY=make16(gDATA[0],gDATA[1])+5700;//Se suma 5700 para igualar la velocidad angular a cero
}

void Linea()//Función seguidor de línea
{
//Asignaciones de variables para los sensores bajos SIL y SDL
   if(PA1==0&&PA3==1)
   {
       SDL=0;
       SIL=1;
   }
   else if(PA1==1&&PA3==0)
   {
       SDL=1;
       SIL=0;
   }
   else
   {
       SDL=0;
       SIL=0;
   }
//Asignaciones de variables para los sensores altos SIH y SDH
   if(PA0==0&&PA4==1)
   {
       SDH=0;
       SIH=1;
   }
   else if(PA0==1&&PA4==0)
   {
       SDH=1;
       SIH=0;
   }
   else
   {
       SDH=0;
       SIH=0;
   }
}

void Rampa()//Función de rampa
{
   if(PA2==1)
```

```
   {
      SP=0;
   }
   else
   {
      SP=1;
   }
   //Filtro pasa bajas para detectar la pendiente
   //con 14 muestras para evitar falsas pendientes
   SPN[13]=SPN[12];
   SPN[12]=SPN[11];
   SPN[11]=SPN[10];
   SPN[10]=SPN[9];
   SPN[9]=SPN[8];
   SPN[8]=SPN[7];
   SPN[7]=SPN[6];
   SPN[6]=SPN[5];
   SPN[5]=SPN[4];
   SPN[4]=SPN[3];
   SPN[3]=SPN[2];
   SPN[2]=SPN[1];
   SPN[1]=SPN[0];
   SPN[0]=SP;
   //Si las 14 muestras son 1 entonces hay pendiente
   if (SPN[13]==1 && SPN[12]==1 && SPN[11]==1 && SPN[10]==1 && SPN[9]==1 && SPN[8]==1 && SPN[7]==1 && SPN[6]==
   {
      SP_TRUE=1;
   }//Si las 14 muestras son 0 no hay pendiente
   else if (SPN[13]==0 && SPN[12]==0 && SPN[11]==0 && SPN[10]==0 && SPN[9]==0 && SPN[8]==0 && SPN[7]==0 && SPN
   {
      SP_TRUE=0;
   }
}
```

B. PROGRAMA EN DEV C++ DEL ROBOT BALANCÍN PARA GUARDAR LA INFORMACIÓN EN UN ARCHIVO DE TEXTO

A continuación se muestra el código en Dev C++ que almacena los datos en un archivo de texto.

```
#include <iostream>
#include <string.h>
#include <dos.h>
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#define Ts 0.005
//*****Recepción pic CCS*****//
unsigned char phi_G=0;
unsigned char phid_G=0;
unsigned char omegadD_G=0;
unsigned char omegaD_G=0;
unsigned char uD_G=0;
unsigned char omegadI_G=0;
unsigned char omegaI_G=0;
unsigned char uI_G;
unsigned char pic_sensores=0;
unsigned char inter;
//*****Variables a graficar*****//
float phi=0.0;
float phid=0.0;
float omegadD=0.0;
float omegaD=0.0;
float uD=0.0;
float omegadI=0.0;
float omegaI=0.0;
float uI=0.0;
int SDH=0,SDL=0,SIH=0,SIL=0,SP=0;
//*****Variables de control*****//
int flagcom=0;
float t=0.0;
```

```
int main()
{
HANDLE h; /*handler, sera el descriptor del puerto*/
    DCB dcb; /*estructura de configuración*/
    DWORD dwEventMask; /*mascara de eventos*/
    FILE *fp;

    if((fp=fopen("MONIT.txt","w+"))==NULL)
{
  printf("No se puede abrir el archivo.\n");
  exit(1);
}



