



GRADO EN INGENIERÍA ELECTROMECÁNICA

TRABAJO FIN DE GRADO

Control design of an active suspension system for a
scaled vehicle

Autor: José Esteban Rivero Ríos
Director: Juan Luis Zamora Macho

Madrid

Agosto de 2019

AUTHORIZATION FOR DIGITALIZATION, STORAGE AND DISSEMINATION IN THE NETWORK OF END-OF-DEGREE PROJECTS, MASTER PROJECTS, DISSERTATIONS OR BACHILLERATO REPORTS

1. Declaration of authorship and accreditation thereof.

The author Mr. /Ms. José Esteban Rivero Ríos

HEREBY DECLARES that he/she owns the intellectual property rights regarding the piece of work: Control design of an active suspension system for a scaled vehicle that this is an original piece of work, and that he/she holds the status of author, in the sense granted by the Intellectual Property Law.

2. Subject matter and purpose of this assignment.

With the aim of disseminating the aforementioned piece of work as widely as possible using the University's Institutional Repository the author hereby **GRANTS** Comillas Pontifical University, on a royalty-free and non-exclusive basis, for the maximum legal term and with universal scope, the digitization, archiving, reproduction, distribution and public communication rights, including the right to make it electronically available, as described in the Intellectual Property Law. Transformation rights are assigned solely for the purposes described in a) of the following section.

3. Transfer and access terms

Without prejudice to the ownership of the work, which remains with its author, the transfer of rights covered by this license enables:

- a) Transform it in order to adapt it to any technology suitable for sharing it online, as well as including metadata to register the piece of work and include "watermarks" or any other security or protection system.
- b) Reproduce it in any digital medium in order to be included on an electronic database, including the right to reproduce and store the work on servers for the purposes of guaranteeing its security, maintaining it and preserving its format.
- c) Communicate it, by default, by means of an institutional open archive, which has open and cost-free online access.
- d) Any other way of access (restricted, embargoed, closed) shall be explicitly requested and requires that good cause be demonstrated.
- e) Assign these pieces of work a Creative Commons license by default.
- f) Assign these pieces of work a **HANDLE** (*persistent URL*). by default.

4. Copyright.

The author, as the owner of a piece of work, has the right to:

- a) Have his/her name clearly identified by the University as the author
- b) Communicate and publish the work in the version assigned and in other subsequent versions using any medium.
- c) Request that the work be withdrawn from the repository for just cause.
- d) Receive reliable communication of any claims third parties may make in relation to the work and, in particular, any claims relating to its intellectual property rights.

5. Duties of the author.

The author agrees to:

- a) Guarantee that the commitment undertaken by means of this official document does not infringe any third party rights, regardless of whether they relate to industrial or intellectual property or any other type.

- b) Guarantee that the content of the work does not infringe any third party honor, privacy or image rights.
- c) Take responsibility for all claims and liability, including compensation for any damages, which may be brought against the University by third parties who believe that their rights and interests have been infringed by the assignment.
- d) Take responsibility in the event that the institutions are found guilty of a rights infringement regarding the work subject to assignment.


6. Institutional Repository purposes and functioning.

The work shall be made available to the users so that they may use it in a fair and respectful way with regards to the copyright, according to the allowances given in the relevant legislation, and for study or research purposes, or any other legal use. With this aim in mind, the University undertakes the following duties and reserves the following powers:

- a) The University shall inform the archive users of the permitted uses; however, it shall not guarantee or take any responsibility for any other subsequent ways the work may be used by users, which are non-compliant with the legislation in force. Any subsequent use, beyond private copying, shall require the source to be cited and authorship to be recognized, as well as the guarantee not to use it to gain commercial profit or carry out any derivative works.
- b) The University shall not review the content of the works, which shall at all times fall under the exclusive responsibility of the author and it shall not be obligated to take part in lawsuits on behalf of the author in the event of any infringement of intellectual property rights deriving from storing and archiving the works. The author hereby waives any claim against the University due to any way the users may use the works that is not in keeping with the legislation in force.
- c) The University shall adopt the necessary measures to safeguard the work in the future.
- d) The University reserves the right to withdraw the work, after notifying the author, in sufficiently justified cases, or in the event of third party claims.

Madrid, on 28 of August 2019,

HEREBY ACCEPTS

Signed.....

Reasons for requesting the restricted, closed or embargoed access to the work in the Institution's Repository

I, hereby, declare that I am the only author of the project report with title:

Control design of an active suspension system for a scaled vehicle

which has been submitted to ICAI School of Engineering of Comillas Pontifical University in the academic year 2018-2019. This project is original, has not been submitted before for any other purpose and has not been copied from any other source either fully or partially. All information sources used have been rightly acknowledged.

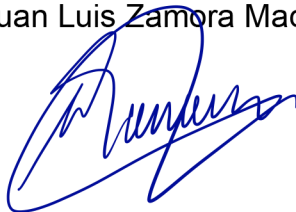
Fdo.: José Esteban Rivero Ríos Date: 28/08/2019

A handwritten signature in black ink, appearing to read 'JER', with a long horizontal stroke extending to the right.

I authorize the submission of this project

PROJECT SUPERVISOR

Fdo.: Juan Luis Zamora Macho Date: 28/08/2019

A handwritten signature in blue ink, appearing to read 'JL Zamora Macho', with a large, stylized initial 'J'.



COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA ELECTROMECÁNICA

TRABAJO FIN DE GRADO

Control design of an active suspension system for a
scaled vehicle

Autor: José Esteban Rivero Ríos

Director: Juan Luis Zamora Macho

Madrid

Agosto de 2019

Special Thanks:

To Juan Luis for his help and advice.

To my family for supporting throughout all these years.

Control design of an active suspension system for a scaled vehicle

Author: Rivero Ríos, José Esteban.

Director: Zamora Macho, Juan Luis.

Collaborating Entity: ICAI – Comillas Pontifical University

1. INTRODUCTION

This project consists of taking a RC car prototype with a working active suspension system [ALEX11] from a previous thesis [JIMÉ16] and upgrading its electronic hardware with the goal of being able to implement more complex control strategies than the hardware was previously capable of.

This project expects to reduce the gap the Comillas Pontifical University has on this field, by preparing a working prototype that can be used during classes or other projects in this field.

The objectives for this work are the following:

- Update the electronic components of the car, without hampering the performance of the active suspension system.
- Program the necessary drivers in MATLAB/Simulink for full vehicle functionality
- Identify the necessary models of the vehicle in order to design a control system.

The prototype consists of a Turnigy TD10 kit, shown in Figure 1, equipped with a custom designed active suspension system.



Figure 1: Turnigy TD10 touring car kit

The upgraded prototype will be powered by a Raspberry Pi 3 B+ alongside a 4 expansion boards from the MikroE family of Click boards enabling use of PWM, IMU, ADC and encoder capabilities to the Raspberry Pi.

This new electronic hardware will work alongside the already installed active suspension system, a Turnigy XK3650 3900KV brushless dc motor, a Turnigy Trackstar 1/10th 80A Turbo Electronic speed controller [MODE19], a 7.2V 2-cell LiPo battery, and four Sharp infrared sensors.

The ADC and encoder modules will require programming the driver from scratch, while the PWM, IMU and RC receiver have existing drivers that can be easily implemented and modified if necessary.

2. METHODOLOGY

In order to accomplish the objectives of this work, the first step was to check the state of the prototype. After that any old, incompatible, or faulty hardware was removed or replaced, work began on installing the new components onto the car chassis. After all the new hardware was installed, work began on preparing and programming the necessary drivers for correct functionality of all the hardware on board the car.

Then, work began on identifying the models of the vehicle. Unfortunately, due to difficulties in obtaining one model and a general lack of time, work on designing the necessary controls was not started.

1. Peripheral choice

The Raspberry Pi is the chosen control unit, and while it is capable microcomputer it lacks accessories needed to fully operate the RC car with its active suspension system. To solve

this 4 expansion boards are used alongside the Raspberry Pi in order to operate the entire system. Thanks to MATLAB's and Simulink's support package for the Raspberry Pi, setting up the necessary communications between the Raspberry pi and the boards is quite simple.

The first expansion board is the PWM click board which handles all the PWM signals need for the suspension servos, direction servo, and the Brushless motor. All of this is controlled though I2C protocol [CIRC16].

The Second expansion board is the ADC3 Click, which is a 4-channel analogue to digital converter, which receives all the signals from the 4 infrared sensors already installed which are used for the height measurement of the vehicle. This component also communicates with the Raspberry through I2C protocol.

Third expansion board is the Counter click, which takes the phase sensor output from the brushless motor to count the number of revolutions the motor makes, as this output matches a quadrature encoder output. With this count the model can calculate the true speed of the car. This board is controlled through the SPI protocol [MIKE17].

The last expansion board needed is the MPU IMU Click which is an inertial motion unit [CHEN94] that has a 3-axis gyroscope [DEIM31] and accelerometer [MOHA18], finishing the sensors needed for the correct operation of the active suspension system. This board communicates with the Raspberry PI through SPI protocol as well.

One last electronic component is the FS-A8S radio control receiver, which communicates with the Raspberry PI through its serial port.

2. Installation of new hardware

After checking for faulty hardware, the first step was to remove the old electronic hardware that will be replaced. In this case it was an old Arduino based microcontroller and the old radio control receiver which is incompatible with the new hardware.

Afterward the installation of new hardware began, correcting the lack of space on the chassis by adding an expansion plate to the chassis, creating a second level where most of

the new components were installed. The finished result can be seen in figure 2:

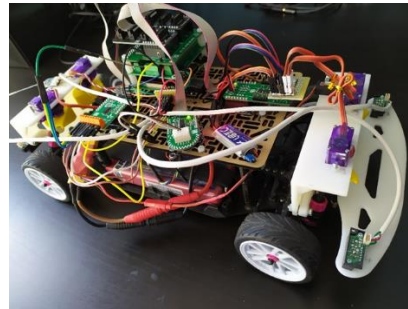


Figure 2.: Finished RC car

3. Software setup

After all the new hardware is installed, the next step starts with setting up all the necessary software to control the hardware available on the vehicle

First was setting up the radio control receiver, in this case a FS-A8S receiver, which uses the IBUS protocol over the Raspberry PI's serial port. Multiple drone projects use this communication protocol, so a working code was available and with some changes, the component was implemented successfully.

The next component to setup was the PWM click board, which will control all the installed motors on the car. This was also simple, as another project had utilized this board as well, although more changes were needed for it to be fully functional. The most important changes consisted of correctly assigning each channel to each motor and setting the servomotor safety limits as it was judged that the limits were better set, on the module's code instead of in the main control code.

The next component configured was the ADC3 click board. This one required the driver to be coded from scratch. First a configuration MATLAB .m script was made in order to initialize all the initial configuration of the board, as well as loading the required I2C addresses into memory. Then a Simulink model was created with 2 main parts: an initialization phase which oversees proper initialization of the board, and a main phase, enabled once the previous phase is complete. The main phase handles the main operation loop of the ADC which consists of sequentially measuring the output of each infrared sensor, reading the data from the ADC and processing

it to obtain a measurement in volts. The ADC model also includes the small conversion code for the infrared sensor.

Then next board to be implemented was the MPU IMU click, this one also had a working driver from previous drone projects and was implemented with some small changes. The code was also setup to work with an Extended Kalman Filter (EKF) [SING18] also previously coded, with a little setup this was also implemented successfully.

The final board to implement is the Counter click board. This one also needed to be coded from scratch and follows the same procedure as the ADC3 click, a MATLAB .m script file with all the necessary address and variables, a Simulink model with two phases, one to configure the board, and the other for its main operation. This board's main operation consists of counting each revolution the motor makes, and then each sample time the code calculates the difference between the count now and during the last sampling time to obtain the revolutions per sampling time, which is then converted to motor RPM and finally the vehicle's speed in m/s.

These models are all implemented in a master Simulink model which is structured into 3 categories: HARDWARE, MONITORIZATION, and CONTROL. All these sections are tied together by a data bus that handles all the variables the model needs to function.

The hardware section contains all the models that pertain to using a specific type of hardware, the previously mentioned models are all included here.

The monitorization section handles all the needed scopes to view the telemetry data from the vehicle during operation, relayed through the data busses.

And lastly, the control section contains the state machine and the control blocks, these are implemented as MATLAB function blocks. The state machine guides the code through 5 self-explanatory states: Boot, Sensor calibration, operation point set, standby, locked motors and operation. The control block handles basic operation like relaying receiver commands to the motors as well as

the control strategies implemented and other control critical code like the EKF.

This master Simulink model is complimented by a library of .m scripts that contain all the necessary data to initialize the model. These are led by the main configuration script that when run, loads all the other scripts in order, setting up everything from control initialization, to the creation of the extremely useful data busses.

One last thing before the identification of the models, is the creation of a simple speed PI regulator, to ensure that during the tests the speed of the car does not fluctuate. This regulator was made by simply setting up a simple PI, changing the K and Ti between test until a satisfactory result was achieved.

4. Identifying the vehicle's models

For the chassis to achieve stability, the active suspension system must be able to at least control the following variables:

- Pitch angle of the vehicle
- Roll angle of the vehicle
- The vertical acceleration (height) of the vehicle

Therefore, the project needs at least those models before the control can be designed.

To accomplish this a set of 3 test were created. The tests consisted of using a PRBS signal [JACK71] to control the motions of the active suspension system with the goal of only having one type of movement, e.g. for the pitch test the system only changed the pitch angle, etc. The measurements from the IMU were recorded and then processed using MATLAB's system identification toolbox to calculate a transfer function describing the model.

The results were as follows:

- The pitch model was identified with an 87.96% match
- The roll model was identified with a 92.57% match
- The height model was not identified as its match was around the 39%

Therefore, the height test needed redesigning, work began on applying a least squares estimation method [GIOR85] when the project ran out of time, and as such the height model was not identified nor the controls designed.

3. RESULTS

The results of this work are composed of the working prototype alongside the Simulink model that controls it.

The electronics of the car have been fully upgraded, with no drawback to the active suspension system, and while the car is heavier the active suspension system is more than powerful enough so that this is not an issue.

The Simulink model created allows for full control of the vehicle in manual mode using the joysticks on the transmitter to manually control the pitch and roll angles, full access to all the sensors and measurements the electronic system takes, and finally the necessary framework so that a control scheme can be easily implemented without many changes needed to the model as a whole.

4. CONCLUSIONS

In the end only 2 of the main objectives of this work were completed, both of which involved completing the RC car prototype and its control model, so that the vehicle is operational.

The last objective was not strictly necessary as the manual operation shows the full functionality of the prototype and previous work has demonstrated that this active suspension system is effective in stabilizing the chassis when driving through irregular terrain.

5. References

[ALEX11] C. Alexandru and P. Alexandru, "A comparative analysis between the vehicles'," INTERNATIONAL JOURNAL OF MECHANICS, vol. 5, no. 4, p. 8, 2011.

[JIMÉ16] F. d. B. Jiménez Valverde, "Diseño de un sistema de suspensión activa para un

vehículo a escala," Trabajo de fin de grado, ICAI, Universidad Pontificia Comillas, Madrid, 2016.

[AZUM19] T. Azuma, "What Is Traction Control And How Does It Work? - CAR FROM JAPAN," 25 February 2019. [Online]. Available: <https://carfromjapan.com/article/car-maintenance/traction-control-system/>. [Accessed 25 August 2019].

[MODE19] Modelflight, "What is an Electronic Speed Controller and how does it differ from brushed to brushless motors?," Modelflight, 06 February 2019. [Online]. Available: <https://www.modelflight.com.au/blog/electronic-speed-controllers>. [Accessed 25 August 2019].

[MIKE17] MIKEGRUSIN, "Serial Peripheral Interface (SPI) - Learn.sparkfun.com," SparkFun, 2017. [Online]. Available: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all>. [Accessed 25 August 2019].

[CHEN94] J.-H. Chen, S.-C. Lee and D. B. DeBra, "Gyroscope free strapdown inertial measurement," Journal of Guidance, Control, and Dynamics, vol. 17, no. 2, pp. 286-290, 1994.

[CHEN94] J.-H. Chen, S.-C. Lee and D. B. DeBra, "Gyroscope free strapdown inertial measurement," Journal of Guidance, Control, and Dynamics, vol. 17, no. 2, pp. 286-290, 1994.

[MOHA18] Z. Mohammed, I. M. Elfadel and M. Rasras, "Monolithic Multi Degree of Freedom (MDoF) Capacitive MEMS Accelerometers," Micromachines, vol. 9, no. 11, p. 602, 2018.

[DEIM31] R. F. Deimel, "Mechanics of the Gyroscope," Nature, vol. 128, no. 3225, pp. 289-289, 1931.

[JACK71] P. A. Jackson, "PRBS cross-correlation measurements by hybrid computational techniques," The Computer Journal, vol. 14, no. 1, pp. 49-54, 1971.

[GIOR85] A. A. Giordano and F. M. Hsu, Least Square Estimation with Applications to Digital Signal Processing, New York: John Wiley & Sons, Inc., 1985.

Control de un sistema de suspensión activa para un vehículo a escala

Autor: Rivero Ríos, José Esteban.

Director: Zamora Macho, Juan Luis.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

1. INTRODUCCIÓN

Este proyecto consiste en tomar un prototipo de coche RC con un sistema de suspensión activa [ALEX11] de TFG previo [JIMÉ16] y actualizar su hardware electrónico con el objetivo de poder implementar estrategias de control más complejas de las que el hardware era capaz anteriormente.

Este proyecto espera reducir la brecha que la Universidad Pontificia Comillas tiene en este campo, al preparar un prototipo funcional que pueda usarse durante las clases u otros proyectos en este campo.

Los objetivos de este trabajo son los siguientes:

- Actualizar los componentes electrónicos del automóvil, sin obstaculizar el rendimiento del sistema de suspensión activa.
- Programar los controladores necesarios en MATLAB / Simulink para la funcionalidad completa del vehículo
- Identificar los modelos necesarios del vehículo para diseñar un sistema de control.

El prototipo consiste en un kit Turnigy TD10, que se muestra en la Figura 1, equipado con un sistema de suspensión activa diseñado a medida.



Figura 1.: Kit de turismos Turnigy TD10

El prototipo actualizado estará controlado por una Raspberry Pi 3 B+ junto con 4 tarjetas de expansión de la familia MikroE de tarjetas Click que permiten el uso de PWM, IMU, ADC y encoder a la Raspberry Pi.

Este nuevo hardware electrónico funcionará junto con el sistema de suspensión activa ya instalado, un motor de CC sin escobillas Turnigy XK3650 3900KV, un controlador de velocidad electrónico (ESC en inglés) [MODE19] Turnigy Trackstar Turbo 1 / 10th 80ª, una batería LiPo de 7.2V de 2 celdas, y cuatro sensores infrarrojos de distancia Sharp.

Los módulos ADC y codificador requerirán programar el controlador desde cero, mientras que el PWM, IMU y receptora de RC tienen controladores existentes que pueden implementarse y modificarse fácilmente si es necesario.

2. METODOLOGÍA

Para lograr los objetivos de este trabajo, el primer paso fue verificar el estado del prototipo. Una vez desinstalado/reemplazado todo el hardware viejo o defectuoso, se comenzó a instalar los nuevos componentes al chasis del coche. Después de instalar todo el nuevo hardware, se comenzó a trabajar en la preparación y programación de los controladores necesarios para la correcta

funcionalidad de todo el hardware a bordo del automóvil.

Luego, se comenzó a trabajar para identificar los modelos del vehículo. Desafortunadamente, debido a las dificultades para obtener un modelo en específico y una falta de tiempo, no se comenzó a trabajar en el diseño de los controles necesarios.

1. Elección de periféricos

La Raspberry Pi es la unidad de control elegida para este proyecto, y aunque es un microordenador capaz, carece de los accesorios necesarios para operar completamente el coche RC con su sistema de suspensión activa. Para resolver esto, se utilizan 4 tarjetas de expansión junto con a la Raspberry Pi para operar todo el sistema. Gracias al paquete de soporte de MATLAB y Simulink para Raspberry Pi, configurar las comunicaciones necesarias entre la placa y las tarjetas es muy simple.

La primera tarjeta de expansión es la PWM Click que maneja todas las señales PWM necesarias para los servos de suspensión, el servo de dirección y el motor sin escobillas. Todo esto se controla a través del protocolo I2C [CIRC16].

La segunda tarjeta de expansión es el ADC3 Click, que es un convertidor analógico/digital de 4 canales, que recibe todas las señales de los 4 sensores infrarrojos ya instalados utilizados para medir la altura del vehículo. Este componente también se comunica con la Raspberry a través del protocolo I2C.

La tercera tarjeta de expansión es el Counter Click, que toma la salida del sensor de fase del motor sin escobillas para contar el número de revoluciones que hace el motor, ya que esta salida coincide con una salida del codificador de cuadratura. Con este recuento, el modelo puede calcular la velocidad real del automóvil. Esta tarjeta se controla mediante el protocolo SPI [MIKE17].

La última tarjeta de expansión necesaria es el MPU IMU Click, que es una unidad de movimiento inercial (Inertial Motion Unit en inglés) [CHEN94] que tiene un giroscopio de 3 ejes [DEIM31] y un acelerómetro de 3 ejes [MOHA18], terminando los sensores

necesarios para el correcto funcionamiento del sistema de suspensión activa. Esta tarjeta también se comunica con la Raspberry Pi a través del protocolo SPI.

Un último componente electrónico es el receptor de radio control FS-A8S, que se comunica con el Raspberry Pi a través de su puerto serie.

2. Instalación de nuevo hardware

Después de verificar el hardware defectuoso, el primer paso fue desinstalar el hardware electrónico antiguo que será reemplazado. En este caso, era un antiguo microcontrolador basado en Arduino y el antiguo receptor de radio control que es incompatible con el nuevo hardware.

Luego comenzó la instalación del nuevo hardware, corrigiendo la falta de espacio en el chasis instalando una placa de expansión al chasis, creando un segundo nivel donde se instalaron la mayoría de los nuevos componentes. El resultado final se puede ver en la figura 2:

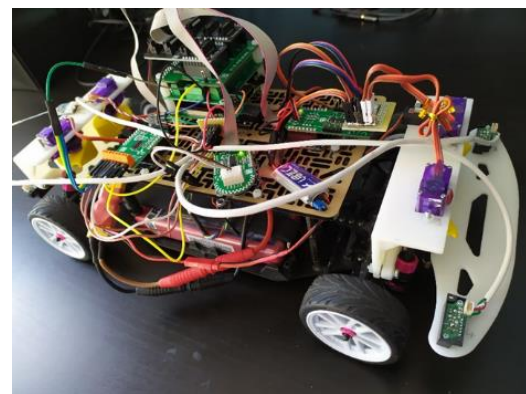


Figura 2.: coche RC terminado

3. Configuración del software

Después de instalar todo el nuevo hardware, el siguiente paso comienza con la configuración de todo el software necesario para controlar el hardware disponible en el vehículo

Primero fue configurar el receptor de radio control, en este caso un receptor FS-A8S, que utiliza el protocolo IBUS a través del puerto serie de la Raspberry Pi. Varios proyectos de drones utilizan este protocolo de comunicación, por lo que un controlador

estaba disponible y con algunos cambios, el componente se implementó con éxito.