    /*abrimos el puerto*/
    h=CreateFile("COM3",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);
    if(h == INVALID_HANDLE_VALUE)
{
         /*ocurrio un error al intentar abrir el puerto*/
    }
/*obtenemos la configuración actual*/
if(!GetCommState(h, &dcb))
{
         /*error: no se puede obtener la configuración*/
    }
    /*Configuramos el puerto*/
dcb.BaudRate = 115200;
    dcb.ByteSize = 8;
    dcb.Parity = NOPARITY;
    dcb.StopBits = TWOSTOPBITS;
    dcb.fBinary = TRUE;
    dcb.fParity = TRUE;
    /* Establecemos la nueva configuración */
    if(!SetCommState(h, &dcb))
{
        /* Error al configurar el puerto */
    }
    DWORD n;
    char enviar;
    int recibido;
    /* Para que WaitCommEvent espere el evento RXCHAR */
    printf("vamos a esperar\r\n");
    SetCommMask(h, EV_RXCHAR);
    while(1)
{
```

```
     recibido=0;
        while(1)
{
         ReadFile(h, &recibido, 1/* leemos un byte */, &n, NULL);
            if(!n)
             break;
            else
{
             if(flagcom!=0)
flagcom++;
if(recibido==0xAA)
    {
         flagcom=1;
         recibido=0;
        }
        if(flagcom==2)
        {
          phi_G=recibido;
      }
      if(flagcom==3)
        {
          phid_G=recibido;
      }
      if(flagcom==4)
        {
          omegadD_G=recibido;
      }
      if(flagcom==5)
        {
          omegaD_G=recibido;
      }
      if(flagcom==6)
        {
          uD_G=recibido;
      }
      if(flagcom==7)
        {
          omegadI_G=recibido;
      }
      if(flagcom==8)
        {
          omegaI_G=recibido;
      }
      if(flagcom==9)
        {
```

```
          uI_G=recibido;
      }
         if(flagcom==10)
         {
         phi=((180.0/255.0)*phi_G-90.0)*3.1415/180.0;
         phid=((180.0/255.0)*phid_G-90.0)*3.1415/180;
         omegadD=((50.0/255.0)*omegadD_G)-25.0;
         omegaD=((12.0/255.0)*omegaD_G)-6.0;
         uD=(12.0/255.0*uD_G)-6.0;
         omegadI=((50.0/255.0)*omegadI_G)-25.0;
         omegaI=((12.0/255.0)*omegaI_G)-6.0;
         uI=(12.0/255.0*uI_G)-6.0;
pic_sensores=recibido;

inter=pic_sensores;
inter=inter&0b00010000;
SIH=inter>>4;
SIH=SIH;

inter=pic_sensores;
inter=inter&0b00001000;
SIL=inter>>3;
SIL=SIL;

inter=pic_sensores;
inter=inter&0b00000100;
SP=inter>>2;
SP=SP;

inter=pic_sensores;
inter=inter&0b00000010;
SDL=inter>>1;
SDL=SDL;

inter=pic_sensores;
SDH=inter&0b00000001;
SDH=SDH;
fprintf(fp,"%3.3f\t%3.3f\t%3.3f\t%3.3f\t%3.3f\t%3.3.f\t%3.3f\t%3.3f\t%3.3f\t%d\t%d\t%d\t%d\t%d\n",t,phid,phi,o
t=t+Ts;
printf("phi=%f\r\n",phi);
flagcom=0;
    }
    }
     }//CIERRE WHILE()
    }//CIERRE WHILE()
```

```
fclose(fp);
return 0;
}
```

C. PROGRAMA PARA GRÁFICAR LOS RESULTADOS GUARDADOS EN EL ARCHIVO DE TEXTO CON MATLAB

A continuación se muestra el código que gráfica los resultados en Matlab 2017b obtenidos en el archivo de texto.