El siguiente componente por configurar fue la tarjeta de PWM, que controlará todos los motores instalados en el automóvil. Esto también fue simple, ya que otro proyecto también había utilizado esta tarjeta, aunque se necesitaban más cambios para que fuera completamente funcional. Los cambios más importantes consistieron en asignar correctamente cada canal a cada motor y establecer los límites de seguridad del servomotor, ya que se consideró que sería más seguro establecer los límites en el código del módulo en lugar del código de control principal.

El siguiente componente configurado fue la tarjeta ADC3 Click. Esta requirió que el controlador se programará desde cero. Primero se realizó un script de configuración MATLAB .m para inicializar toda la configuración inicial de la tarjeta, así como cargar las direcciones I2C requeridas en la memoria. Luego se creó un modelo Simulink con 2 partes principales: una fase de inicialización que supervisa la inicialización adecuada de la tarjeta y una fase principal, habilitada una vez que se completa la fase anterior, que maneja el bucle de operación principal del ADC que consiste en medir secuencialmente la salida de cada sensor infrarrojo, leer los datos del ADC, y procesarlos para obtener una medición en voltios. El modelo ADC también incluye el pequeño código de conversión para el sensor infrarrojo.

Luego, la siguiente tarjeta que se implementó fue el MPU IMU Click, esta también tenía un controlador funcional usado en proyectos de drones anteriores y se implementó con algunos pequeños cambios. El código también se configuró para funcionar con un Filtro de Kalman Extendido (EKF – Extended Kalman Filter en inglés) [SING18] también programado previamente, con una pequeña configuración esto también se implementó con éxito.

La placa final para implementar es la tarjeta Counter click. Esta también necesitaba ser programada desde cero y sigue el mismo procedimiento que el caso de la ADC3 Click, un archivo de script MATLAB .m con todas las

direcciones y variables necesarias, un modelo Simulink con dos fases, una para configurar la tarjeta y la otra para su operación principal. La operación principal de esta placa consiste en contar cada revolución que hace el motor, y luego, cada tiempo de muestreo, el código calcula la diferencia entre el recuento ahora y durante el último tiempo de muestreo para obtener las revoluciones por tiempo de muestreo del motor, que luego se convierte a RPM del motor y finalmente a la velocidad del vehículo en m / s.

Todos estos modelos se implementan en un modelo maestro de Simulink que se estructura en 3 categorías: HARDWARE, MONITORIZACIÓN y CONTROL. Todas estas secciones están unidas por un bus de datos que maneja todas las variables que el modelo necesita para funcionar.

La sección de hardware contiene todos los modelos relacionados con el uso de un tipo específico de hardware, todos los modelos mencionados anteriormente se incluyen aquí.

La sección de monitorización maneja todos los scopes necesarios para ver los datos de telemetría del vehículo durante la operación, transmitidos a través de los buses de datos.

Y, por último, la sección de control contiene la máquina de estado y los bloques de control utilizados. Estos se implementan como bloques de función MATLAB. La máquina de estado guía el código a través de 5 estados: arranque, calibración de los sensores, punto de operación, espera, motores bloqueados y operación. El bloque de control maneja la operación básica como transmitir comandos del receptor a los motores, así como las estrategias de control implementadas y otros códigos críticos del control como el EKF.

Este modelo maestro de Simulink se complementa con una biblioteca de scripts .m que contienen todos los datos necesarios para inicializar el modelo. Estos están dirigidos por el script de configuración principal que, cuando se ejecuta, carga todos los otros scripts en orden, configurando todo, desde la inicialización del control, hasta la creación de los buses de datos extremadamente útiles.

Una última cosa antes de la identificación de los modelos es la creación de un regulador PI

de velocidad simple, para asegurar que durante las pruebas la velocidad del automóvil no fluctúe. Este regulador se realizó simplemente configurando un PI simple, cambiando el K y el Ti entre pruebas hasta que se logró un resultado satisfactorio.

4. Identificación de los modelos del vehículo.

Para que el chasis logre estabilidad, el sistema de suspensión activa debe poder al menos controlar las siguientes variables:

- Ángulo de cabeceo del vehículo
- Ángulo de balanceo del vehículo
- La aceleración vertical (altura) del vehículo.

Por lo tanto, el proyecto necesita al menos esos modelos antes de poder diseñar el control.

Para lograr esto, se creó un conjunto de 3 pruebas. Las pruebas consistieron en utilizar una señal PRBS [JACK71] para controlar los movimientos del sistema de suspensión activa con el objetivo de tener solo un tipo de movimiento durante cada ensayo, por ejemplo, para la prueba de cabeceo, el sistema solo cambió el ángulo de cabeceo, etc. Las mediciones de la IMU se registraron para luego procesarlas utilizando la herramienta de identificación de sistemas de MATLAB para calcular una función de transferencia que describa el modelo.

Los resultados fueron los siguientes:

- El modelo de cabeceo se identificó con una coincidencia del 87.96%
- El modelo de balanceo se identificó con una coincidencia del 92.57%
- El modelo de altura no se identificó ya que su coincidencia era de alrededor del 39%.

Por lo tanto, la prueba de altura necesitaba un nuevo diseño, el trabajo comenzó a aplicar un método de estimación de mínimos cuadrados [GIOR85] cuando el proyecto se quedó sin tiempo, y como tal el modelo de altura no fue identificado ni los controles diseñados.

3. RESULTADOS

Los resultados están compuestos por el prototipo terminado y su modelo de Simulink que lo controla.

La electrónica del automóvil se ha actualizado por completo, sin inconvenientes para el sistema de suspensión activa, y aunque el automóvil es más pesado, el sistema de suspensión activa es más potente de lo necesario como para que esto no sea un problema.

El modelo Simulink creado permite el control total del vehículo en modo manual utilizando los joysticks en el transmisor para controlar manualmente los ángulos de cabeceo y balanceo, da acceso completo a todos los sensores y mediciones que toma el sistema electrónico, y finalmente el marco necesario para que un esquema de control se puede implementar fácilmente sin muchos cambios necesarios para el modelo en su conjunto.

4. CONCLUSIONES

Al final, solo se completaron 2 de los objetivos principales de este trabajo, los cuales implicaron completar el prototipo del automóvil RC y su modelo de control, de modo que el vehículo esté operativo.

El último objetivo no era estrictamente necesario ya que la operación manual muestra la funcionalidad completa del prototipo y el trabajo previo ha demostrado que este sistema de suspensión activa es efectivo para estabilizar el chasis cuando se conduce por terreno irregular.

5. REFERENCIAS

[ALEX11] C. Alexandru and P. Alexandru, "A comparative analysis between the vehicles'," INTERNATIONAL JOURNAL OF MECHANICS, vol. 5, no. 4, p. 8, 2011.

[JIMÉ16] F. d. B. Jiménez Valverde, "Diseño de un sistema de suspensión activa para un vehículo a escala," Trabajo de fin de grado, ICAI, Universidad Pontificia Comillas, Madrid, 2016.

[AZUM19] T. Azuma, "What Is Traction Control And How Does It Work? - CAR FROM

JAPAN," 25 February 2019. [Online]. Available: <https://carfromjapan.com/article/car-maintenance/traction-control-system/>. [Accessed 25 August 2019].

[MODE19] Modelflight, "What is an Electronic Speed Controller and how does it differ from brushed to brushless motors?," Modelflight, 06 February 2019. [Online]. Available: <https://www.modelflight.com.au/blog/electronic-speed-controllers>. [Accessed 25 August 2019].

[MIKE17] MIKEGRUSIN, "Serial Peripheral Interface (SPI) - Learn.sparkfun.com," SparkFun, 2017. [Online]. Available: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all>. [Accessed 25 August 2019].

[CHEN94] J.-H. Chen, S.-C. Lee and D. B. DeBra, "Gyroscope free strapdown inertial measurement," *Journal of Guidance, Control, and Dynamics*, vol. 17, no. 2, pp. 286-290, 1994.

[CHEN94] J.-H. Chen, S.-C. Lee and D. B. DeBra, "Gyroscope free strapdown inertial measurement," *Journal of Guidance, Control, and Dynamics*, vol. 17, no. 2, pp. 286-290, 1994.

[MOHA18] Z. Mohammed, I. M. Elfadel and M. Rasras, "Monolithic Multi Degree of Freedom (MDoF) Capacitive MEMS Accelerometers," *Micromachines*, vol. 9, no. 11, p. 602, 2018.

[DEIM31] R. F. Deimel, "Mechanics of the Gyroscope," *Nature*, vol. 128, no. 3225, pp. 289-289, 1931.

[JACK71] P. A. Jackson, "PRBS cross-correlation measurements by hybrid computational techniques," *The Computer Journal*, vol. 14, no. 1, pp. 49-54, 1971.

[GIOR85] A. A. Giordano and F. M. Hsu, *Least Square Estimation with Applications to Digital Signal Processing*, New York: John Wiley & Sons, Inc., 1985



Control design of an active suspension system for a scaled vehicle

Report

Author: José Esteban Rivero Ríos

Director: Juan Luis Zamora Macho

Madrid

August 2019

TABLE OF CONTENTS

Part 1: Report	1
Chapter1: Introduction	2
1.1. State of the Art	3
1.2. Work motivation.....	5
1.3. Objectives	5
1.4. Methodology / Designed Solution.....	5
1.5. Resources / Materials applied	6
Chapter 2: Description of mechanical components.....	7
2.1. RC Prototype chassis: Turnigy TD10	7
2.2. Mechanical design of the active suspension system.....	7
2.2.1. Front and rear shelves	9
2.2.2. Semi cylinder acting as the servo's arm	10
2.2.3. Actuator arms	10
2.3. Servomotors Installed.....	11
2.4. Motor, ECS Installed	12
2.5. Infrared sensors	15
2.6. Testing tracks.....	19
2.7. Pololu RP5 Expansion plate	20
Chapter 3: Electronic hardware components installation and software setup	21
3.1. RASPBERRY PI 3 B+	21
3.2. MikroE Click boards	22
3.2.1. Pi 3 Shield Click, Shuttle Click and MikroBUS Shuttle boards	22
3.2.2 ADC3 Click.....	24
3.2.3 PWM Click.....	26
3.2.4. Counter Click.....	28
3.2.5. MPU IMU Click.....	30
3.3. PI EzConnect	31
3.4. Receiver and transmitter combo.....	32
3.4.1. Flysky FS-A8S receiver & FS-I6S transmitter combo.....	32

3.5. Finished RC car.....	34
Chapter 4: Obtaining the models	36
4.1. Model Tests	36
4.1.1. Pitch test and plant calculation	37
4.1.2. Roll test and plant calculation	41
4.1.2. Roll test and plant calculation	43
Chapter 6: Conclusions	45
Chapter 7: Future developments	46
References	47
Part 2: Economic analysis	49
Chapter 1: Economic Analysis.....	50
Part 3: Annexes.....	51
Chapter 1: CONFIG_RC_CAR.m.....	52
Chapter 2: CONFIG_MCP3428.m.....	57
Chapter 3: CONFIG_LS7366R.m.....	59
Chapter 4: CONTROL UPDATE FUNCTION	60
Chapter 5: STATE MACHINE UPDATE FUNCTION	67

TABLE OF FIGURES

Figure 1.1.: Mercedes-Benz’s Active Body Control system concept.....	4
Figure 1.2.: BOSE’s bulky Linear electromagnetic motors	4
Figure 2.1.: Turnigy TD10 Kit and specs.	7
Figure 2.2.: Common active suspension setup.....	8
Figure 2.3.: Active suspension system installed on the Prototype.	8
Figure 2.4.: Front shelf render.....	9
Figure 2.5.: Rear shelf render.	9
Figure 2.6.: Servo arm render.....	10
Figure 2.7.: Actuator arm render.....	11
Figure 2.8.: Turnigy 380mg Micro servomotor.	12
Figure 2.9.: Turnigy XK3650 3900KV motor	12
Figure 2.10.: Sensored brushless wire specifications.....	13
Figure 2.11.: Turnigy Trackstar 1/10th 80A Turbo Sensored ESC with specs	13
Figure 2.12.: Turnigy TrackStar Turbo and Waterproof ESC Programming Box with available options.....	14
Figure 2.13.: Sharp GP2Y0A51SK0F infrared sensor	15
Figure 2.14.: Infrared sensor measurement timing	15
Figure 2.15.: Voltage output vs Distance in cm from the sensor	16
Figure 2.16.: Look up table setup.	17
Figure 2.17.: Infrared sensor’s Simulink model.....	17
Figure 2.18.: Front bumper render.....	18
Figure 2.19.: The high bumps testing tracks aligned, for regular bumpy terrain.	19
Figure 2.20.: The high bumps testing tracks misaligned, for Irregular bumpy terrain. .	19
Figure 2.21.: Pololu RP5/Rover 5 Expansion Plate.	20
Figure 3.1.: Raspberry Pi 3 B+.....	21
Figure 3.2.: PI 3 Shield click	22
Figure 3.3.: Shuttle click on the left, mikroBUS Shuttle to the right.	23
Figure 3.4.: ADC 3 Click board	24
Figure 3.7.: ADC configuration model	25
Figure 3.6.: ADC read model.....	25

Figure 3.7.: PWM Click board.	26
Figure 3.8.: PWM signal diagram.....	27
Figure 3.9.: PWM control model	27
Figure 3.10.: Counter Click board	28
Figure 3.11.: Motor phase sensors	28
Figure 3.12.: Model for initializing the counter board.	29
Figure 3.13.: Model for Counter processing.....	30
Figure 3.14.: MPU IMU click board.....	30
Figure 3.15.: Pi-EzConnect shield	31
Figure 3.16.: FS-A8S receiver	32
Figure 3.17.: PPM frame scheme	33
Figure 3.18.: FS-I6S transmitter	33
Figure 3.19.: Finished RC car	34
Figure 4.1.: Simulink model used to test for the necessary transfer functions	37
Figure 4.2.: Pitch test results, calibration time section omitted	38
Figure 4.3.: Pitch data import.....	38
Figure 4.4.: Prepared Pitch data	39
Figure 4.5.: Identification toolbox	39
Figure 4.6.: Process models settings and results for pitch	40
Figure 4.7.: Simulated Pitch output vs Measured Pitch output.	40
Figure 4.8.: Roll test results, calibration time section omitted	41
Figure 4.9.: Prepared Roll data	41
Figure 4.10.: Process models settings and results for roll.....	42
Figure 4.11.: Simulated Roll output vs Measured Roll output.	42
Figure 4.12.: Height test results, calibration time section omitted	43
Figure 4.13.: Prepared height data.....	43
Figure 4.14.: Process models settings and results for height.....	44

PART 1: REPORT

CHAPTER1: INTRODUCTION

Since the creation of the first internal combustion engine powered vehicle, and the subsequent creation of the automotive industry, Engineers have always strived to design vehicles to be, faster, safer, more efficient and comfortable over the years. Driven by this wish to create the best transportation experience, engineers have added new systems and features to vehicles that there is no comparison between them and the first models in 1885. The latest of these new features is autonomous driving, which as its name implies, looks to replace the driver with specialized hardware and software to allow its passengers to relax, or get some work done while the vehicle takes them to their next destination.

While this concept is not a new one, in the last decade there have been enormous advances in the field, spearheaded by Google in 2015-2016, today some high-end, luxury models have some type of autonomous driving features, with more complete prototypes on the way. The most complete experience belonging to tesla, which in early 2019 upgraded their autopilot system to a near full-self driving system although still requiring the driver to have their eyes on the road due to safety regulations. With a possible new paradigm shift in the automotive sector, some companies like ClearMotion have started to focus on the main subject of this work, active suspension systems, with the idea to complement the rise of self-driving cars. Once driving is out of the equation, most passengers will wish to partake on other activities during their commute, such as work, reading, or maybe simply watch the scenery pass by, and as many would agree, having bumps, potholes or any disturbances to a smooth drive can disturb any of these activities highlighting the significance of these systems.

There are 3 types of suspension: passive, semi-active, and active. Traditional suspension systems are passive in nature, they receive the energy from any bumps and other disturbances, in order to dampen them reducing their effect on the chassis of the vehicle but not being able to fully eliminate them. An intermediate point would be semi-active system that changes the dampening depending on the road conditions as picked by an array of sensors, with the goal of adjusting the right amount of dampening to every situation. Finally, Active suspension systems have actuator allowing them to fight against any changes in the pitch, roll, or elevation of the chassis reaching a higher degree of stability, unlike passive systems which can only absorb energy and redirect it, active systems can also add more thanks to their actuators allowing them to counteract their effects [ALEX11], [SHIR19].

This project will attempt to upgrade the electronics of an RC car of a previous thesis [JIMÉ16], ensuring correct functionality current installed active suspension system with the new hardware, creating a new Simulink model to take full advantage of the newly added hardware, and applying a simple control strategy to finish.

1.1. State of the Art

Ways to increase the stability of cars have always been in the forefront of engineers' minds, whether to increase performance on the racing track, or in unforgiving irregular terrain, or to simply increase safety and control of their vehicles. Once electronics started to shrink and it became viable for them to be embedded in multiple systems, the automotive sector was no different. Perhaps one of the more common types of ESC (Electronic Stability Control) in today's cars is the Traction Control System or TCS [AZUM19], which detects possible losses in traction and automatically applies brakes or cuts engine power to prevent them, increasing stability and safety during the ride.

Active suspension systems operate differently, instead of changing the user's input they actively change parts of the suspension to counter any unwanted force or disturbances that may occur during driving. One of the first examples of an active suspension system in consumer vehicles came in the form of the SC-CAR (or Citroën's Active Roll control system in English) in Citroën's Xantia Activa model in 1994, and consisted of an active Anti-Roll bar using a hydraulic system that stiffens or loosens the Anti-Roll bar depending on instructions from the ESC, fighting the roll of the car during turns increasing stability, but only for the roll of the vehicle.

The first complete active suspension system was introduced in 1999 with Mercedes-Benz's Active Body Control (ABC) introduced in their Mercedes-Benz CL-Class C215. Their ABC system consisted of telescopic hydraulic actuators that increase or lower the height of each wheel depending on the wide array of sensors in wide array of sensors on the vehicle, keeping the vehicle level. The system also allowed for height adjustable suspension and self-leveling suspension with user selectable profiles to adapt the suspension depending on the requirements of the drive and the comfort of the user, allowing for better fuel consumption and handling thanks to a better adjusted aerodynamic profile. Later, Mercedes-Benz would introduce the concept of PRE-SCAN suspension, taking the system of Active Body Control and adding LIDAR sensors to anticipate bumps or potholes on the road, allowing the vehicle to pre-emptively adjust the suspension instead of simply reacting to it once it reaches it.

Mercedes-Benz F 700 PRE-SCAN® system control

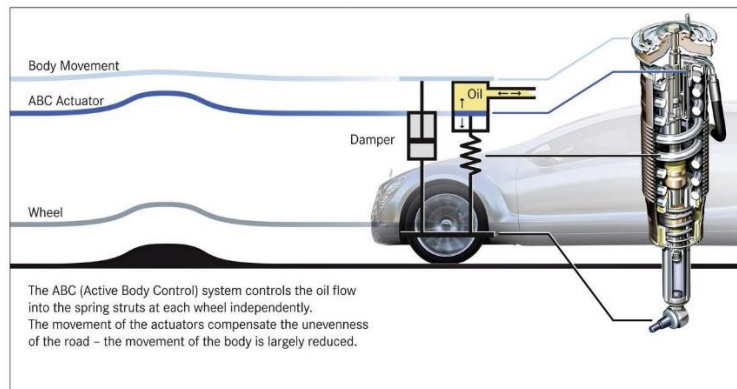


Figure 1.1.: Mercedes-Benz's Active Body Control system concept

The PRE-SCAN + ABC concept would later evolve into the Magic Body Control system first seen in Mercedes-Benz's S-Class (W222) model released in 2013. Using a stereo camera instead of LIDAR technology, the Magic body control system can scan up to 15m of the road ahead of the vehicle and adjusts the system accordingly.

Mercedes-Benz's hydraulic actuators are not the only active suspension system around. BOSE has been working on Project Sound which had its first unveiling in 2004 installed in a Lexus LS 400, their system replaces the typical shock absorbers with a linear electromagnetic motors (LEM), thanks to their speed, a sophisticated sensor system to scan the road ahead and a sophisticated control system BOSE's system makes any road bumps, tilting due to braking or acceleration virtually unnoticeable, to the point of looking unreal in their video showcasing the system. Unfortunately, BOSE was unable to reach production status for this system due to the economic recession of 2008 and coupled with the fact that their LEM were bulky and expensive to make.



Figure 1.2.: BOSE's bulky Linear electromagnetic motors

ClearMotion would later buy Project Sound, combining BOSE's road sensing and control software with their own active valve dampeners, replacing BOSE's linear

electromagnetics motors. Their digital chassis system is set to reach mass production in 2019 With the goal of targeting the autonomous vehicle market.

1.2. Work motivation

The main motivation for this project is to have a working active suspension system for the advance control classes in the post-grad programs of the university, creating more interest in a field which shows a lot of promise in the coming years, especially for consumers of autonomous vehicles.

This project will take the active suspension prototype made in a previous thesis [JIMÉ16] and update its electronic hardware with the goal of allowing more sophisticated control schemes to be implemented on the prototype, and greater ease of use of the prototype in future investigations.

1.3. Objectives

- Update the electronic control hardware installed in the prototype, without affecting the active suspension system installed.
- Setup a Simulink model that allows the newly installed hardware to operate the entirety of the RC car effectively allowing for more sophisticated control schemes to be implemented.
- Setup a basic control system to test the active suspension system in the RC car with the new hardware.

1.4. Methodology / Designed Solution

First to remove the old electronic hardware installed onto the prototype car, mainly the old Ardupilot microprocessor and the Walkera receiver. Then all mechanical components of the prototype are checked for faults, ensuring that all mechanical aspects of the prototype are working properly.

Then an expansion plate is installed to provide more space for the new electronic components to be installed, alongside all the necessary cables. Each new component is then tested and calibrated.

Then the real models are obtained by running the appropriate tests and using MATLAB's system identification toolbox. Once the models are identified, appropriate control strategies will be designed to achieve a smooth operation of the active suspension system, leading to a stabilized chassis for the prototype.

Once the controls are in place, a final round of tests are made in order to collect the results of the project and reach a conclusion.

1.5. Resources / Materials applied

- Software
 - MATLAB 2018b
 - Simulink
 - PuTTY
- Hardware
 - Computer with the previously mentioned software
 - Turnigy TD10 RC touring Car chassis, with Turnigy motor and ESC and installed with the previous active suspension system design and infrared sensors
 - LiPo Battery 2-cell battery with voltage regulator for Raspberry PI
 - FlySky Radio control transmitter with corresponding receptor
 - Raspberry PI with a 16gb SD card with the MathWorks Raspbian OS image
 - PI-Ez Connect Shield board
 - PI click Shield board
 - ADC click board
 - Counter Click board
 - PWM click board
 - IMU click Board
 - Pololu Expansion plate

CHAPTER 2: DESCRIPTION OF MECHANICAL COMPONENTS

2.1. RC Prototype chassis: Turnigy TD10

The Radio control car kit chosen for the project is the Turnigy TD10 touring car Kit, a chassis designed for hobbyists that wish to set up their vehicle for a fraction of the cost of a new complete RC car setup. The kit consists of the chassis for the car complete with a transmission and suspension system and space for the battery, brushless motor, ESC, direction servo, and a microprocessor to control it all.



Specs:

Gear Ratio: **1.9:1 (38T/20T)**

Wheelbase: **257mm**

Tread (F/R): **163 F/R**

Overall Length: **343mm (without body and foam bumper)**

Overall Width: **188mm**

Figure 2.1.: Turnigy TD10 Kit and specs.

This kit was also chosen for its sports design over a more all-terrain type vehicle, as those vehicles have suspension better suited for irregular terrain and the changes to the suspension would not be as noticeable. It also has the advantage of having a suspension system like a normal car, making the project more appropriate for a consumer release.

2.2. Mechanical design of the active suspension system

As mentioned in the previous thesis on this project, the active suspension design is a bit unconventional. Typical active suspension setups for RC cars place the servos parallel to the floor the servos taking the weight of the car radially. The setup then uses small plastic pieces attached to the servo to move the shock coils up and down, changing the wheel height to adapt to the terrain, as shown in the image below [reference to image]



Figure 2.2.: Common active suspension setup.

Unfortunately, a stress analysis of this setup shows that if the car were to take a significant bump or takes multiple bumps at a decent cruise speed the servo's gears will take most of the shock, if they do not break outright, the repetitive stress would fatigue the parts enough for them to be more prone to failure in the future. That's not mentioning the fact that the servo arms could begin to warp after enough abuse from irregular terrain any decent speed, this is especially important this time around as the new hardware installed onto the car increases its weight enough to make this a significant problem. This design is only useful for smooth tracks designed for racing, looking to increase the stability of the vehicle during turns and as such does not consider these drawbacks too significant.

To solve this, the designed proposed in the previous report uses a semi cylinder design which allows the servomotor axis to be perpendicular to the ground, absorbing any stress axially instead of radially, protecting the servo's gears as the stress passes onto the chassis which is more than capable of absorbing these quick shocks. The semi cylinder parts are also larger and work axially, avoiding any warping during high periods of stress.



Figure 2.3.: Active suspension system installed on the Prototype.

Thus, this project will keep this design moving forward after explaining the decisions taken in their design.

2.2.1. FRONT AND REAR SHELVES

The shelves act as holders for the suspension servomotor and the actuator arms, allowing the servomotors to rest in the correct position and the actuator arms to move freely through a plain bearing and 2 washers. They are designed to be affixed to the vehicle using 4 metric 3 screws just like the suspension supports of the vehicles that were removed. Both shelves are designed to fit perfectly with the rest of the kit, avoiding any collisions with any moving parts during operation. The main difference between each shelf is the extra slot to accommodate the front differential. The following images are renders of both shelves:

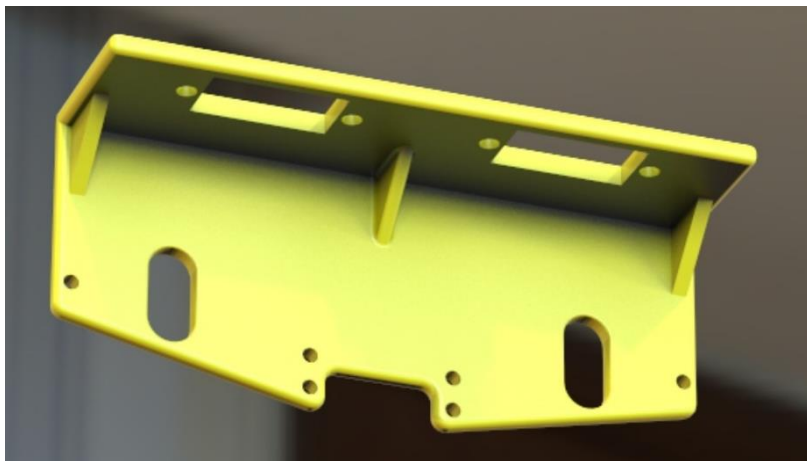


Figure 2.4.: Front shelf render.

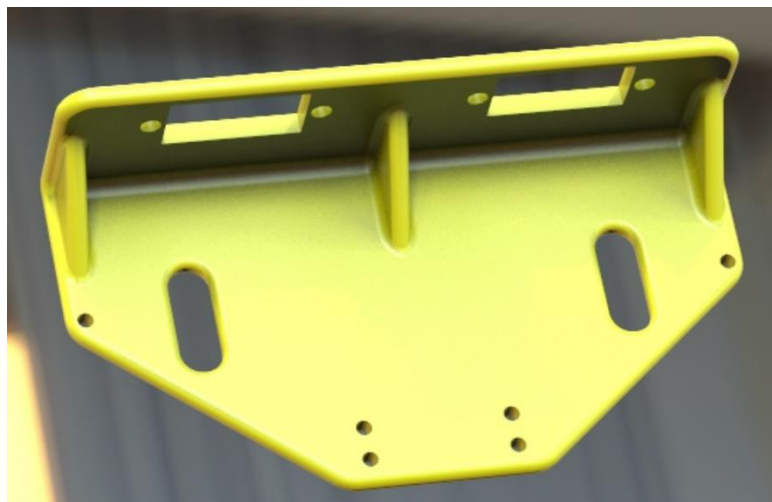


Figure 2.5.: Rear shelf render.

2.2.2. SEMI CYLINDER ACTING AS THE SERVO'S ARM

As mentioned before, the servo arms need redesigning to be able to withstand the irregular terrain the vehicle might encounter as well as lowering or raising the wheels as needed. This was accomplished by designing cylinders cut by an oblique plane in such a way that the surface allows the entire 180° of rotation from the servomotor to be transformed into a near vertical linear translation, if an appropriate actuator arm is used. With this arm, as the servo arm rotates the actuator arms gets pushed down, shifting the position of the wheel downwards as well and lastly raising the chassis which is the end goal. Clearly if the rotation is in the opposite direction the chassis lowers as the wheels are “raised” upwards due to the actuator arm not being pushed as much.

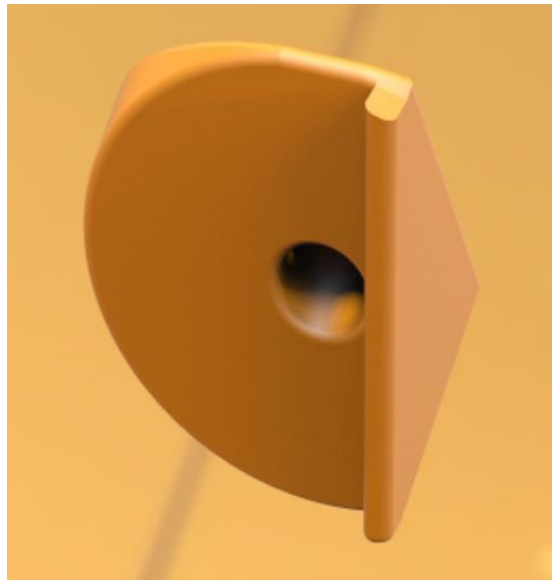


Figure 2.6.: Servo arm render.

Evidently the second servo arm is the mirror image of the first, the suspension uses 4 of these arms, one for each wheel.

2.2.3. ACTUATOR ARMS

This is the last piece in the suspension system designed for this project, it consists of an 3d printed arm that is attached in one end to the shelf using 2 washers and a plain bearing to allow rotation around that axis. The one end of the shock coil is screwed into the middle of the arm and lastly the arm ends on a point where it makes contact with

the servo arm, working as described in its section of this document, and with this last piece in each wheel the car has a working active suspension.



Figure 2.7.: Actuator arm render.

Although that initial design had some kinks. The main problem was the fact that the 3D printed point would wear out the servo arms and itself very quickly as 3D materials tend to be rough creating too much friction between the pieces, this was solved by designing and adding a bronze point cover to the actuator arm reducing the friction between the pieces and increasing the durability of both pieces.

2.3. Servomotors Installed

The servomotors chosen for the suspension system are the Turnigy 380mg Micro servomotors. These micro sized servos were chosen for their small size, low voltage requirements (as the previous micro controller was only capable of 5V), and finally for their sturdy construction, the servos make use of metal and carbon gears making them sturdier than standard servos. This is a bit unnecessary as the new suspension system reduces the stress placed on the servos but chosen anyways for increased reliability. These servos are also more than capable of lifting the car up and down thanks to their 4.2kg/cm stall torque when powered at 6V, allowing them to easily act as the active suspension for the vehicle

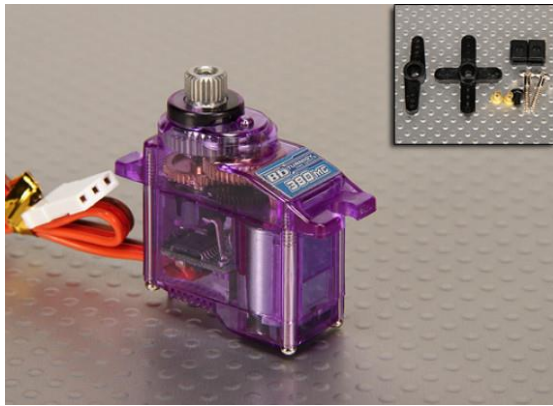
**Specs:**Model: **Turnigy 380MG**Operating Voltage: **4.8V / 6.0V**Operating Current: **200mA / 250mA**Operating Speed at 4.8V: **0.15sec/60° at no load**Operating Speed at 6.0V: **0.14sec/60° at no load**Stall Torque at 4.8V: **3.6kg/cm**Stall Torque at 6.0V: **4.2kg/cm**Size: **26 x 13 x 26mm**Weight: **16.3g**Spline Count: **25**Gear Material: **Metal**

Figure 2.8.: Turnigy 380mg Micro servomotor.

2.4. Motor, ECS Installed

The motor installed on the RC car is the one of the manufacture's recommended motors for the TD10 kit, a Turnigy XK3650 3900KV brushless dc inrunner Sensored motor. The 3900KV does not stand for Kilovolts but instead denotes the RPM the motor gives per volt; thus, this motor specifically gives 3900 RPM for each volt given to it, which is more than capable for the project.



Figure 2.9.: Turnigy XK3650 3900KV motor

A 3-phase motor like this one requires an ESC (Electronic Speed Controller) to function properly as the 3 phases need to be managed properly for the motor to work at all [MODE19]. The motor also has an output for ESC for telemetry, the sensor connection transmits the state of the phases in the motor as well as it's temperature for the ESC to

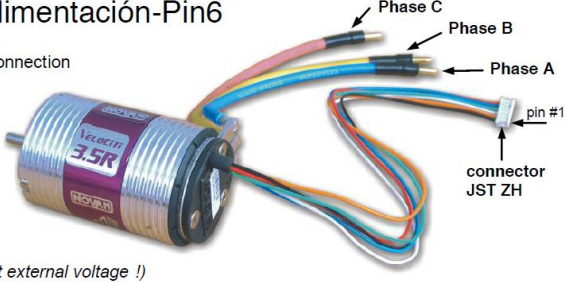
cut off the motor when it reaches high temperatures. This can be seen in the following images with the wiring specifics that apply to this motor:

SENSOR MOTORS:
Sensor motor according to EFRA specification:
 - must have 6-pin JST ZH connector model ZHR-6 or equivalent, marked as SZH-002TP0.5 26-28 awg. for sensors and heat sensor connection

Pins specification of this connector:
Pin #1 – black wire, ground potential (minus)
Pin #2 – orange wire, sensor phase C
Pin #3 – white wire, sensor phase B
Pin #4 – green wire, sensor phase A
Pin #5 – blue wire, motor temperature sensing, 10 kΩ NTC
 (other end of sensor is on ground potential, pin #1)
Pin #6 – red wire, sensors feeding, +5.0 V ± 10%.
 (supply voltage for sensors provide controller, don't connect external voltage !)

- power wires are marked A, B, C – connect to phases of controller, with the same name.
 A for phase A
 B for phase B
 C for phase C

Alimentación-Pin6



Example: Motor by NOVAK, Velocity 3.5R Brushless Motor

Figure 2.10.: Sensored brushless wire specifications

The ESC chosen is the recommended one for this motor a Turnigy Trackstar 1/10th 80A Turbo Sensored ESC, which is more than enough to power the car and the all its installed servomotors thanks to its built in BEC (Battery Eliminator circuit), which as its name implies, is designed to replace the need of separate batteries in RC vehicles in case you need different voltage requirements. Unfortunately, the processor driving the program on the vehicle requires 5V and therefore another voltage regulator is necessary to power the rest of the components installed. The following image has the details on the ESC's Stats.



Spec:

- Max Voltage: **2s Lipo / 6 cell Nimh**
- Motor limit: **10.5T**
- Resistance: **0.0004 ohm**
- Operating current: **80A**
- Burst Current: **380A**
- Bec Current: **3A**
- Bec Voltage: **6V**
- Weight: **45g (without cables)**

Figure 2.11.: Turnigy Trackstar 1/10th 80A Turbo Sensored ESC with specs

One thing to note, this ESC requires a separate programmer in order to customize specialized options such as turbo timing and different operation modes. The programming box used is the Turnigy TrackStar Turbo and Waterproof ESC Programming Box with the following options for this ESC:



Programming Options (Turbo ESC):

- Operation Mode
- Initial Brake
- Drag Brake
- Brake Strength
- Punch Profile
- Neutral Dead Band
- Boost Timing
- Turbo Slope
- Turbo Timing
- Boost Timing RPM
- Turbo Delay
- Boost Timing ACC
- Reverse Speed
- Drive Frequency
- Brake Frequency
- Temperature Set
- Restore Default

Figure 2.12.: Turnigy TrackStar Turbo and Waterproof ESC Programming Box with available options.

2.5. Infrared sensors



Figure 2.13.: Sharp GP2Y0A51SK0F infrared sensor

For the infrared sensors needed in the vehicle, the project uses 4 Sharp GP2Y0A51SK0F infrared sensors as they are easy to use, simply requiring power and an ADC to see the resulting value. The infrared once powered takes around 21ms before its output represents the distance measured, taking around 16.5ms to update the measurement as shown in the following image:

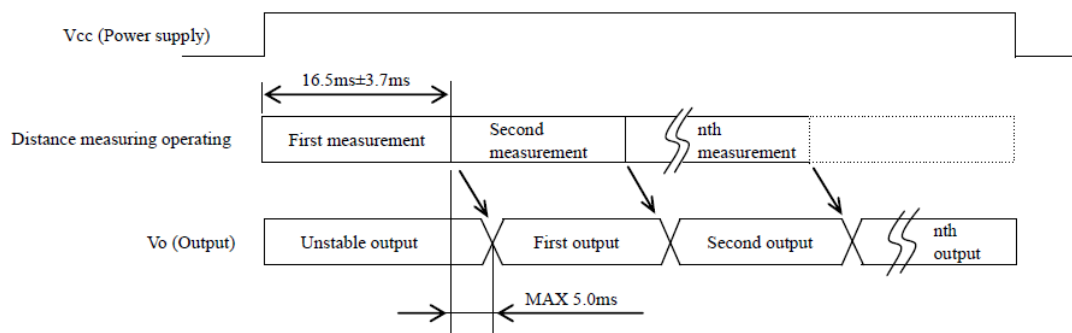


Figure 2.14.: Infrared sensor measurement timing

Once the measurement is taken using an ADC, it can then be converted into the needed distance by simply looking up the value to the following diagram:

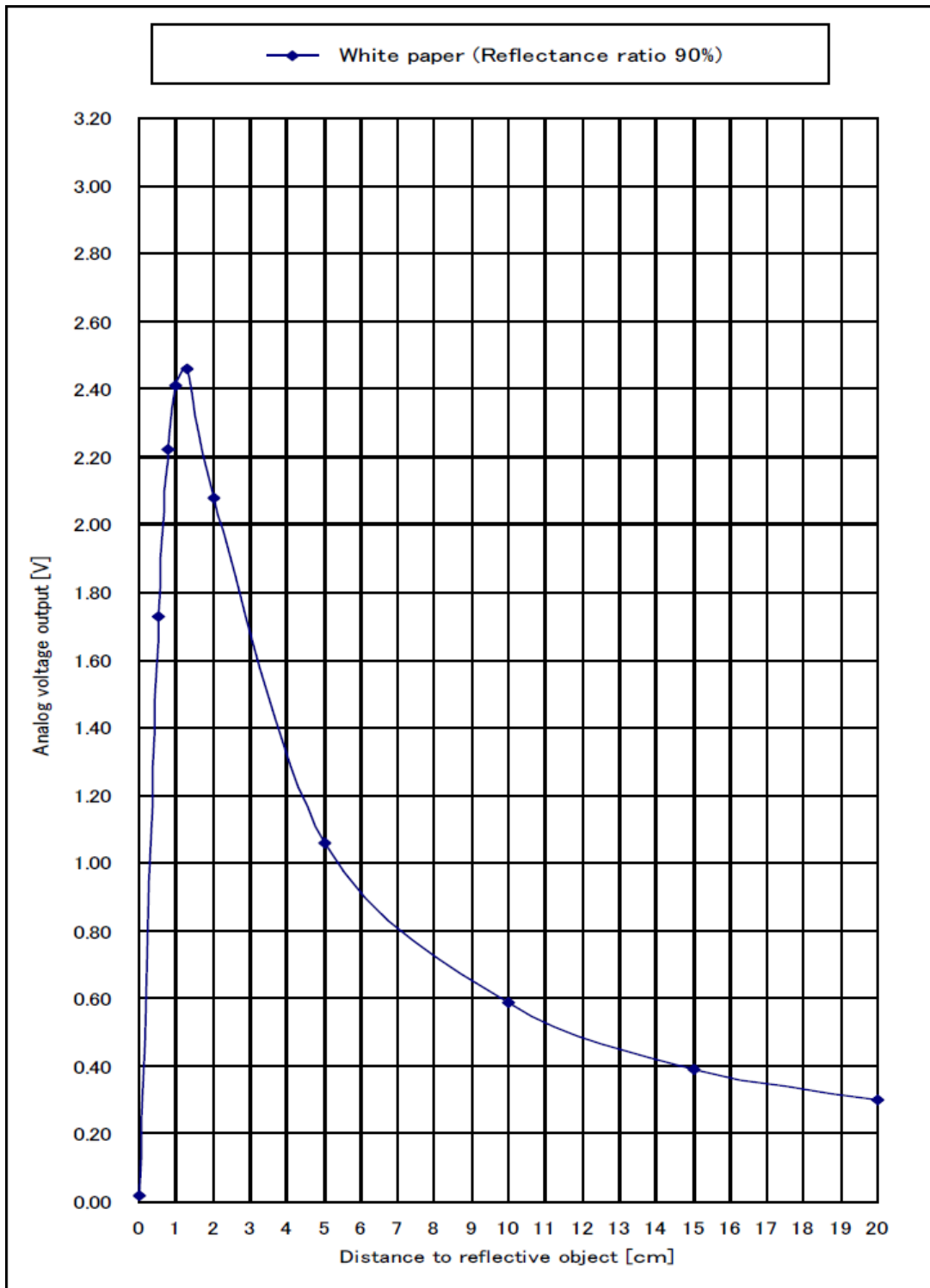


Figure 2.15.: Voltage output vs Distance in cm from the sensor

This diagram is programmed into the Simulink model using the lookup table block, in which we can add the values from the diagram above and allow the model to compare the voltage value received from the ADC and obtain the needed measurements. The lookup table set up can be seen in the following image

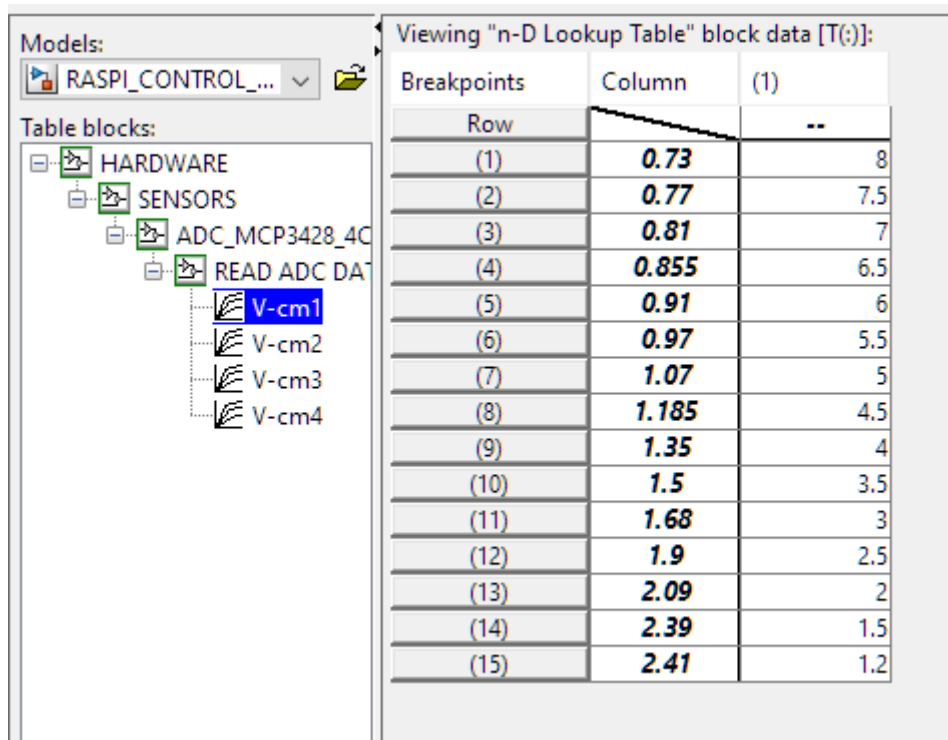


Figure 2.16.: Look up table setup.