```
%Gráfica de control de posición.
clc;
clear all;
close all;
%leyendo las muestras tomadas al sistema.
%fid=fopen('MONIT_bien.txt','r');%se abre el archivo.
fid=fopen('MONIT.txt','r');%se abre el archivo.
datos=fscanf(fid, '%f', [14 inf]);% datos tiene 12 renglones.
fclose(fid);
t=datos(1,:);       %se obtiene la primera columna y se agrega a t.
phid=datos(2,:);    %se obtiene la segunda columna y se agrega a phid.
phi=datos(3,:);     %se obtiene la tercer columna y se agrega a phi.
omegadI=datos(4,:); %se obtiene la cuarta columna y se agrega a omegadI.
omegaI=datos(5,:);  %se obtiene la quinta columna y se agrega a omegaI.
omegadD=datos(6,:); %se obtiene la sexta columna y se agrega a omegadD.
omegaD=datos(7,:);  %se obtiene la séptima columna y se agrega a omegadD.
uI=datos(8,:);      %se obtiene la octava columna y se agrega a uI.
uD=datos(9,:);      %se obtiene la novena columna y se agrega a uD.
sIH=datos(10,:);    %se obtiene la décima columna y se agrega a sIH.
sIL=datos(11,:);    %se obtiene la onceaba columna y se agrega a sIL.
sP=datos(12,:);     %se obtiene la doceava columna y se agrega a sP.
sDL=datos(13,:);    %se obtiene la onceaba columna y se agrega a sDL
sDH=datos(14,:);    %se obtiene la onceaba columna y se agrega a sDH
%**************************primera gráfica*****************************%
%Se gráfica en el primer canal la posición deseada (phid) y la posición
%actual (phi) contra el tiempo (t).
figure(1)
subplot(5,1,1)
plot(t,phid,t,phi);
xticklabels({})
set(legend('$\phi_d[rad]$','$\phi[rad]$'), 'interpreter', 'latex')
grid on;
%Se gráfica en el segundo canal la velocidad deseada del controlador
```

```
%maestro
subplot(5,1,2)
plot(t,omegadI,t,omegaI);
xticklabels({})
ylim([-21 21])
set(legend('$\omega_{dI}[\frac{rad}{s}]$','$\omega_I[\frac{rad}{s}]$'), 'interpreter', 'latex')
grid on;
%Se gráfica en el tercer canal el voltaje aplicado al motor izquierdo (uI).
subplot(5,1,3)
plot(t,uI);
xticklabels({})
set(legend('$u_I[V]$'), 'interpreter', 'latex')
grid on;
%Se gráfica en el cuarto canal la velocidad deseada del motor derecho
%(omegadD) y la velocidad actual de la llanta derecha (omegaD) contra el
%tiempo (t)
subplot(5,1,4)
plot(t,omegadD,t,omegaD);
xticklabels({})
ylim([-25 25])
set(legend('$\omega_{dD}[\frac{rad}{s}]$','$\omega_D[\frac{rad}{s}]$'), 'interpreter', 'latex')
grid on;
%Se gráfica en el quinto canal el voltaje aplicado al motor derecho (uD).
subplot(5,1,5)
plot(t,uD);
set(legend('$u_D[V]$'), 'interpreter', 'latex')
xlabel('$t[s]$', 'interpreter', 'latex')
grid on;
%**************************segunda gráfica*****************************%
%Se gráfica en el primer canal la respuesta del sensor seguidor de línea
%izquierdo SIH
figure(2)
subplot(5,1,1)
plot(t,sIH);
xticklabels({})
ylim([0 1.1])
set(legend('$sIH$'), 'interpreter', 'latex')
grid on;
%Se gráfica en el segundo canal la respuesta en el tiempo del sensor
%seguidor de línea derecho SIL
subplot(5,1,2)
plot(t,sIL);
xticklabels({})
ylim([0 1.1])
set(legend('$sIL$'), 'interpreter', 'latex')
```