The model used in the Simulink system is the following:

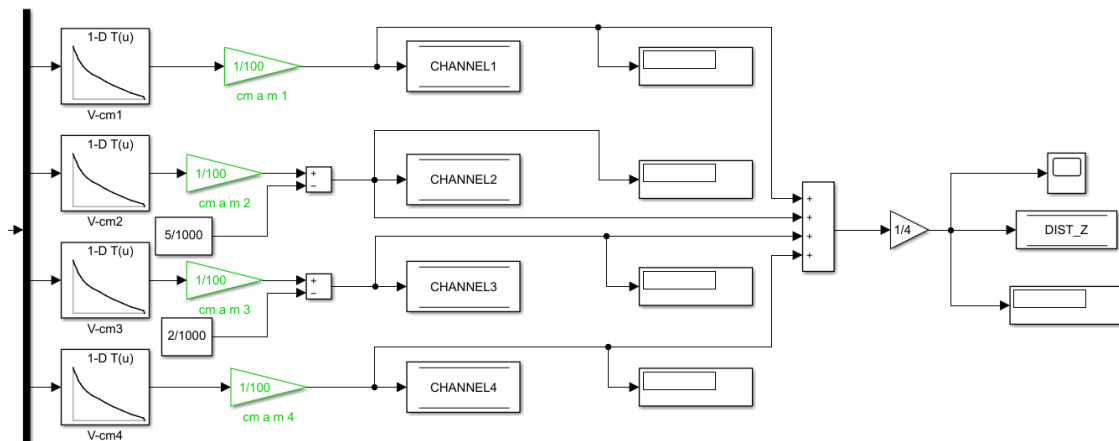


Figure 2.17.: Infrared sensor's Simulink model.

The model uses the aforementioned lookup tables and then simply converts the measurement from cm to m and then saves them into the control BUS for use elsewhere in the model, the infrared sensors on the back of the RC car are at slightly different heights than to front sensors, hence the added 5 and 2mm added to the measurements in the model.

To mount the front sensors, a front bumper was designed to hold the 2 sensors on the front of the vehicles, a render can be seen in the following image:



Figure 2.18.: Front bumper render

As seen above, the piece has 2 slots for the sensors just Infront of the wheels for the active suspension to anticipate changes in the terrain. The rear sensors can just be attached to the rear shelf using double sided tape in a secure and accurate manner, so no new pieces are required, although they are slightly higher than their front brothers which is addressed in the code described before.

2.6. Testing tracks

To test the vehicle 2 sets of testing tracks were made, one set made with a height 0.8cm between the ground and the top of the bump (about 30% of the wheel height), and the other is 0.5cm height. Each set is made of 2 sheets of aluminum, each with the same bumps so that we can set up the track for regular bumps or irregular bumps, depending if the tracks are aligned or not when setup. The tracks were made using a bending machine [reference?] having the slopes be 3cm long and the tops of the bumps 6cm long. The high bumps testing track can be seen below in both regular and irregular terrain configurations:

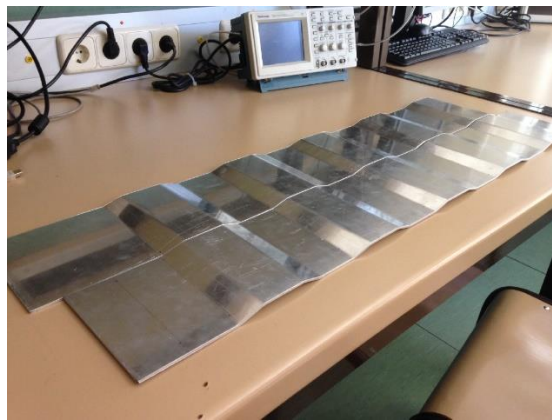


Figure 2.19.: The high bumps testing tracks aligned, for regular bumpy terrain.

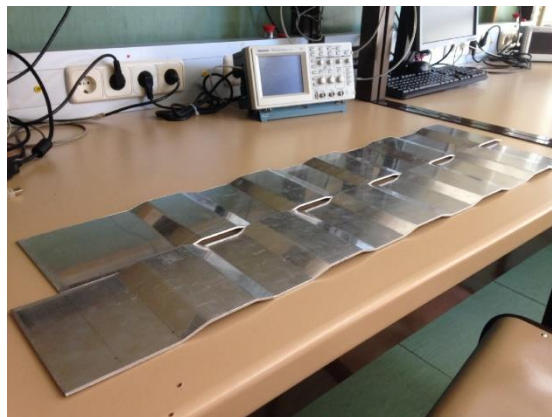


Figure 2.20.: The high bumps testing tracks misaligned, for Irregular bumpy terrain.

2.7. Pololu RP5 Expansion plate

Before any of the new electronic components can be installed, the prototype needs more space to support them. To solve this, an expansion plate is installed creating a second level to the vehicle where all the rest of the electronics components can be installed onto. The plate used is the Pololu RP5/Rover 5 Expansion Plate RRC07B

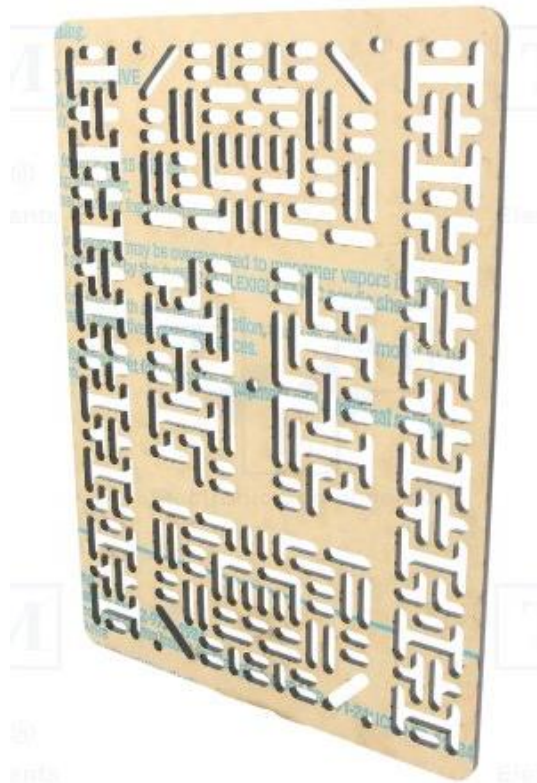


Figure 2.21.: Pololu RP5/Rover 5 Expansion Plate.

This plate is drilled with 3mm slots allowing easy installation of most electronic boards with 3mm screws and nuts, allowing the installation of new hardware without getting in the way of the locomotive parts of the vehicle as well as its active suspension system.

CHAPTER 3: ELECTRONIC HARDWARE

COMPONENTS INSTALLATION AND SOFTWARE

SETUP

3.1. RASPBERRY PI 3 B+



Figure 3.1.: Raspberry Pi 3 B+

The board chosen to control all the new and old components used in the RC Car is the Raspberry Pi 3 B+, that at the time of the project start was the latest and most capable product of the Raspberry Pi Foundation.

This microcomputer is one of the more affordable options of its class, capable as well as being extremely easy to use with Simulink as it has an official support toolbox allowing access to all the GPIO (General Purpose I/O) pins and communications capabilities of the Raspberry Pi 3 B+.

The main advantage of using this microcomputer instead of the specialized ArduPilot microprocessor used previously, is the computing power, the 1.4GHz 64-bit quad-core processor dwarfs the ATmega2560 used in the Ardupilot, allowing for more complex control strategies to be implemented.

However this comes at a tradeoff, a microcomputer is designed to be a cheap computer replacement and consequently, it does not have some of the integrated peripherals our previous microprocessor had, especially since the Ardupilot was designed for RC control: it has no analog inputs therefore it cannot use the infrared sensors already installed, nor does it have an integrated IMU for all the necessary angle measurements needed for

the control strategies, it also does not have the same facility in handling all the PWM channel inputs from the installed receiver and all the PWM outputs. Thus, it is necessary to include extra boards to extend the capabilities of the Raspberry. These extra boards communicate with the Raspberry through the integrated communication options of the Raspberry: serial, I2C [CIRC16], and SPI [MIKE17], and after an initial configuration these peripherals work just as well as the previous integrated solutions.

The main reason for swapping out the old microprocessor is the official MATLAB/Simulink Support Package for Raspberry PI hardware, which includes a customized Raspbian OS image designed to work alongside MATLAB and Simulink allowing for Simulink models and code to be run in external mode, allowing changes to parameters during code execution and monitoring of all the sensors and internal parameters to simplify code creation as all data can be seen in real time, greatly simplifying the programming process.

3.2. MikroE Click boards

The family of expansion boards chosen to increase the Raspberry PI's capabilities is MikroE's family of Click boards [add reference] which use the mikroBUS Socket standard [reference] to create a modular set of boards that can be easily added and swapped out as long as they use the mikroBUS socket.

3.2.1. PI 3 SHIELD CLICK, SHUTTLE CLICK AND MIKROBUS SHUTTLE BOARDS

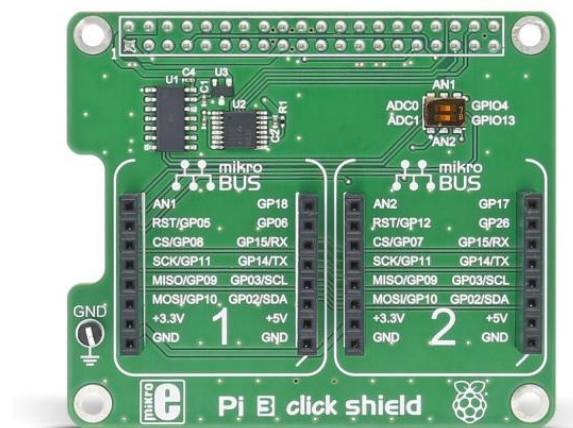


Figure 3.2.: PI 3 Shield click

This shield board is the basic requirement to make the Raspberry Pi compatible with any of the click boards, as it provides 2 mikroBUS sockets and connects them onto the Raspberry Pi's 40-pin header providing everything needed for any mikroBUS compatible boards to function, by simply providing direct connections between the Raspberry's GPIO pins and the mikroBUS sockets in order to connect the click boards directly to the raspberry.

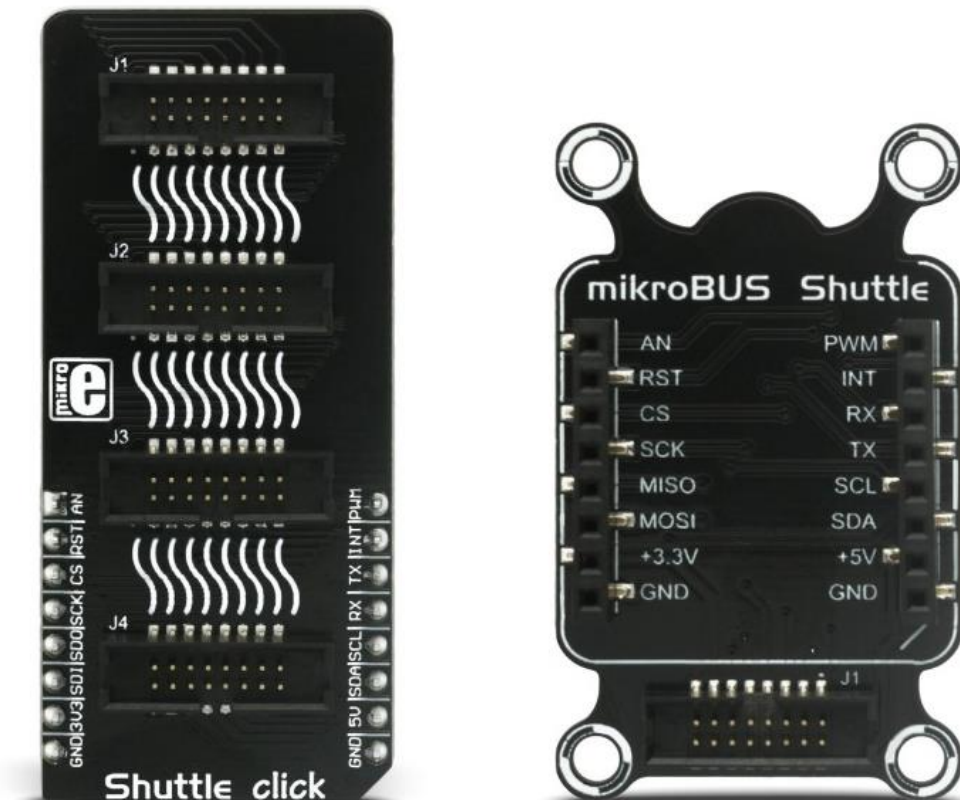


Figure 3.3.: Shuttle click on the left, mikroBUS Shuttle to the right.

This project will use 4 different click boards; thus, the project needs more sockets to accommodate them all. For this the project will use the Shuttle click socket expansion board which has 4 standard 16-pin connectors to add up to 4 mikroBUS Shuttles which are satellite boards where click boards can be plugged in using 16-pin flat Ribbon cables. With these 2 boards we can expand the number of sockets available to the Raspberry Pi, the project specifically requires 4 mikroBUS Shuttles and 2 Shuttle Clicks, due to some incompatibilities between the click boards used on the RC Car.

3.2.2 ADC3 Click

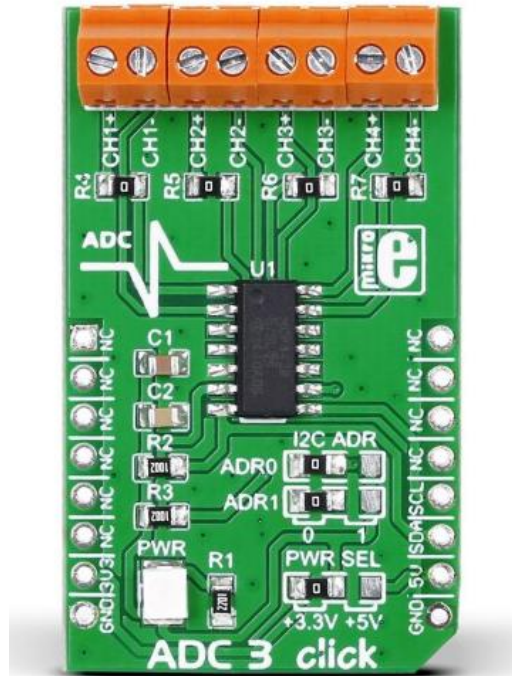


Figure 3.4.: ADC 3 Click board

The ADC3 Click board uses a MCP3428 ADC wired to the mikroBUS Socket and to 8 solderless connectors. The MCP3428 is a 16-bit, 4-channel differential Analog-to-Digital converter that communicates through I2C protocol which makes it a breeze to use with the Raspberry PI installed. The project uses this ADC to read the analog values from the infrared distance sensors installed on the RC Car, as the ADC uses differential input each IR sensor's signal is connected to the positive terminal while a ground connection is used on the negative terminal. This ADC has an internal voltage reference of 2.048V and consequently it cannot read any voltage signal larger than the [Vref], luckily this voltage is only reached when the distance is smaller than around 2 cm, which during normal operation the sensors will not reach those values.

The MCP3428 is configured to use the one-shot conversion mode and, 12bits for the sample resolution size, samples per second are irrelevant in one-shot conversion mode, as the ADC only makes a measurement when prompted by the Raspberry, nevertheless the setting is set to 240SPS as a value is needed for the configuration command. The MCP3428 in one-shot conversion works by taking a measurement as long as a Ready bit in its configuration register is set to 0, when set to one manually it retakes the measurement, this allows for the user to take measurements when needed and conserve some power. As the project needs to measure all 4 channels, instead of just

changing the Ready bit, the entire configuration byte is sent in order to reset the ready bit and change the channel to measure with the goal of measuring each channel sequentially.

To command and receive the converted values from the ADC the following Simulink model was used:

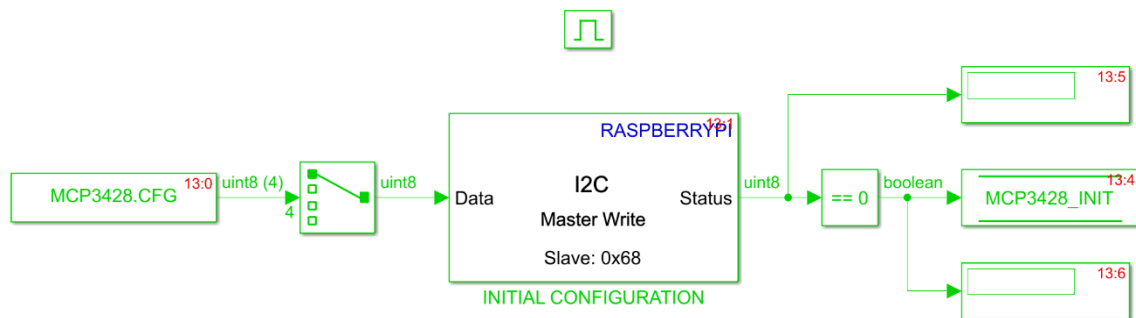


Figure 3.7.: ADC configuration model

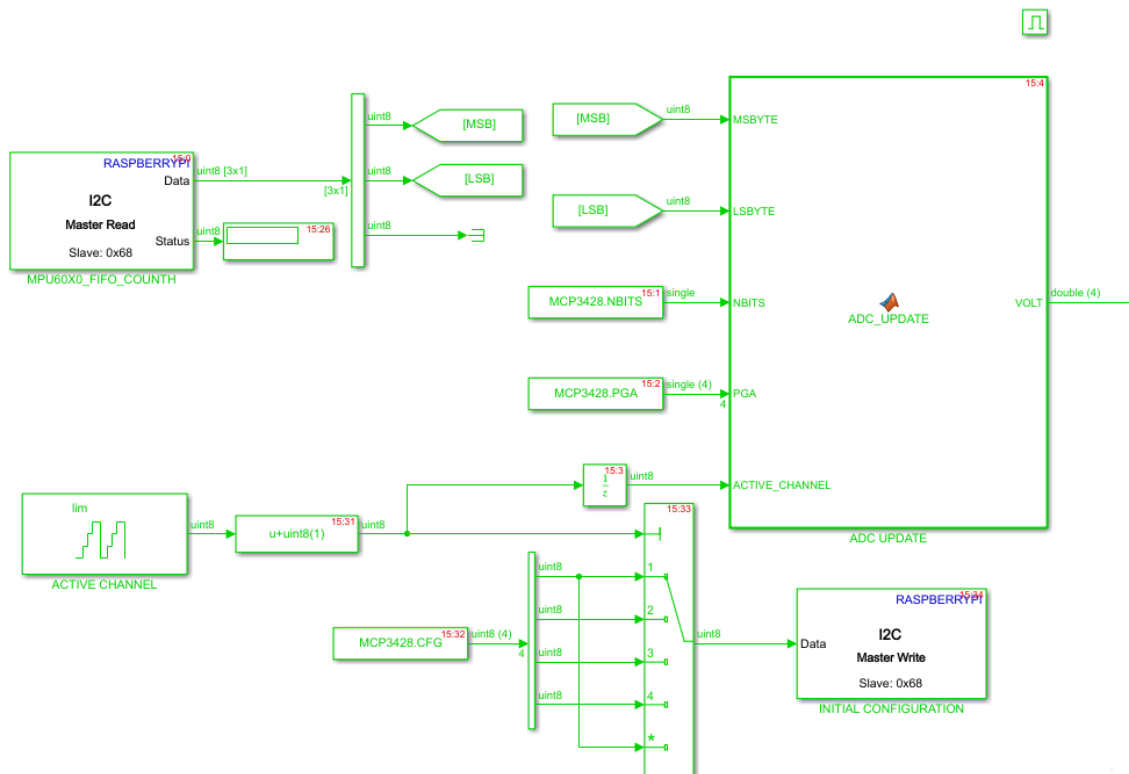


Figure 3.6.: ADC read model

The Simulink model performs 4 functions: first the model initializes the ADC board by writing the configuration byte to the ADC board through the Raspberry I2C Master Write

block and checks for the status of the write. If successful the model moves on to the main control loop for the board which consists of sending a configuration byte corresponding to the channel to be read, then requests the measurement from the ADC and finally converts the output into first volts, and then it's sent to the infrared's model for conversion into the necessary units.

3.2.3 PWM CLICK

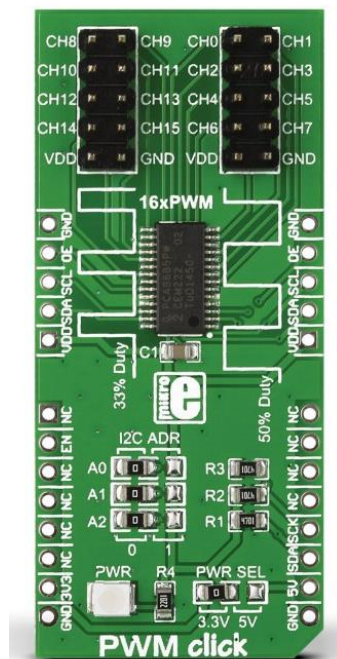


Figure 3.7.: PWM Click board.

The PWM click consists of a PCA9685PW chip, which is an I2C-bus controlled 16-channel LED controller optimized for RGBA color applications, wired to the necessary mikroBUS socket and to 30 male header pins, giving access to all the outputs the chip needs. Even though the chip is designed for LED control, it still outputs easily controllable PWM signals which work with any PWM driven peripherals, including all the motors installed on the RC Car: the 4 servomotors that drive the active suspension, the single direction servomotor and the ESC which in turn controls the Brushless motor. [If to this point PWM Signals haven't been introduced, do so]

The PCA9685PW requires 3 configuration registers to be setup before any of the PWM channels can be used, the MODE1 and MODE2 registers store the operation configuration of the board, such as whether sleep mode is activated, and if auto increase is activated. The last register is the PWM pre-scaler, which allows to quickly change the frequency of the PWM signal, in this case the motors need 50Hz.

After the operation registers are configured, the board requires the duty cycle of each PWM signal in order to generate them, this is set up using the LEDx_ON and LEDx_OFF registers. These 12bit registers hold a value between 0-4095, where 0 represents the start of the cycle and 4095 represents the end of the cycle, they are also split into most and least significant 4bit and 8bit registers respectively and the x represents the channel the registers control. The chip needs the duty cycle to be inputted by specifying the period in which the PWM signal is Vcc each cycle as shown in the next figure from the chip's datasheet

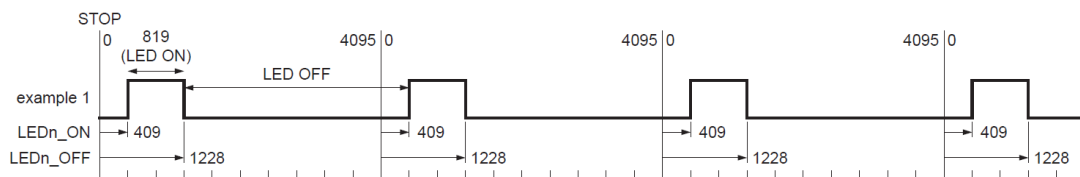


Figure 3.8.: PWM signal diagram

Using the registers and simple math any duty cycle can be replicated in the board.

The model used to control the board is shown in the next figure

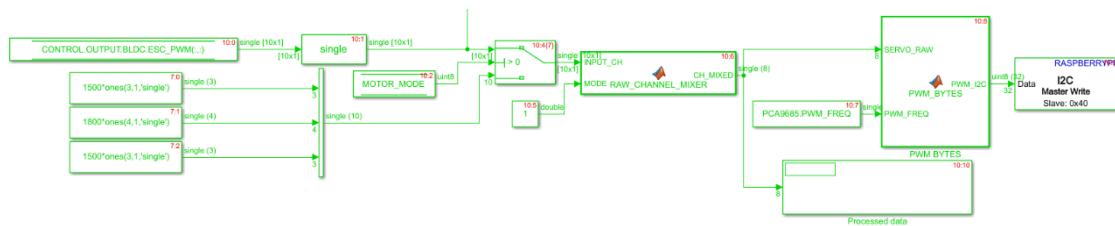


Figure 3.9.: PWM control model

After writing the configuration registers, this model simply converts the values received from the control program and converts them into instructions that can be loaded into the PCA9685PW LED. Before the signals are converted the program checks if the instructions from the controls are within the operation range of the motors, and switches around the signals into their appropriate channels. Once everything is prepared the PWM_BYTES function translates the μ s used in RC equipment into the ON and OFF counts for each channel and it gets written in the respective register using the I2C Master Write block.

3.2.4. COUNTER CLICK

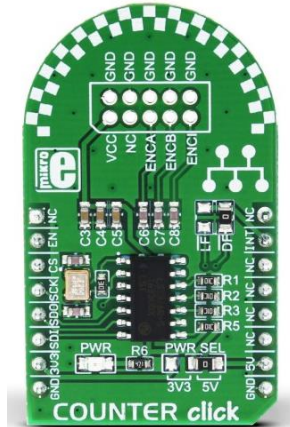


Figure 3.10.: Counter Click board

The Counter Click board uses a LS7366R Chip a 32-bit quadrature counter, designed to be used in conjunction with quadrature encoders and communicates through the SPI protocol. While the project does not use one of those encoders, the ESC that controls the motor has a sensor plug that includes the state of the 3 phases used to drive the motor, which can be used like the output of an incremental quadrature encoder as shown in the following image:

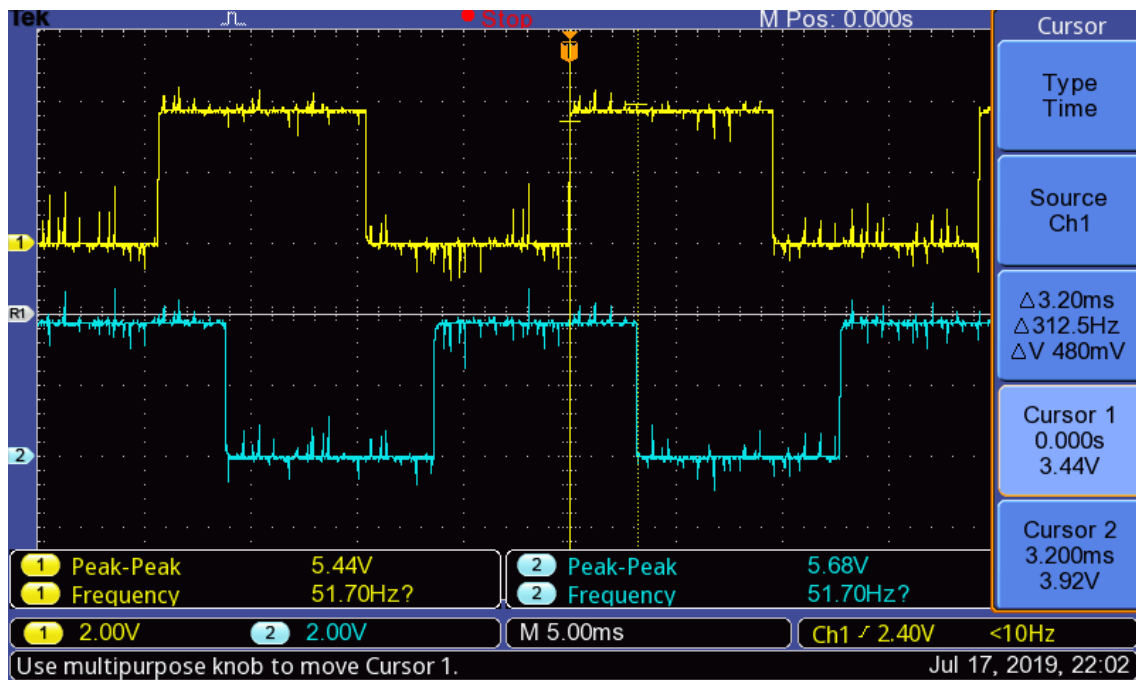


Figure 3.11.: Motor phase sensors

The sensor signals of the ESC are square and out of phase 90deg, which is the exact input the counter is expecting. Thus, the counter can count the number of revolutions the brushless motor does as its rotation depends on the state of the 3 phases driving the motor. With the number of rotations stored in the counter, the code can then calculate the difference between counts each sampling period to calculate the number of revolutions per second the brushless motor makes, allowing the program to know the exact speed of the car after taking into account the gear ratios between the motor and the wheels and wheel diameter to calculate the speed of the car in m/s. This can be seen in the following Simulink models:

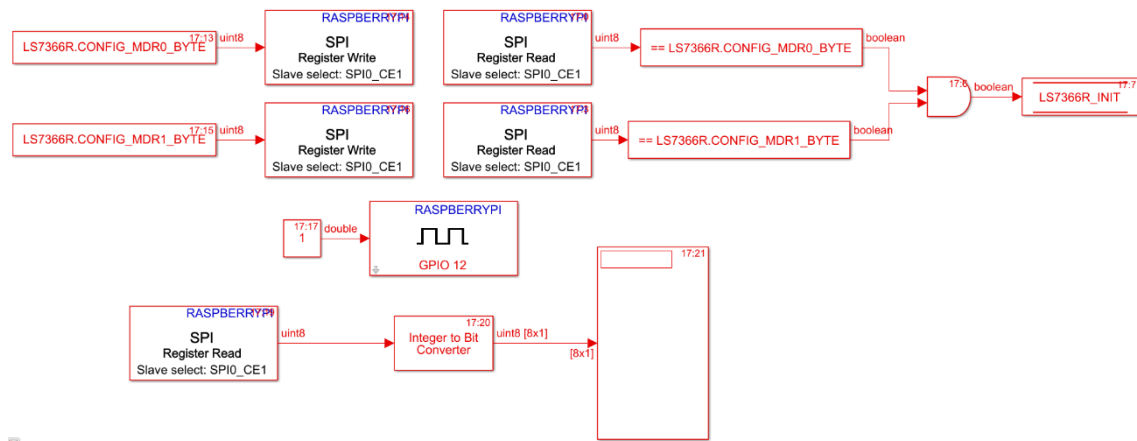


Figure 3.12.: Model for initializing the counter board.

First the program initializes the board, the LS7366R has 2 mode registers to configure, MDR0 controls whether the counter uses the quadrature counting mode, if it's in free-running mode, and whether the index input is activated among other things that can be seen in the chip's datasheet {reference}. MDR1 controls the number of bytes the counter uses, whether counting is enabled and allows the user to enable several flags to monitor the counter. For this project, the LS7366R is configured in x1 quadrature, free-running mode with the index pin disabled and using 3 bytes for the count. One last detail about the counter is that the counter has an enable pin which must be set to Vcc for the counter to count, this leads to a GPIO conflict with the PWM click board as the enable pin is mapped to the same mikroBUS Socket pin if both boards are connected to the same Shuttle click board as the PWM click needs 0V in that pin. Therefore, the car has 2 Shuttle clicks to avoid placing both boards on the same mikroBUS socket.

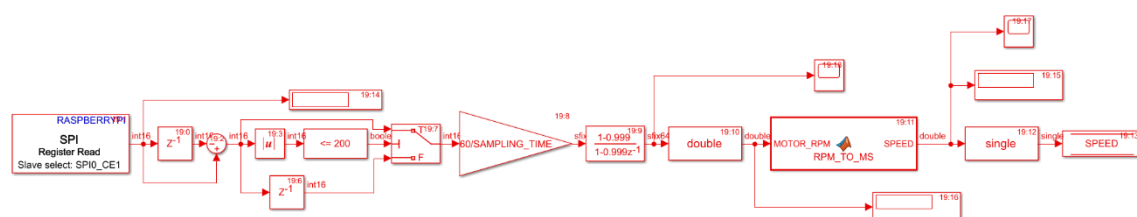


Figure 3.13.: Model for Counter processing

After a proper initialization of the counter board, the program requests each sampling period the current count of the counter, calculates the difference with the previous count, converts the number into revolutions per minute, and filters the number to get a steady value. After that a small MATLAB function converts the RPM into m/s which is then stored in the control bus for use elsewhere in the program.

3.2.5. MPU IMU CLICK

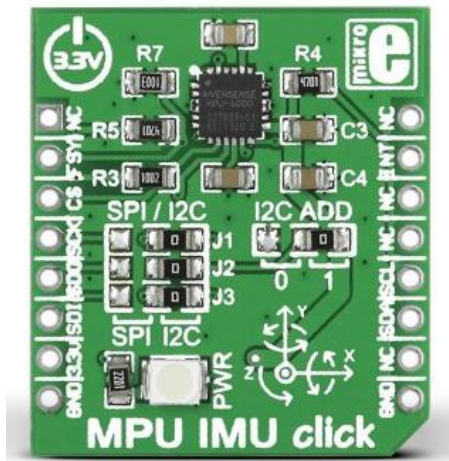


Figure3.14.: MPU IMU click board

The MPU IMU Click board is composed of an MPU-6000 chip, this Inertial Motion Unit (IMU) [DEIM31] chip is one of the first to contain a “Motion processing unit” that includes a 3 gyroscopes, one for each axis, and a 3 accelerometers in a single chip, alongside the needed computing power to calculate all the changes in the object’s angular or perpendicular acceleration, measurements which are essential for the project. This board also can communicate through I2C or SPI, with the latter being the protocol used in this case.

An accelerometer detects changes in the proper acceleration of a given object, in this case the RC car [MOHA18]. This IMU has 3 accelerometers which allows it to measure changes in acceleration in each of the 3 axes of the car, in a proper frame. Out of these axes the program only needs the vertical acceleration to measure changes in the height of the car alongside the infrared sensors.

A gyroscope is a device that detects changes in the angular acceleration of the object it’s affixed to, allowing the program to keep track of the orientation of the vehicle and react accordingly [DEIM31]. The IMU has 3 gyroscopes, one for each of the axis: pitch,

roll, and yaw. The program only needs pitch and roll, which are affected by the movement of a vehicle in irregular terrain.

Most of the code for this specific IMU chip was already made for other drone projects, so it was easily repurposed and implemented onto this project

The IMU's data measurements registers can be accessed using the Raspberry's SPI communication blocks, the data output from the IMU is then processed into rad/s^2 and m/s^2 respectively and lastly it is processed by the IMU calibration function block which as long as the calibration is activated in the program it will calculate the offsets of the IMU measurements, although the RC Car should be level during this calibration for the measurement to be accurate. After calibration the block simply removes the offset from the IMU measurements and saves the measurements to the control data bus.

IMU also takes advantage of a previously designed Extended Kalman Filter (EKF) [SING18] designed for it. An EKF is a type of predictive filter that estimates the next measurements in order to calculate a weighted average with the measurement, hopefully reducing the noise or errors in the measurement. An EKF is the non-linear version of a Kalman filter, and due to the linearization needed it no longer is a optimal estimator and therefore there can be errors in estimation causing the measurements to diverge from the true value. Nevertheless, it gives out a reasonable performance making it the de facto standard for GPS and most navigation systems and its performance with the prototype is quite noticeable.

3.3. Pi EzConnect

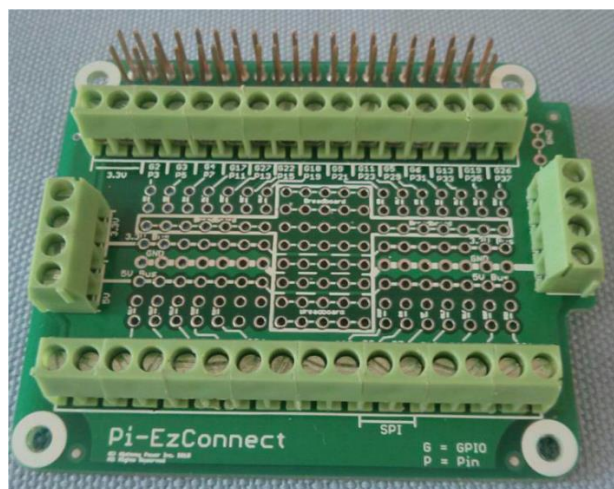


Figure 3.15.: Pi-EzConnect shield

Before installing the PI 3 Click shield, the project needs a way to access some of the Raspberry's GPIO pins, namely the dedicated serial pins, and unfortunately the PI 3 Click shield takes up the entirety of the Raspberry's 40-pin header without providing a passthrough for other boards to use. To remedy this, the project uses the PI EzConnect, which is a prototyping shield board which provides solderless connections to all of the Raspberry Pi's GPIO pins, as well as GPIO solder points for more permanent connections, and finally in has a breadboard sections where standard male or female headers can be soldered for a compact breadboard section.

This project only makes use of the solderless points, as the PI EzConnect is sandwiched between the Raspberry Pi and the Shield Click, and there is very little clearance to solder the headers and connect the cables.

3.4. Receiver and transmitter combo

3.4.1. FLYSKY FS-A8S RECEIVER & FS-I6S TRANSMITTER COMBO

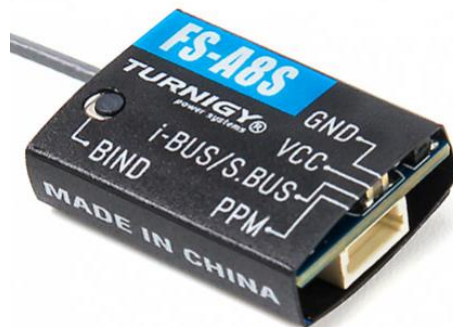


Figure 3.16.: FS-A8S receiver

To control the RC car, we need a controller with a transmitter and a corresponding receiver connected to the Raspberry Pi to relay those commands, the receiver chosen is the FS-A8S made by Flysky. Designed with aerial drones in mind, the FS-A8S is 2.4GHz receiver using 8 channels with standard PPM or 18 channels with IBUS. PPM stands for Pulse-Position Modulation and in the field of hobbyist radio control a PPM signals looks like miniature PWM signals all sent together. A PPM frame is 22.5ms long and is composed of a total of 9 pulses separated by 3ms spaces, 8 of which are for each one of the 8 channels it transmits, each pulse ranging from 0.7ms to 1.7ms and the final is for the start pulse which is as long as need to get the total frame length of 22.5ms. This setup allows the receiver to only use three wires to relay the controller data alongside as the necessary ground and Vcc wires, a big advantage over the previous PWM protocol

which has a set of 3 wires per each channel to relay to the vehicle. A typical PPM frame can be seen in the following image taken from [MFTE19]

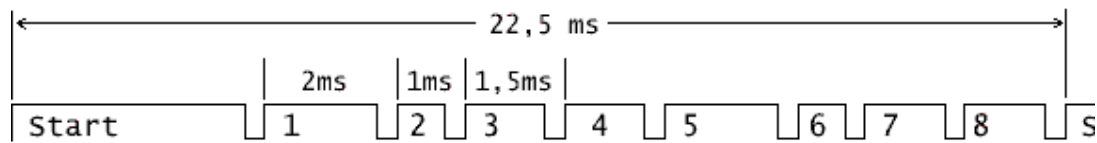


Figure 3.17.: PPM frame scheme

IBUS is a relatively new protocol by Flysky and it's their version of the SBUS protocol introduced by Futaba mostly used nowadays by Futaba itself and FrSky. As its name might imply it consists of a digital serial protocol. IBUS communicates using serial UART and allows for two-way communication between the receiver and the transmitter allowing the vehicle in question to send battery, speed and other telemetry data back to the user, unfortunately the FS-A8S is a one-way receiver so this cannot be implemented. IBUS supports up to 18 channels with any compatible transmitter and receiver combo.

The project will be using the FS-A8S through I-BUS as it's easier to implement with the Raspberry Pi than PPM, due to the Pi already having a serial port and corresponding serial read blocks from its toolset, making it trivial to access the controller's data stream and commands once the transmission is parsed for each channel's values.



Figure 3.18.: FS-I6S transmitter

The transmitter used alongside the FS-A8S is the FS-I6S, another Flysky product in order to capitalize on the proprietary IBUS protocol. The FS-I6S is a 10 channel 2.4 GHz receiver which is more than enough for the current project, as only 6 of those channels will be used as follows:

- Channel 1: Represents the right X-axis stick which handles manual control of the rear suspension servos
- Channel 2: Represents the right Y-axis stick which handles the Vehicle's throttle
- Channel 3: Represents the left Y-axis stick which handles the manual control of the front suspension servos
- Channel 4: Represents the left X-axis stick which handles the turning servo
- Channel 5: Represents the top left switch which changes between automatic and manual control of the suspension servos
- Channel 8: Represents the top right switch from the left which acts as a Safety switch

The Top left switch allows the user to toggle between manual control of the Pitch and roll of the vehicle, while automatic mode lets the control handle the suspension. The safety switch disables input to the servos for calibration purposes, in its up position it enables the motors while the bottom position disables them.

3.5. Finished RC car

The finished fully assembled RC car prototype looks as follows:

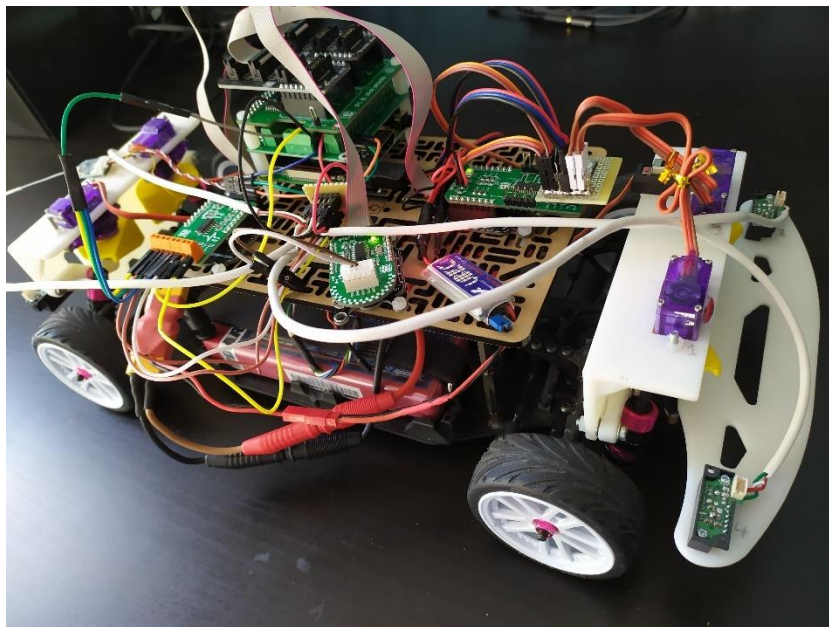


Figure 3.19.: Finished RC car

The main difference from the previous design is the second level for all the electronic hardware, except for the IMU which is directly installed onto the chassis of the car, to ensure correct readings from the IMU. The second difference is the weight, with the

second level, as well as more electronics, end up raising the weight of the car but the suspension system servomotors are more than capable of handling the extra weight, testing done with the manual control of the suspension systems shows no noticeable difference between pre upgrade performance and now.

CHAPTER 4: OBTAINING THE MODELS

In order to be able to design a control scheme for the active suspension system, the transfer functions that link the movement of the RC car with its sensors are needed. These movements are the pitch, roll and the height of the vehicle, and as these are 3 distinct movements, a transfer function will be generated for each, with the end goal of controlling each movement separately using a PID regulator for each movement.

To achieve this, 3 tests will be conducted, one for each movement, that send a PRBS (Pseudorandom Binary Sequence) signal to the appropriate servomotors. A PRBS signal is chosen as its Fourier harmonics analysis shows that all of its harmonics have the same amplitude, therefore the generated model will be compatible for all frequencies [JACK71].

To generate this PRBS signal we use the MATLAB command `idinput` as follows:

```
TEST PITCH PRBS = idinput(512, 'prbs', [0 0.02], [1500 2100]);
```

- 512 stands for the number of samples to be generated
- 'prbs' signals the type of input signal to generate
- [0 0.02] Is the frequency range the signal will have. This needs to be small, in order of servomotor movement be fast enough to react to the changes it will face in a real track.
- [1500 2100] represents the range of values the signal can take; this range is chosen to use the full range of motion the suspension motors can give without lowering the chassis into the ground

Once the signal is generated it needs to be looped so that the test can be as long as needed, for this the Repeating Signal Star block is used, looping the signal indefinitely until the test is done.

4.1. Model Tests

The test requires that the car pitches forward and backward, while using the full range of motion the suspension servos can give before it collides with the ground. During the test, the pitch angle measured by the vehicle's IMU will be recorded.