```
grid on;
%En el tercer canal se gráfica la respuesta en el tiempo del sensor que detecta la
%pendiente sP
subplot(5,1,3)
plot(t,sP);
xticklabels({})
ylim([0 1.1])
set(legend('$sP$'), 'interpreter', 'latex')
grid on;
%Se gráfica en el segundo canal la respuesta en el tiempo del sensor
%seguidor de línea derecho SDL
subplot(5,1,4)
plot(t,sDL);
xticklabels({})
ylim([0 1.1])
set(legend('$sDH$'), 'interpreter', 'latex')
grid on;
%Se gráfica en el segundo canal la respuesta en el tiempo del sensor
%seguidor de línea derecho SDH
subplot(5,1,5)
plot(t,sDH);
ylim([0 1.1])
set(legend('$sDL$'), 'interpreter', 'latex')
xlabel('$t[s]$', 'interpreter', 'latex')
grid on;
%*****************************tercera gráfica*******************************%
%Se gráfica en el primer canal la posición deseada (phid) y la posición
%actual (phi) contra el tiempo (t).
figure(3)
subplot(2,1,1)
plot(t,phid,t,phi);
xticklabels({})
set(legend('$\phi_d[rad]$','$\phi[rad]$'), 'interpreter', 'latex')
grid on;
%En el tercer canal se gráfica la respuesta en el tiempo del sensor que detecta la
%pendiente sP
subplot(2,1,2)
plot(t,sP);
ylim([0 1.1])
set(legend('$sP$'), 'interpreter', 'latex')
xlabel('$t[s]$', 'interpreter', 'latex')
grid on;
```

D. VISTAS DE PIEZA DE SENSORES EN SOLIDWORKS 2018

A continuación se muestran las vistas frontal, superior, lateral y isométrica de la pieza de los sensores con cotas en $[cm]$.

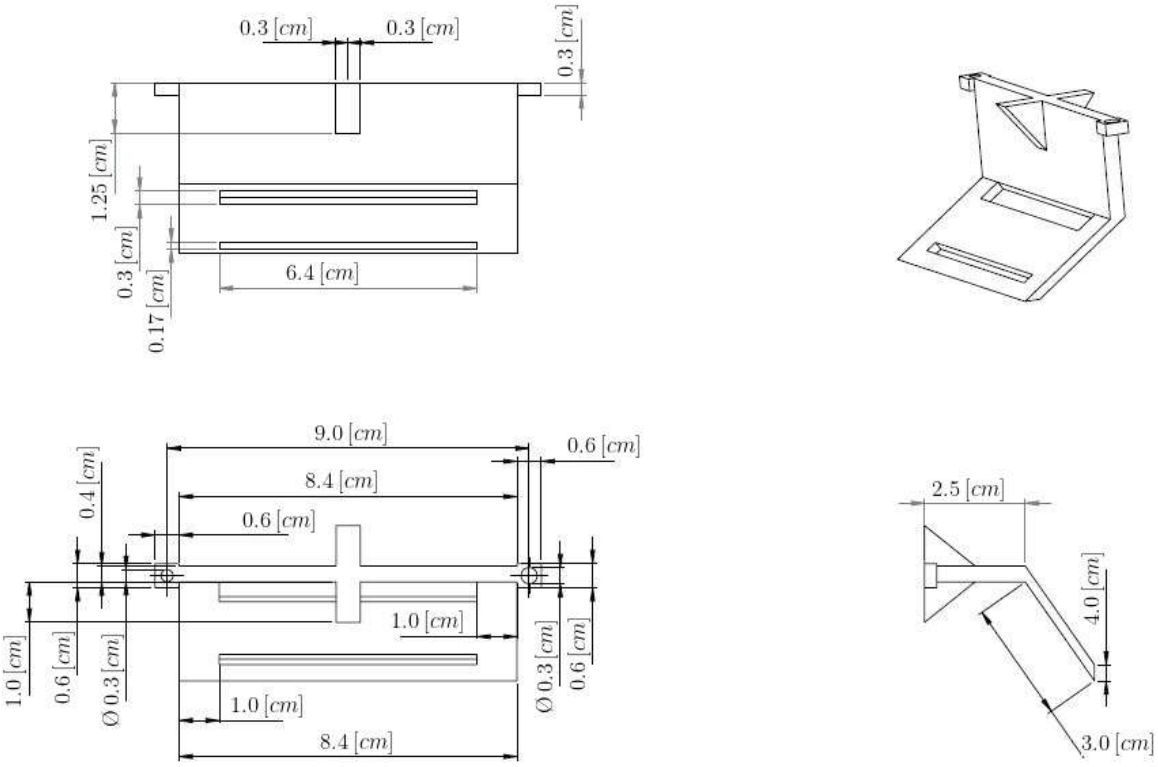

Figura D.1: Vistas de pieza de sensores en Solidworks 2018.