The following model is the one used for all these tests:

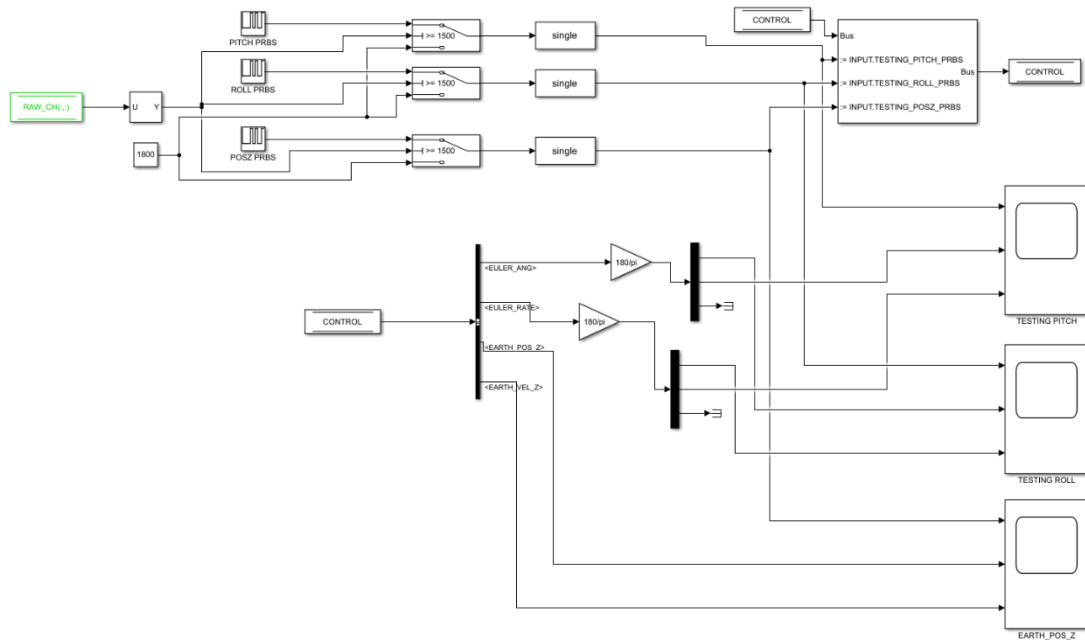


Figure 4.1.: Simulink model used to test for the necessary transfer functions

The Repeating Sequence Star blocks input the generated PRBS signals into the control BUS which then gets relayed into the appropriate channels in the main control update block, this block also decides which test to run each time, depending on the setup configuration, both of these files can be seen in the annex section of this document.

Once the IMU's calibration phase is complete, the test can be started from the transmitter. As the test is underway the 3 scopes seen in the model record their respective movement alongside its angular rate or speed and the input PRBS signal generated.

After the tests the data recorded is then prepared for processing with MATLAB's system identification command

4.1.1. PITCH TEST AND PLANT CALCULATION

This test requires the vehicle to pitch forwards and backwards, in order to see the effect of the full pitching motion on the pitch angle of the vehicle measure by the installed IMU.

Using the testing model shown in Figure 4.1. configured for a pitch test, the following data was obtained:

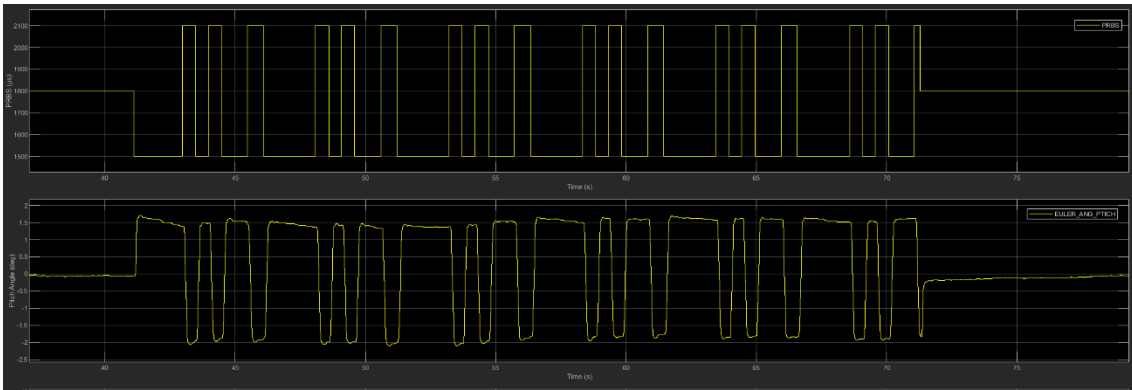


Figure 4.2.: Pitch test results, calibration time section omitted

Now the data needs to be prepared before we can calculate the transfer function of the plant. This is accomplished with the `DATA_PRE_PROCESSING_PITCH.m` script, which remove the mean value of the data as well as reducing the range of the PRBS signal to $[-1, 1]$ in order for it to match the control to be designed for them.

Once the data is prepared, it is imported into the System Identification toolbox, which attempt to estimate the transfer functions of the plant of each movement

First the data must be imported from the workspace as a time domain signal as shown in the following image:

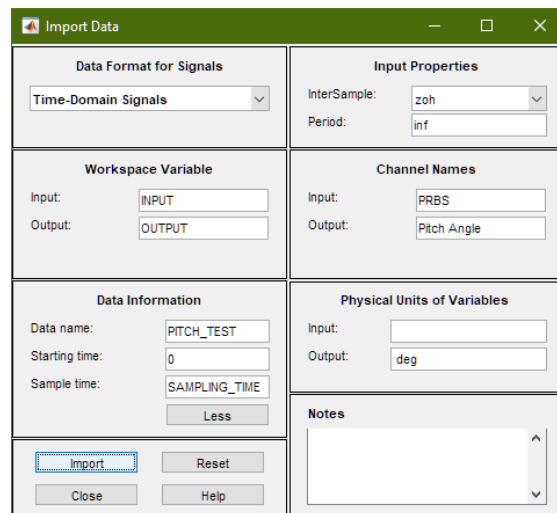


Figure 4.3.: Pitch data import

The script that prepares the data for the toolbox renames the PRBS signal as `INPUT` and the pitch/roll/height measurement as `OUTPUT`. The data imported can be seen in the following image:

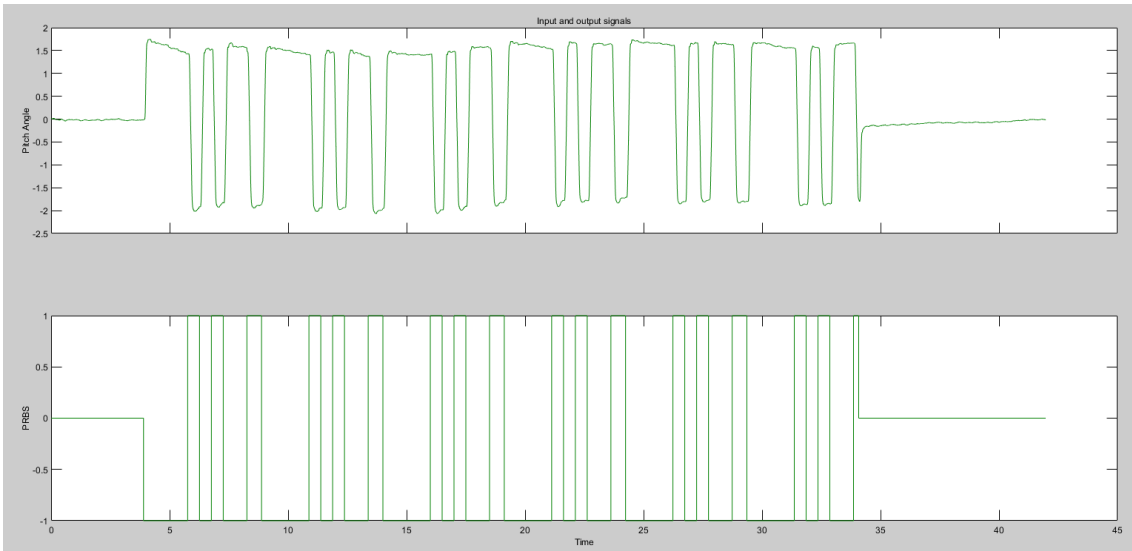


Figure 4.4.: Prepared Pitch data

Once successfully imported as time domain data, the data must be then set both as *Working Data* and *Validation Data* in the toolbox before it can be processed. It is worth mentioning that the toolbox can also perform some pre-processing such the range of the data if needed. to process the model, select the *Estimate -->* box and in the following dropdown menu selected the option *Process Models* as seen in the following screenshot:

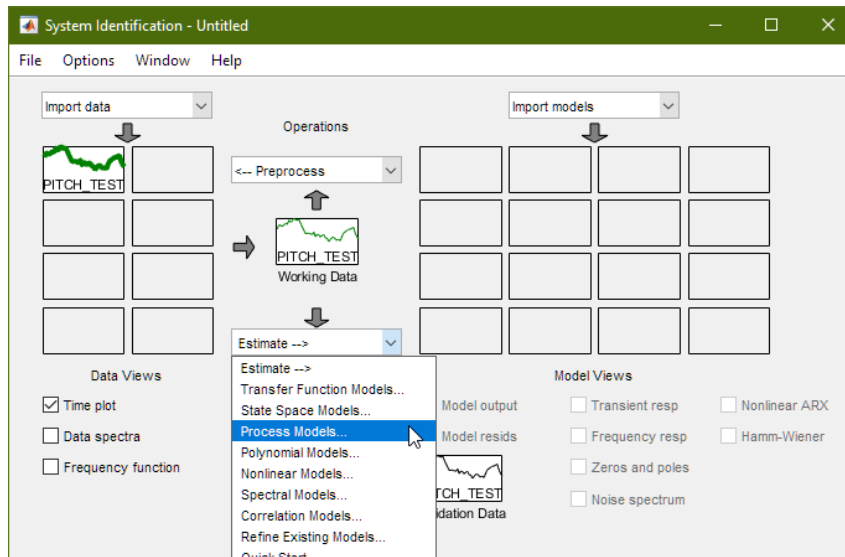


Figure 4.5.: Identification toolbox

This brings up the Processing Models window, here the number of poles, whether a zero is calculated can be selected, as well as initial estimates for each of the transfer function's values. Once set the toolbox will estimate the best fit for the settings and data provided.

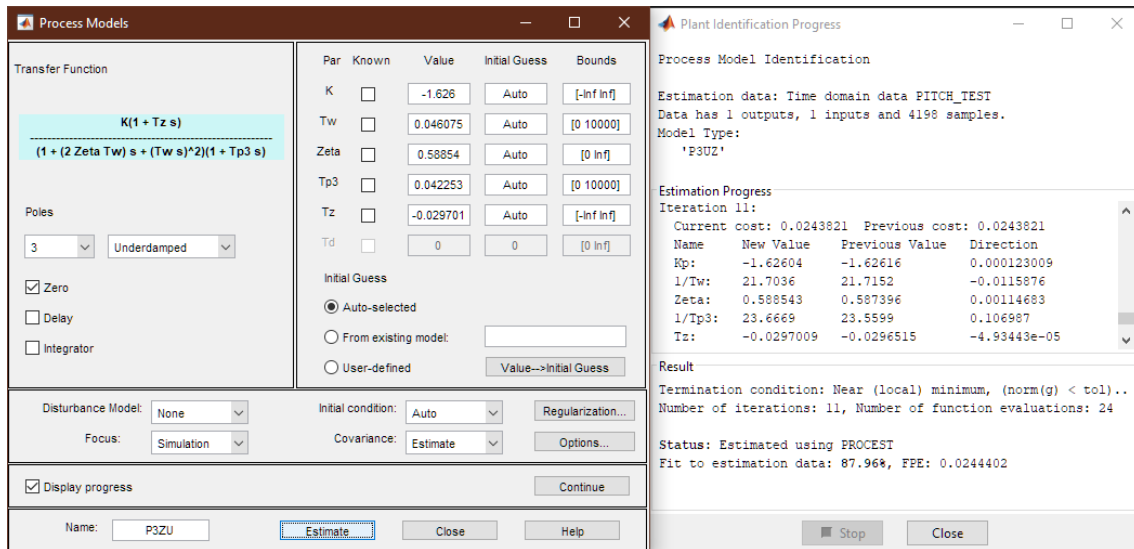


Figure 4.6.: Process models settings and results for pitch

In the case of the pitch the best match is 87.96% and the fit to the measured data can be seen below:

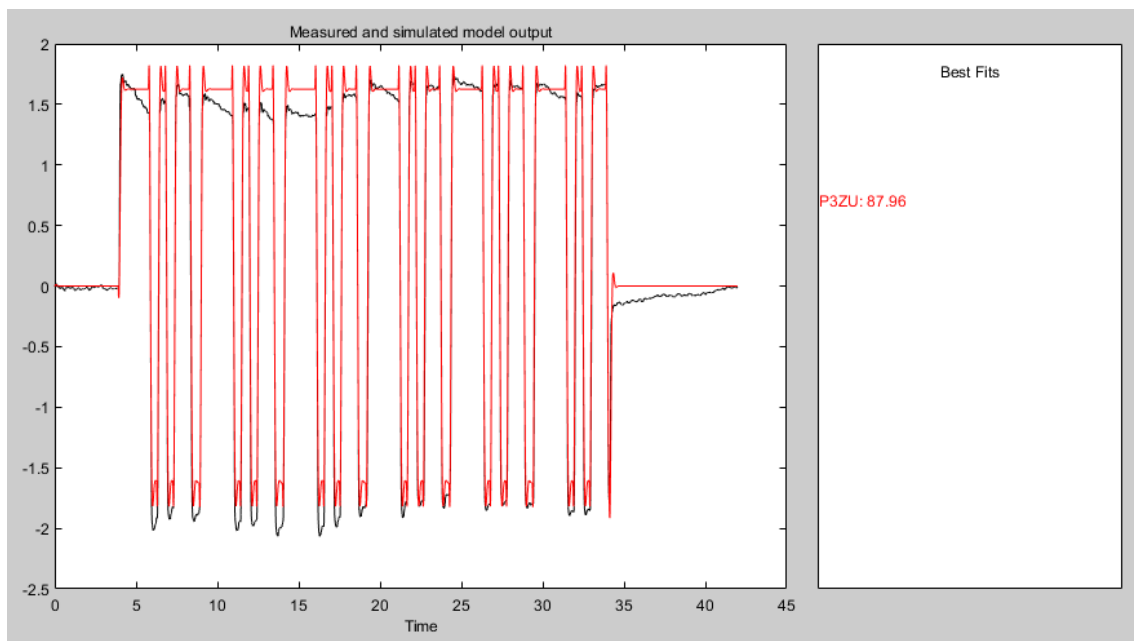


Figure 4.7.: Simulated Pitch output vs Measured Pitch output.

Leaving us with the following transfer function

$$TF_{Pitch} = \frac{-53.8398(s + 33.6689)}{(s + 2.3667)(s^2 + 25.547 + 471.052)}$$

4.1.2. ROLL TEST AND PLANT CALCULATION

This test requires the vehicle to roll from left to right, in order to see the effect of the full roll motion on the roll angle of the vehicle measured by the installed IMU.

This follows the same method as before, using the testing model shown in Figure 4.1. configured this time for a roll test, the following data was obtained:

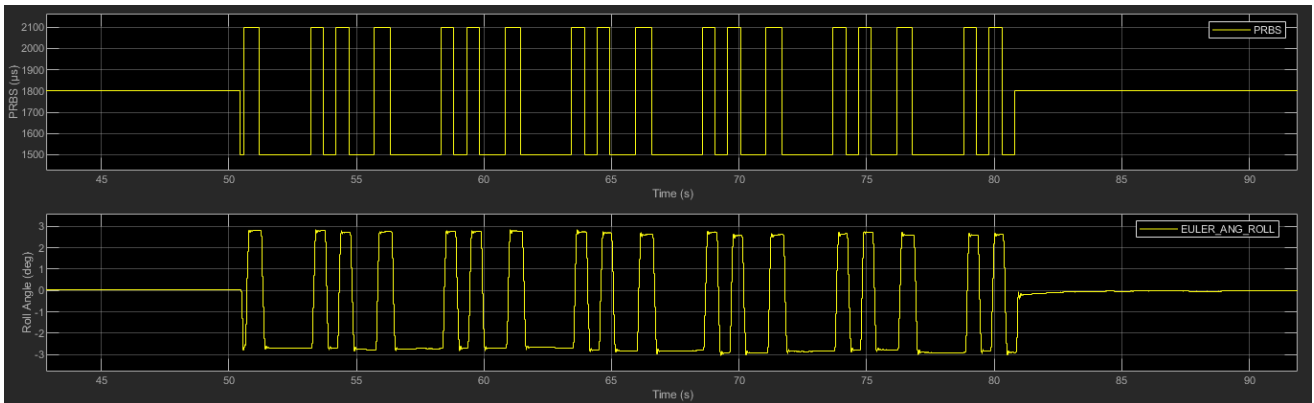


Figure 4.8.: Roll test results, calibration time section omitted

As mention during the pitch test the data needs to be prepared using DATA_PRE_PROCESSING_ROLL.m script, which does the same functions as it's pitch counter part only for the roll measured data. The pre-processed data can be seen in the next image:

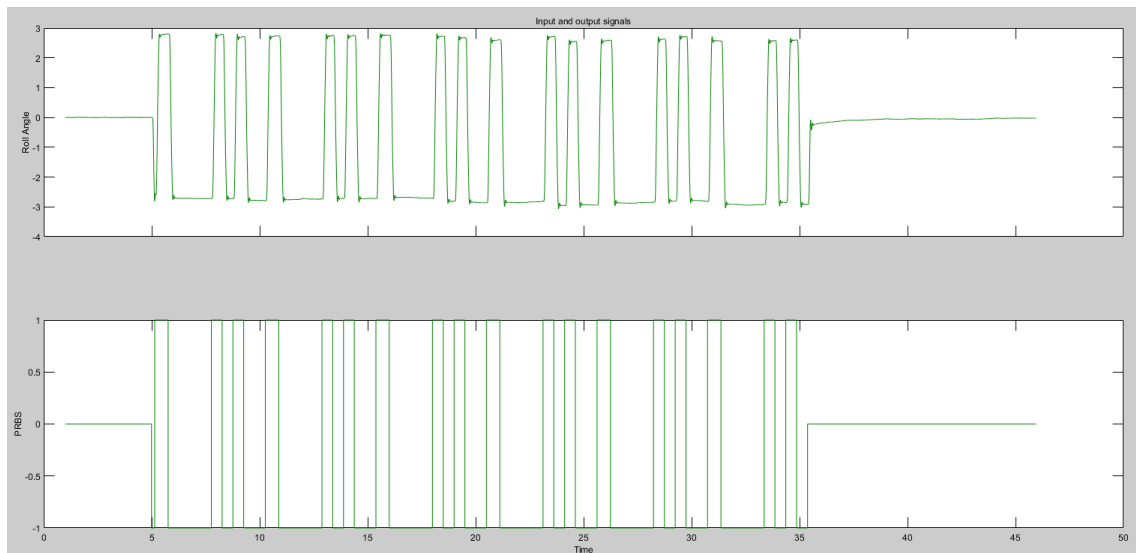


Figure 4.9.: Prepared Roll data

Following the same method as before, the calculated transfer function for the Roll plant is as follows:

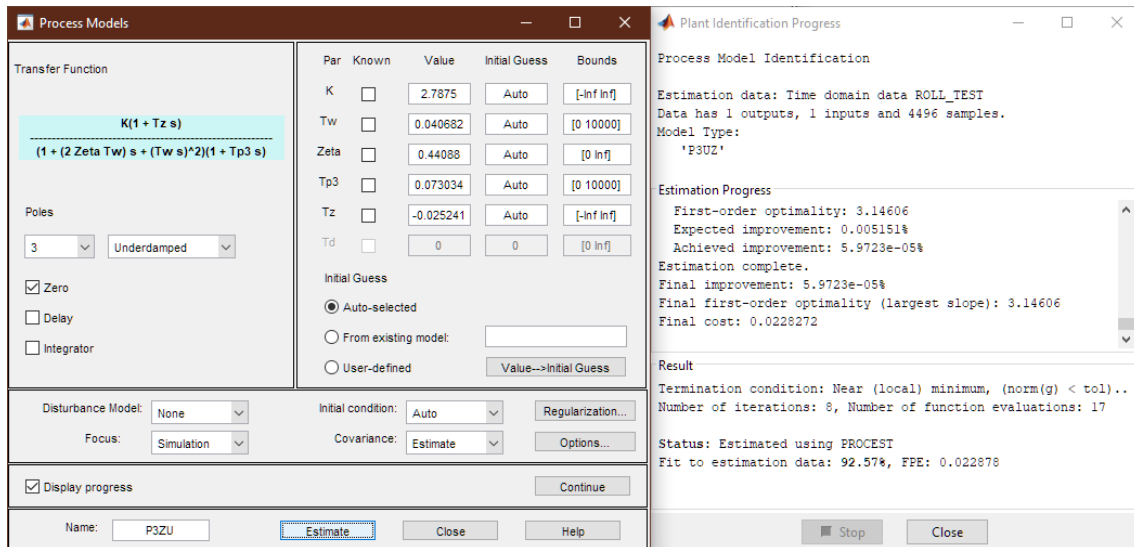


Figure 4.10.: Process models settings and results for roll

In the case of the roll the best match is 92.57% and the fit to the measured data can be seen below:

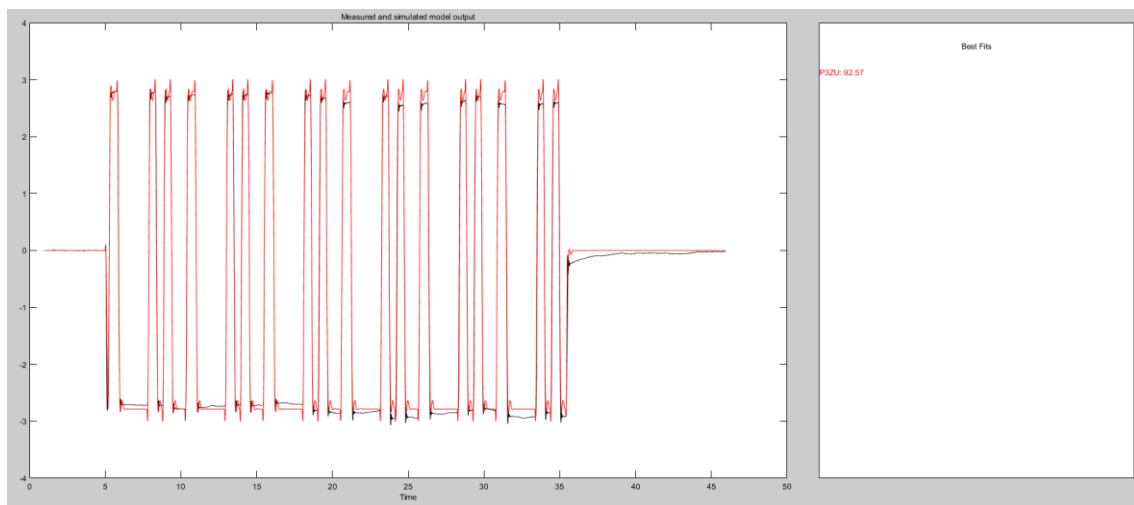


Figure 4.11.: Simulated Roll output vs Measured Roll output.

Leaving us with the following transfer function

$$TF_{Roll} = \frac{582.092(s + 39.6181)}{(s + 13.6923)(s^2 + 21.6745s + 604.22)}$$

4.1.2. ROLL TEST AND PLANT CALCULATION

This test requires the vehicle to raise its chassis up and down, in order to see the effect of the full vertical range of the system on the height of the vehicle measured by the installed IMU and infrared sensors.

Once again this follows the same method as before, using the testing model shown in Figure 4.1. configured this time for a roll height, the following data was obtained:

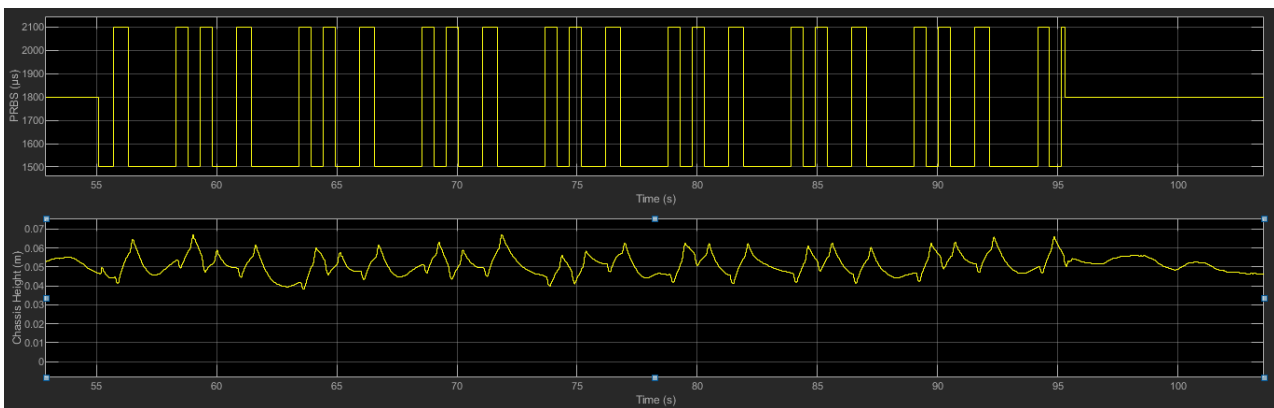


Figure 4.12.: Height test results, calibration time section omitted

As mention during the pitch test the data needs to be prepared using DATA_PRE_PROCESSING_POS_Z.m script, which does the same functions as it's pitch counterpart only for the height test measured data. The pre-processed data can be seen in the next image:

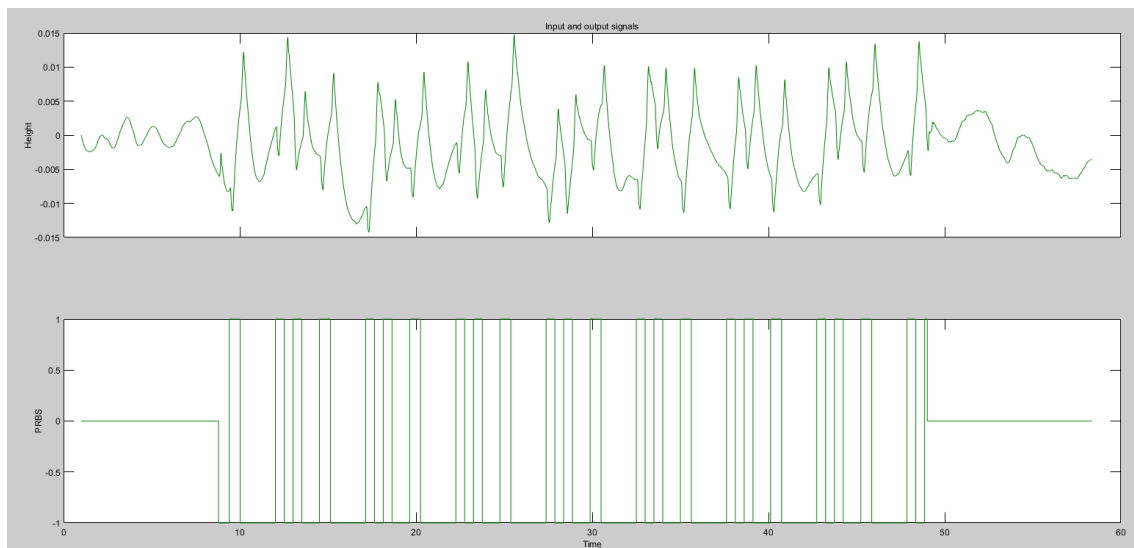


Figure 4.13.: Prepared height data

Following the same method as before, the calculated transfer function for the height plant is as follows:

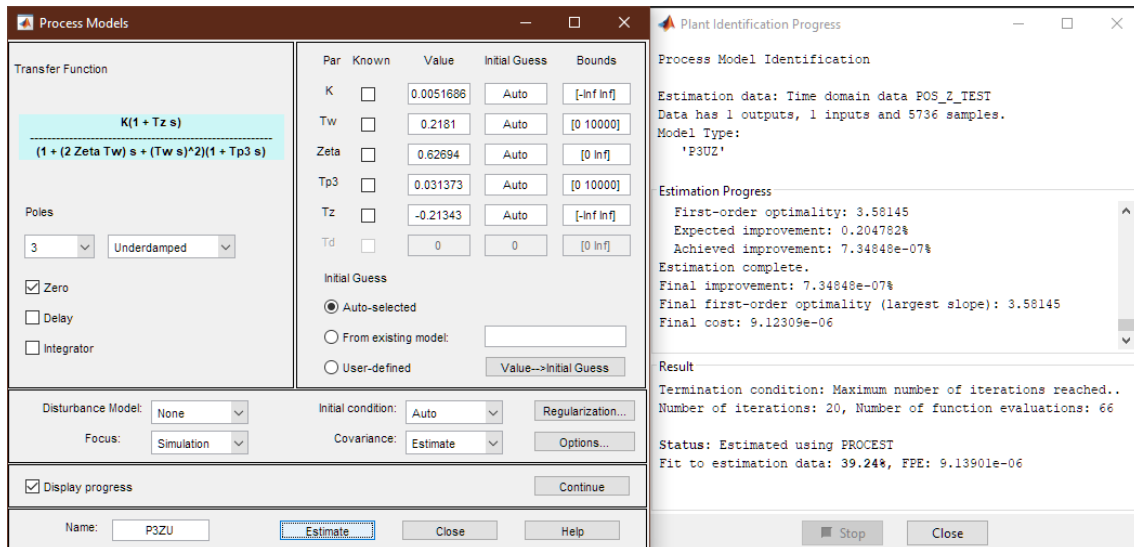


Figure 4.14.: Process models settings and results for height

Unfortunately, in the case of the height the best match is 39.24% which is not enough to represent the plant of the height properly. To get around this, another method of estimation can be used, with a least squares approach [GIOR85] looking like the best for this set of data, but there was not enough time to set up the necessary scripts to perform this estimation and as such the height transfer function was not calculated and consequently the controls for each plant were not made due to a lack of time.

CHAPTER 6: CONCLUSIONS

Even though it was not possible to finish the end goal of this work, that being the designing a control scheme to test the active suspension system on the RC car, it was possible to successfully finish the other 2 objectives, which are to upgrade the electronic hardware without hampering the mechanical active suspension system, and creating a Simulink model that allows full functionality of the RC car prototype and allows for complex control strategies to be implemented.

The first of these objectives, is to upgrade the electronic hardware without hampering the functionality of the active suspension system, this was successfully implemented adding a new floor to the vehicle housing all the new electronics without changing the performance of the active suspension system during manual trials.

The second objective is to prepare a Simulink model allowing access to all the new and old functionality of the car, allowing direct telemetry data to be collected from the car without needing to stop to access the data from the onboard memory thanks to the official Simulink Raspberry Pi support and correct setup of all the components.

With these 2 objectives, the main motivation to upgrade and setup this vehicle with a working active suspension setup for future classes and projects is accomplished. All in all, the prototype is ready to test out new control schemes in the control focused classes or project in the university, and hopefully it helps create new projects in this ever-increasing field in the future.

CHAPTER 7: FUTURE DEVELOPMENTS

There are various new avenues of research available to continue from this work, either stemming directly from this project as improvements or completely new ones:

- Finishing the simple control strategy planned for the project
- The testing of different and more sophisticated control strategies, researching their effectiveness on active suspension systems, using this setup.
- Designing more conventional and complex semi-active/active suspension systems such as hydraulics and electromagnetic suspensions and testing them using quarter-car setups (only one wheel)
- Testing the previous points on real vehicles or on half-car models

REFERENCES

- [ALEX11] C. Alexandru and P. Alexandru, "A comparative analysis between the vehicles'," INTERNATIONAL JOURNAL OF MECHANICS, vol. 5, no. 4, p. 8, 2011.
- [SHIR19] Shirish, "What Is Active Suspension Or Adaptive Suspension? - CarBikeTech," carbiketech.com, 12 April 2019. [Online]. Available: <https://carbiketech.com/active-suspension-adaptive-suspension/>. [Accessed 25 August 2019].
- [JIMÉ16] F. d. B. Jiménez Valverde, "Diseño de un sistema de suspensión activa para un vehículo a escala," Trabajo de fin de grado, ICAI, Universidad Pontificia Comillas, Madrid, 2016.
- [AZUM19] T. Azuma, "What Is Traction Control And How Does It Work? - CAR FROM JAPAN," 25 February 2019. [Online]. Available: <https://carfromjapan.com/article/car-maintenance/traction-control-system/>. [Accessed 25 August 2019].
- [MODE19] Modelflight, "What is an Electronic Speed Controller and how does it differ from brushed to brushless motors?," Modelflight, 06 February 2019. [Online]. Available: <https://www.modelflight.com.au/blog/electronic-speed-controllers>. [Accessed 25 August 2019].
- [CIRC16] Circuit Basics, "Basics of the I2C Communication Protocol," Circuit Basics, 13 February 2016. [Online]. Available: <http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol>. [Accessed 25 August 2019].
- [MIKE17] MIKEGRUSIN, "Serial Peripheral Interface (SPI) - Learn.sparkfun.com," SparkFun, 2017. [Online]. Available: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all>. [Accessed 25 August 2019].
- [CHEN94] J.-H. Chen, S.-C. Lee and D. B. DeBra, "Gyroscope free strapdown inertial measurement," Journal of Guidance, Control, and Dynamics, vol. 17, no. 2, pp. 286-290, 1994.
- [MOHA18] Z. Mohammed, I. M. Elfadel and M. Rasras, "Monolithic Multi Degree of Freedom (MDoF) Capacitive MEMS Accelerometers," Micromachines, vol. 9, no. 11, p. 602, 2018.
- [DEIM31] R. F. Deimel, "Mechanics of the Gyroscope," Nature, vol. 128, no. 3225, pp. 289-289, 1931.
- [SING18] H. Singh, "Extended Kalman Filter: Why do we need an Extended Version?," Medium, 7 April 2018. [Online]. Available: <https://towardsdatascience.com/extended-kalman-filter-43e52b16757d>. [Accessed 25 August 2019].
- [JACK71] P. A. Jackson, "PRBS cross-correlation measurements by hybrid computational techniques," The Computer Journal, vol. 14, no. 1, pp. 49-54, 1971.

[GIOR85] A. A. Giordano and F. M. Hsu, Least Square Estimation with Applications to Digital Signal Processing, New York: John Wiley & Sons, Inc., 1985.

[MFTE19] MFTech, "R/C PPM encoding - MFTech," MFTech, [Online]. Available: http://www.mftech.de/ppm_en.htm. [Accessed 25 August 2019].

[PAGO06] F. L. Pagola, Regulación Automática, Madrid: Universidad Pontificia Comillas, 2006.

PART 2: ECONOMIC ANALYSIS

CHAPTER 1: ECONOMIC ANALYSIS

As it was previously mentioned in the state-of-the-art section of this work, the field of active suspension is a one that is in constant growth, as all luxury car brands have or are starting their own version of Mercedes-Benz's Magic body control, and others like ClearMotion are exclusively developing their own systems to license to the main car manufactures like, Audi, Mercedes Benz, Renault, etc. That's not counting the importance placed into these systems by professional competition teams as the difference in handling is night and day.

This positions this work in a market that is about to boom. Autonomous vehicles are becoming each day more of a reality and they will need some type active suspension system to stay relevant, as most users will want to focus on other things while the car handles the navigation and driving, and unstable rides make that difficult.

In addition, most automotive manufacturing companies are constantly designing and redesigning their systems, as while these systems are effective in stabilizing the vehicle, they are also still far too expensive to market to lower end vehicle series: as the cost in designing, fabrication, implementation still carry too many costs and that not taking into account the price of the necessary high-end sensors for the system that normally require expensive maintenance.

Therefore, this field is one that in the future is going to need more experienced and trained engineers that can effectively design new strategies and find viable ways to implement them, something that the Comillas Pontifical University can support with more projects like this one, creating a new avenue for employment in multinational enterprises.

PART 3: ANNEXES

CHAPTER 1: CONFIG_RC_CAR.M

This code sets up everything needed for the model to function, it loads all the necessary data and configuration files to run all the peripherals

```
clc
clear
format short e

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% GENERAL PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Sampling time (s)
SAMPLING_TIME = 10e-3;
fprintf('SAMPLING_TIME (s) = %g s\n', SAMPLING_TIME);
% Gravity in Madrid (m/s^2)
GRAVITY = 9.80208;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% SENSORS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% SENSOR UPDATE PERIOD (samples): [ Frequency Slot]
% Update frequency in samples. If frequency = 0, sensor is not
activated
SENSOR_FREQ = [
    % ACCELEROMETER XYZ -> 1
    1 0
    % IR DISTANCE Z AXIS -> 2
    1 0

];

% ***** IMU CONFIGURATION *****
run('../HARDWARE_COMPONENTS/CONFIG_MPU6000')
% ***** MAGNETOMETER CONFIGURATION *****
run('../HARDWARE_COMPONENTS/CONFIG_NO_MAGNETOMETER')
% ***** GPS CONFIGURATION *****
run('../HARDWARE_COMPONENTS/CONFIG_M8NUBLOX')
% ***** BAROMETER CONFIGURATION *****
run('../HARDWARE_COMPONENTS/CONFIG_MS5611')
% ***** BATTERY MANAGEMENT SYSTEM *****
run('../HARDWARE_COMPONENTS/CONFIG_BMS')
% ***** ADC MCP3428 SETUP *****
run('../HARDWARE_COMPONENTS/CONFIG_MCP3428')
% ***** COUNTER LS7366R SETUP *****
run('../HARDWARE_COMPONENTS/CONFIG_LS7366R')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% RC RECEIVER
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
```

```

% ***** RC RECIEVER CONFIGURATION *****
run('../HARDWARE_COMPONENTS/CONFIG_RCRX_IBUS')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ACTUATORS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ***** BLDC CONFIGURATION *****
run('../HARDWARE_COMPONENTS/CONFIG_BLDC')
run('../HARDWARE_COMPONENTS/CONFIG_PCA9685')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% MODEL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
run('../SOFTWARE_COMPONENTS/MODEL/CONFIG_MODEL')
MODEL_INI.PARAM.SENSOR_FREQ = SENSOR_FREQ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% EKF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cd '../SOFTWARE_COMPONENTS/EKF'
EKF_INI = CONFIG_EKF(MODEL_INI);
SENSOR_ACT = boolean([ ...
    (SENSOR_FREQ(1,1)>0)*ones(3,1) % ACCEL XYZ
    (SENSOR_FREQ(2,1)>0) %DIST_Z
]);
EKF_INI.SENSOR_ACT = SENSOR_ACT;
clear SENSOR_ACT
cd ../../CONFIGURATION

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CONTROL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cd '../SOFTWARE_COMPONENTS/CONTROL'
CONTROL_INI = CONFIG_CONTROL(MODEL_INI);
CONTROL_INI.EKF = EKF_INI;
CONTROL_INI.INPUT.IMU = IMU_INI;
CONTROL_INI.INPUT.RCRX = RCRX_INI;
CONTROL_INI.INPUT.ADC = ADC_INI;
CONTROL_INI.INPUT.ENCODER = ENCODER_INI;
CONTROL_INI.OUTPUT.BLDC = BLDC_INI;
CONTROL_INI.PARAM.SENSOR_FREQ = single(SENSOR_FREQ);
%PRBS
TEST_PITCH_PRBS = idinput(512, 'prbs', [0 0.02], [1500 2100]);
TEST_ROLL_PRBS = idinput(512, 'prbs', [0 0.02], [1500 2100]);
TEST_POSZ_PRBS = idinput(512, 'prbs', [0 0.02], [1500 2100]);
cd ../../CONFIGURATION

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% MAVLINK
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% UART transfer rate (bits/s)
MAVLINK_TRANSFER_RATE = 57600; % UART
fprintf('MAVLINK TRANSFER RATE = %d bits/s\n',MAVLINK_TRANSFER_RATE);
% MAVLINK sampling time
% MAVLINK SAMPLING_TIME = 2*SAMPLING_TIME;
MAVLINK_SAMPLING_TIME = SAMPLING_TIME;
fprintf('MAVLINK SAMPLING TIME = %g s\n',MAVLINK_SAMPLING_TIME);
% MAVLINK buffer size
MAVLINK_BUFFER_SIZE = 50;
cd ../SOFTWARE_COMPONENTS/MAVLINK
fprintf('MAVLINK BUFFER SIZE = %d bytes\n',MAVLINK_BUFFER_SIZE);
% SYSTEM MAVLINK configuration
PARAM_LIST_LEN = 1;
disp('MAVLINK FOR CONTROL SYSTEM:')
MAVLINK_SYS_INI = CONFIG_MAVLINK(MAVLINK_BUFFER_SIZE,[0 30
26],1,0,MAVLINK_SAMPLING_TIME,PARAM_LIST_LEN);
% PC MAVLINK configuration
disp('MAVLINK FOR PC CONTROL STATION:')
MAVLINK_PC_INI = CONFIG_MAVLINK(MAVLINK_BUFFER_SIZE,[0
191],0,0,MAVLINK_SAMPLING_TIME,PARAM_LIST_LEN);
cd ../../CONFIGURATION

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% BUS DEFINITIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% -----
-----
cd '../BUS_DEFINITIONS'
BusDefinition(MODEL_INI,'MODEL_Bus')
BusDefinition(EKF_INI,'EKF_Bus')
BusDefinition(CONTROL_INI,'CONTROL_Bus')
BusDefinition(MAVLINK_SYS_INI,'MAVLINK_Bus')
% -----
-----
cd ../CONFIGURATION

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% SIMULINK MODEL CONFIGURATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
cd('../SIMULINK');
MODEL_SLX = 'RASPI_CONTROL_SYSTEM';
open(MODEL_SLX)
% RUN_MODE_DEFINITION
% / 0. REAL-TIME SIMULATION / 1. CONTROL VALIDATION / 2. IMPLEMENTATION
RUN_MODE = 2;
switch RUN_MODE
    case 0 % REAL-TIME SIMULATION

set_param(MODEL_SLX,'FixedStep','MODEL_INI.PARAM.SIM_SAMPLING_TIME');
    set_param(MODEL_SLX,'StopTime','inf');
    set_param([MODEL_SLX '/HARDWARE'],'Commented','on');
    set_param([MODEL_SLX '/SIMULATION'],'Commented','off');
    set_param([MODEL_SLX '/Microseconds at
Start'],'Commented','on');
    set_param([MODEL_SLX '/Microseconds at
End'],'Commented','on');
    set_param([MODEL_SLX '/COMPUTATIONAL_LOAD'],'Commented','on');

```



```

        set_param([MODEL_SLX '/MONITORIZATION/BLACK
BOX'], 'Commented', 'on');
        set_param([MODEL_SLX '/MONITORIZATION/EXTERNAL MODE:
SCOPES'], 'Commented', 'on');
        set_param([MODEL_SLX '/MONITORIZATION/SIMULATION:
SCOPES'], 'Commented', 'off');
        set_param([MODEL_SLX
'/MONITORIZATION/MAVLINK'], 'Commented', 'on');
        set_param([MODEL_SLX '/SIMULATION/Simulation
Pace'], 'Commented', 'off');
        set_param([MODEL_SLX '/MONITORIZATION/SIMULATION:
SCOPES/Second-order LPF 1'], 'Commented', 'through');
        set_param([MODEL_SLX '/MONITORIZATION/SIMULATION:
SCOPES/Second-order LPF 2'], 'Commented', 'through');
        set_param([MODEL_SLX '/MONITORIZATION/SIMULATION:
SCOPES/Second-order LPF 3'], 'Commented', 'through');
        set_param([MODEL_SLX '/MONITORIZATION/SIMULATION:
SCOPES/Second-order LPF 4'], 'Commented', 'through');
        set_param(MODEL_SLX, 'SimulationMode', 'Normal');
        CONTROL_INI.STATE.CURRENT_STATUS = uint8(0);
        CONTROL_INI.STATE.PREVIOUS_STATUS = uint8(0);
        % Closed-loop transfer-function filter activation
        for nn = 1:12
            set_param([MODEL_SLX '/MONITORIZATION/SIMULATION:
SCOPES/Second-order LPF ' num2str(nn)], 'Commented', 'through');
        end
        case 1 % FAST SIMULATION

set_param(MODEL_SLX, 'FixedStep', 'MODEL_INI.PARAM.SIM_SAMPLING_TIME');

set_param(MODEL_SLX, 'StopTime', num2str(MODEL_INI.PARAM.SIM_FINAL_TIME)
);
        set_param([MODEL_SLX '/HARDWARE'], 'Commented', 'on');
        set_param([MODEL_SLX '/SIMULATION'], 'Commented', 'off');
        set_param([MODEL_SLX '/SIMULATION/Simulation
Pace'], 'Commented', 'on');
        set_param([MODEL_SLX '/Microseconds at
Start'], 'Commented', 'on');
        set_param([MODEL_SLX '/Microseconds at
End'], 'Commented', 'on');
        set_param([MODEL_SLX '/COMPUTATIONAL LOAD'], 'Commented', 'on');
        set_param([MODEL_SLX '/MONITORIZATION/BLACK
BOX'], 'Commented', 'on');
        set_param([MODEL_SLX '/MONITORIZATION/EXTERNAL MODE:
SCOPES'], 'Commented', 'on');
        set_param([MODEL_SLX '/MONITORIZATION/SIMULATION:
SCOPES'], 'Commented', 'off');
        set_param(MODEL_SLX, 'SimulationMode', 'Normal');
        % Closed-loop transfer-function filter activation
        for nn = 1:12
            set_param([MODEL_SLX '/MONITORIZATION/SIMULATION:
SCOPES/Second-order LPF ' num2str(nn)], 'Commented', 'through');
        end
        %***** EKF TUNING *****
        % FLIGHT
        CONTROL_INI.STATE.CURRENT_STATUS = uint8(8);
        CONTROL_INI.STATE.PREVIOUS_STATUS = uint8(8);
        %*****
        case 2 % IMPLEMENTATION
            try
                clear rpi

```

```

        rpi = raspberrypi;
        rpi.stopModel(MODEL_SLX);
        aux = pwd;
        rpi.system(['rm -rf \MATLAB_ws/R2018b/' aux(1)])
    catch
    end
    set_param(MODEL_SLX, 'FixedStep', 'IMU_INI.SAMPLING_TIME');
    set_param(MODEL_SLX, 'StopTime', 'inf');
    set_param([MODEL_SLX '/HARDWARE'], 'Commented', 'off');
    set_param([MODEL_SLX '/SIMULATION'], 'Commented', 'on');
    set_param([MODEL_SLX '/MONITORIZATION'], 'Commented', 'off');
    set_param([MODEL_SLX '/Microseconds at
Start'], 'Commented', 'on'); % CHANGED FOR THROTTLE TESTING SHOULD BE
'off'
        set_param([MODEL_SLX '/Microseconds at
End'], 'Commented', 'on'); % CHANGED FOR THROTTLE TESTING SHOULD BE
'off'
        set_param([MODEL_SLX '/COMPUTATIONAL
LOAD'], 'Commented', 'on'); % CHANGED FOR THROTTLE TESTING SHOULD BE
'off'
        set_param([MODEL_SLX '/MONITORIZATION/BLACK
BOX'], 'Commented', 'on');
        set_param([MODEL_SLX '/MONITORIZATION/EXTERNAL MODE:
SCOPES'], 'Commented', 'off');
        set_param([MODEL_SLX '/MONITORIZATION/SIMULATION:
SCOPES'], 'Commented', 'on');
        set_param([MODEL_SLX
'/MONITORIZATION/MAVLINK'], 'Commented', 'on');
        set_param(MODEL_SLX, 'SimulationMode', 'External');
        % set_param([MODEL_SLX '/Target Setup'], 'Commented', 'off');
        % SYSTEM STATUS: ANGLE CONTROL
        CONTROL_INI.STATE.CURRENT_STATUS = uint8(0);
        CONTROL_INI.STATE.PREVIOUS_STATUS = uint8(0);
    otherwise
end

clear RUN_MODE MODEL_SLX
clear MAVLINK_TRANSFER_RATE MAVLINK_BUFFER_SIZE PARAM_LIST_LEN
clear EKF_INPUT_SIZE EKF_STATE_SIZE EKF_OUTPUT_SIZE

```

CHAPTER 2: CONFIG_MCP3428.M

This is the initialization code for the ADC3 Click board, which loads all the necessary addresses and variables needed for the model to function.

```
%% CONFIGURATION FILE FOR THE MCP3428 ANALOGUE TO DIGITAL CONVERTER
% Slave address for ADC converter MCP3428
MCP3428.ADDRESS = bin2dec('01101000');
MCP3428.ADDRESS_WRITE = hex2dec('D0');
MCP3428.ADDRESS_READ = hex2dec('D1');

% Configuration register for CH1, OSM, 240 SPS, PGA=1 (12 bits)
MCP3428.CONFIG_REG_100 = bin2dec('10000000');
% Configuration register for CH2, OSM, 240 SPS, PGA=1 (12 bits)
MCP3428.CONFIG_REG_200 = bin2dec('10100000');
% Configuration register for CH3, OSM, 240 SPS, PGA=1 (12 bits)
MCP3428.CONFIG_REG_300 = bin2dec('11000000');
% Configuration register for CH4, OSM, 240 SPS, PGA=1 (12 bits)
MCP3428.CONFIG_REG_400 = bin2dec('11100000');
% Configuration register for CH1, OSM, 60 SPS, PGA=1 (14 bits)
MCP3428.CONFIG_REG_101 = bin2dec('10000100');
% Configuration register for CH2, OSM, 60 SPS, PGA=1 (14 bits)
MCP3428.CONFIG_REG_201 = bin2dec('10100100');
% Configuration register for CH3, OSM, 60 SPS, PGA=1 (14 bits)
MCP3428.CONFIG_REG_301 = bin2dec('11000100');
% Configuration register for CH4, OSM, 60 SPS, PGA=1 (14 bits)
MCP3428.CONFIG_REG_401 = bin2dec('11100100');
% Configuration register for CH1, CCM, 240 SPS, PGA=1 (12 bits)
MCP3428.CONFIG_REG_110 = bin2dec('10010000');
% Configuration register for CH2, CCM, 240 SPS, PGA=1 (12 bits)
MCP3428.CONFIG_REG_210 = bin2dec('10110000');
% Configuration register for CH3, CCM, 240 SPS, PGA=1 (12 bits)
MCP3428.CONFIG_REG_310 = bin2dec('11010000');
% Configuration register for CH4, CCM, 240 SPS, PGA=1 (12 bits)
MCP3428.CONFIG_REG_410 = bin2dec('11110000');
% Configuration register for CH1, CCM, 60 SPS, PGA=1 (14 bits)
MCP3428.CONFIG_REG_111 = bin2dec('10010100');
% Configuration register for CH2, CCM, 60 SPS, PGA=1 (14 bits)
MCP3428.CONFIG_REG_211 = bin2dec('10110100');
% Configuration register for CH3, CCM, 60 SPS, PGA=1 (14 bits)
MCP3428.CONFIG_REG_311 = bin2dec('11010100');
% Configuration register for CH4, CCM, 60 SPS, PGA=1 (14 bits)
MCP3428.CONFIG_REG_411 = bin2dec('11110100');

%% General parameters
% Reference voltage
MCP3428.VREF = single(2.048); % V
% Samples per second
MCP3428.SPS = uint8(240); % Hz
% Number of bits
MCP3428.NBITS = uint8(12);
% Conversion mode
% / 0: Continuous conversion / 1: One-shot conversion
MCP3428.MODE = boolean(1);
% Configuration bytes for each channel
MCP3428.CFG = [MCP3428.CONFIG_REG_100 MCP3428.CONFIG_REG_200 ...
               MCP3428.CONFIG_REG_300 MCP3428.CONFIG_REG_400]';
% Programmable gain amplifier for each channel
```

```
MCP3428.PGA = uint8([1 1 1 1]');  
% Total number of channels converted  
MCP3428.NUM_CH = 4;  
  
%% INI Setup  
ADC_INI.CHANNEL1 = 0;  
ADC_INI.CHANNEL2 = 0;  
ADC_INI.CHANNEL3 = 0;  
ADC_INI.CHANNEL4 = 0;
```

CHAPTER 3: CONFIG_LS7366R.M

This is the initialization code for the Counter Click board, which loads all the necessary addresses and variables needed for the model to function.

```
%% CONFIGURATION FILE FOR THE LS7366R 32-BIT QUADRATURE COUNTER
LS7366R.SAMPLING_TIME = 1e-03;
%-----
% IR Codes/ADDRESSES
LS7366R.WR_MDR0_ADDRESS = uint8(bin2dec('10001000'));
LS7366R.WR_MDR1_ADDRESS = uint8(bin2dec('10010000'));
LS7366R.RD_MDR0_ADDRESS = uint8(bin2dec('01001000'));
LS7366R.RD_MDR1_ADDRESS = uint8(bin2dec('01010000'));
LS7366R.RD_CNTR_ADDRESS = uint8(bin2dec('01100000'));
LS7366R.RD_STR_ADDRESS = uint8(bin2dec('01110000'));
LS7366R.CLR_CNTR_ADDRESS = uint8(bin2dec('00100000'));
%-----
% CONFIG BYTES
% FILTER CLOCK DIV=1, INDEX DISABLED, FREE-RUNNING MODE, X1 QUADRATURE
MODE
LS7366R.CONFIG_MDR0_BYTE = uint8(bin2dec('01000001'));
% NO FLAGS, COUNTING ENABLED, 2-BYTE MODE
LS7366R.CONFIG_MDR1_BYTE = uint8(bin2dec('00000010'));
%CLEAR COUNTER DUMMY BYTE
LS7366R.CLEAR_CNTR_BYTE = uint8(0);
%-----
% INI
ENCODER_INI.SPEED = single(0);
```

CHAPTER 4: CONTROL UPDATE FUNCTION

This is the code for the control update function that handles the different controls strategies and necessary code for their correct operation.

```
function CONTROL_OUT = CONTROL_UPDATE(CONTROL_IN)

%% COPY CONTROL BUS
CONTROL_OUT = CONTROL_IN;

%% CONTROL_MODE
% / 0. INITIALIZATION / 1. ESTIMATION / 2.
SETTING OPERATION POINT
% / 3. TESTING VELOCITY CONTROL / 4. VELOCITY CONTROL ONLY / 5. FULL
CONTROL
% / 6. MODEL TESTING
CONTROL_MODE = CONTROL_OUT.STATE.CONTROL_MODE;
TESTING_MODE = CONTROL_OUT.STATE.TESTING_MODE;
if CONTROL_MODE == 0 % INITIALIZATION
    return
else
    CONTROL_OUT.EKF.EKF_MODE = CONTROL_OUT.STATE.EKF_MODE;
end

%% READ MEASUREMENTS
GYRO = CONTROL_OUT.INPUT.IMU.GYRO; % rad/s
ACCEL = CONTROL_OUT.INPUT.IMU.ACCEL; % m/s^2
GYRO_MEAN = CONTROL_OUT.INPUT.IMU.GYRO_MEAN; % rad/s
ACCEL_MEAN = CONTROL_OUT.INPUT.IMU.ACCEL_MEAN; % m/s^2
DIST_Z = CONTROL_OUT.INPUT.DIST_Z;
RAW_CH = CONTROL_OUT.INPUT.RCRX.RAW_CH;
SPEED = CONTROL_OUT.INPUT.ENCODER.SPEED;

%% EKF UPDATE
% EKF INPUT = [GYRO_MEAN ; ACCEL_MEAN ; GYRO_BIAS_INPUT ;
ACCEL_BIAS_INPUT]
CONTROL_OUT.EKF.INPUT = [GYRO_MEAN ; ACCEL_MEAN ;
zeros(6,1,'single')];
% EKF OUTPUT
CONTROL_OUT.EKF.OUTPUT = [ACCEL ; DIST_Z];
% EKF UPDATE
CONTROL_OUT.EKF = EKF(CONTROL_OUT.EKF);
% STATE UPDATE
EULER_ANG = CONTROL_OUT.EKF.STATE(1:3); % rad
GYRO_BIAS = CONTROL_OUT.EKF.STATE(4:6); % rad/s
EARTH_VEL_Z = CONTROL_OUT.EKF.STATE(7); % m/s
EARTH_POS_Z = CONTROL_OUT.EKF.STATE(8); % m
ACCEL_BIAS = CONTROL_OUT.EKF.STATE(9:11); % m
CONTROL_OUT.INPUT.EARTH_POS_Z = EARTH_POS_Z;
CONTROL_OUT.INPUT.EARTH_VEL_Z = EARTH_VEL_Z;
GRAVITY = CONTROL_OUT.PARAM.GRAVITY;
CONTROL_OUT.INPUT.EARTH_ACCEL_Z = ...
    single([0,0,1]*(CONTROL_OUT.EKF.PARAM.matDCM_BE*([1 1 -
1]'.*(ACCEL-ACCEL_BIAS)) + [0 0 GRAVITY].'));

```

```

CONTROL_OUT.INPUT.EULER_ANG = EULER_ANG;
BODY_RATE = GYRO - GYRO_BIAS;
EULER_RATE = ...
    CONTROL_OUT.EKF.PARAM.matRATE_BE*BODY_RATE;
CONTROL_OUT.INPUT.EULER_RATE = EULER_RATE;
if CONTROL_MODE == 1 % ESTIMATION
    return
end

%% INITIALIZE OUTPUTS
ESC_PWM = 1500*ones(10,1,'single');
% CONTROL PASS THROUGH FOR INPUTS
ESC_PWM(3) = RAW_CH(3); %STEERING

%% CONTROL PARAMETERS
% GENERAL PARAMETERS
SAMPLING_TIME = CONTROL_OUT.PARAM.SAMPLING_TIME;
%CONTROL OUTPUT VALUE
PITCH_CONTROL = CONTROL_OUT.OUTPUT.PITCH_CONTROL;
ROLL_CONTROL = CONTROL_OUT.OUTPUT.ROLL_CONTROL;
POS_Z_CONTROL = CONTROL_OUT.OUTPUT.POS_Z_CONTROL;

%% CONTROL STATE
persistent initialize PIT_INT_STATE PIT_DER_STATE ROLL_INT_STATE
ROLL_DER_STATE ...
    POS_Z_INT_STATE POS_Z_DER_STATE VEL_INT_STATE
VEL_DER_STATE
if isempty(initialize)
    PIT_INT_STATE = single(0);
    PIT_DER_STATE = single(0);
    ROLL_INT_STATE = single(0);
    ROLL_DER_STATE = single(0);
    POS_Z_INT_STATE = single(0);
    POS_Z_DER_STATE = single(0);
    VEL_INT_STATE = single(0);
    VEL_DER_STATE = single(0);
    initialize = 1;
end
%% SET OPERATION HEIGHT
switch CONTROL_MODE
    case {2,5}
        ESC_PWM(4) = 1800;
        ESC_PWM(5) = 1800;
        ESC_PWM(6) = 1800;
        ESC_PWM(7) = 1800;
    otherwise
end

%% START OF CONTROL STRATEGY REGION
%% VELOCITY XYZ: PID CONTROLLER
switch CONTROL_MODE
    case 2
    case 3 % Square Wave testing
        % INPUTS
        CTRL_INPUT.TARGET =
single((single(CONTROL_OUT.INPUT.TESTING_STEP)-1500)*(1.4e-
3)); %THROTTLE TARGET %single((CONTROL_OUT.INPUT.TESTING_STEP-
1500)*(1.4e-3));
        CTRL_INPUT.MEASUREMENT = single(SPEED);
        CTRL_INPUT.DERIVATIVE = single(0);
        CTRL_INPUT.INT_STATE = VEL_INT_STATE;

```

```

CTRL_INPUT.DER_STATE = VEL_DER_STATE;
% PARAMETERS
CTRL_PARAM.Ts = SAMPLING_TIME;
CTRL_PARAM.K = CONTROL_OUT.PARAM.VEL_K;
CTRL_PARAM.Ti = CONTROL_OUT.PARAM.VEL_Ti;
CTRL_PARAM.Td = CONTROL_OUT.PARAM.VEL_Td;
CTRL_PARAM.b = CONTROL_OUT.PARAM.VEL_b;
CTRL_PARAM.N = CONTROL_OUT.PARAM.VEL_N;
CTRL_PARAM.MAX_CONTROL = CONTROL_OUT.PARAM.VEL_MAX_CONTROL;
CTRL_PARAM.MIN_CONTROL = CONTROL_OUT.PARAM.VEL_MIN_CONTROL;
CTRL_PARAM.INT_DISC_TYPE =
CONTROL_OUT.PARAM.VEL_INT_DISC_TYPE;
CTRL_PARAM.DER_DISC_TYPE =
CONTROL_OUT.PARAM.VEL_DER_DISC_TYPE;
CTRL_PARAM.DER_INPUT = CONTROL_OUT.PARAM.VEL_DER_INPUT;
CTRL_PARAM.ANTIWINDUP = CONTROL_OUT.PARAM.VEL_ANTIWINDUP;
% PID CONTROL
CTRL_OUTPUT = PID(CTRL_INPUT,CTRL_PARAM);
% OUTPUT: ACCELERATION TARGET
ESC_PWM(2) = single(((CTRL_OUTPUT.CONTROL)*(1/(1.4e-
3)))+1500);
% STATE
VEL_INT_STATE = CTRL_OUTPUT.INT_STATE;
VEL_DER_STATE = CTRL_OUTPUT.DER_STATE;

case {4,5} % VELOCITY CONTROL
% INPUTS
CTRL_INPUT.TARGET = single((single(RAW_CH(2))-1500)*(1.4e-
3)); %THROTTLE TARGET %single((CONTROL_OUT.INPUT.TESTING_STEP-
1500)*(1.4e-3));
CTRL_INPUT.MEASUREMENT = single(SPEED);
CTRL_INPUT.DERIVATIVE = single(0);
CTRL_INPUT.INT_STATE = VEL_INT_STATE;
CTRL_INPUT.DER_STATE = VEL_DER_STATE;
% PARAMETERS
CTRL_PARAM.Ts = SAMPLING_TIME;
CTRL_PARAM.K = CONTROL_OUT.PARAM.VEL_K;
CTRL_PARAM.Ti = CONTROL_OUT.PARAM.VEL_Ti;
CTRL_PARAM.Td = CONTROL_OUT.PARAM.VEL_Td;
CTRL_PARAM.b = CONTROL_OUT.PARAM.VEL_b;
CTRL_PARAM.N = CONTROL_OUT.PARAM.VEL_N;
CTRL_PARAM.MAX_CONTROL = CONTROL_OUT.PARAM.VEL_MAX_CONTROL;
CTRL_PARAM.MIN_CONTROL = CONTROL_OUT.PARAM.VEL_MIN_CONTROL;
CTRL_PARAM.INT_DISC_TYPE =
CONTROL_OUT.PARAM.VEL_INT_DISC_TYPE;
CTRL_PARAM.DER_DISC_TYPE =
CONTROL_OUT.PARAM.VEL_DER_DISC_TYPE;
CTRL_PARAM.DER_INPUT = CONTROL_OUT.PARAM.VEL_DER_INPUT;
CTRL_PARAM.ANTIWINDUP = CONTROL_OUT.PARAM.VEL_ANTIWINDUP;
% PID CONTROL
CTRL_OUTPUT = PID(CTRL_INPUT,CTRL_PARAM);
% OUTPUT: ACCELERATION TARGET
ESC_PWM(2) = single(((CTRL_OUTPUT.CONTROL)*(1/(1.4e-
3)))+1500);
% STATE
VEL_INT_STATE = CTRL_OUTPUT.INT_STATE;
VEL_DER_STATE = CTRL_OUTPUT.DER_STATE;
case 6 % PITCH TESTING
switch TESTING_MODE
case 0
case 1

```



```

    if(RAW_CH(7) == 1000)
        %SERVO 1
        ESC_PWM(4) = (1500+300);
        %SERVO 2
        ESC_PWM(5) = (1500+300);
        %SERVO 3
        ESC_PWM(6) = (1800-((1500+300)-1800));
        %SERVO 4
        ESC_PWM(7) = (1800-((1500+300)-1800));
    elseif(RAW_CH(7) == 1500)
        %SERVO 1
        ESC_PWM(4) = CONTROL_OUT.INPUT.TESTING_PITCH_PRBS;
        %SERVO 2
        ESC_PWM(5) = CONTROL_OUT.INPUT.TESTING_PITCH_PRBS;
        %SERVO 3
        ESC_PWM(6) = (1800-
(CONTROL_OUT.INPUT.TESTING_PITCH_PRBS-1800));
        %SERVO 4
        ESC_PWM(7) = (1800-
(CONTROL_OUT.INPUT.TESTING_PITCH_PRBS-1800));
    end
case 2
    if(RAW_CH(7) == 1000)
        %SERVO 1
        ESC_PWM(4) = (1500+300);
        %SERVO 2
        ESC_PWM(5) = (1800-((1500+300)-1800));
        %SERVO 3
        ESC_PWM(6) = (1800-((1500+300)-1800));
        %SERVO 4
        ESC_PWM(7) = (1500+300);
    elseif(RAW_CH(7) == 1500)
        %SERVO 1
        ESC_PWM(4) = CONTROL_OUT.INPUT.TESTING_ROLL_PRBS;
        %SERVO 2
        ESC_PWM(5) = (1800-
(CONTROL_OUT.INPUT.TESTING_ROLL_PRBS-1800));
        %SERVO 3
        ESC_PWM(6) = (1800-
(CONTROL_OUT.INPUT.TESTING_ROLL_PRBS-1800));
        %SERVO 4
        ESC_PWM(7) = CONTROL_OUT.INPUT.TESTING_ROLL_PRBS;
    end
case 3
    if(RAW_CH(7) == 1000)
        %SERVO 1
        ESC_PWM(4) = (1500+300);
        %SERVO 2
        ESC_PWM(5) = (1500+300);
        %SERVO 3
        ESC_PWM(6) = (1500+300);
        %SERVO 4
        ESC_PWM(7) = (1500+300);
    elseif(RAW_CH(7) == 1500)
        %SERVO 1
        ESC_PWM(4) = CONTROL_OUT.INPUT.TESTING_POSZ_PRBS;
        %SERVO 2
        ESC_PWM(5) = CONTROL_OUT.INPUT.TESTING_POSZ_PRBS;
        %SERVO 3
        ESC_PWM(6) = CONTROL_OUT.INPUT.TESTING_POSZ_PRBS;
        %SERVO 4

```

```

        ESC_PWM(7) = CONTROL_OUT.INPUT.TESTING_POSZ_PRBS;
    end
    otherwise
    end
otherwise
end
end

%% PITCH CONTROL: PID CONTROLLER
switch CONTROL_MODE
case 5 % Standard work.
% INPUTS
CTRL_INPUT.TARGET = single(51.5);
CTRL_INPUT.MEASUREMENT = single(EULER_ANG(2));
CTRL_INPUT.DERIVATIVE = single(EULER_RATE(2));
CTRL_INPUT.INT_STATE = PIT_INT_STATE;
CTRL_INPUT.DER_STATE = PIT_DER_STATE;
% PARAMETERS
CTRL_PARAM.Ts = SAMPLING_TIME;
CTRL_PARAM.K = CONTROL_OUT.PARAM.PIT_K;
CTRL_PARAM.Ti = CONTROL_OUT.PARAM.PIT_Ti;
CTRL_PARAM.Td = CONTROL_OUT.PARAM.PIT_Td;
CTRL_PARAM.b = CONTROL_OUT.PARAM.PIT_b;
CTRL_PARAM.N = CONTROL_OUT.PARAM.PIT_N;
CTRL_PARAM.MAX_CONTROL = CONTROL_OUT.PARAM.PIT_MAX_CONTROL;
CTRL_PARAM.MIN_CONTROL = CONTROL_OUT.PARAM.PIT_MIN_CONTROL;
CTRL_PARAM.INT_DISC_TYPE =
CONTROL_OUT.PARAM.PIT_INT_DISC_TYPE;
CTRL_PARAM.DER_DISC_TYPE =
CONTROL_OUT.PARAM.PIT_DER_DISC_TYPE;
CTRL_PARAM.DER_INPUT = CONTROL_OUT.PARAM.PIT_DER_INPUT;
CTRL_PARAM.ANTIWINDUP = CONTROL_OUT.PARAM.PIT_ANTIWINDUP;
% PID CONTROL
CTRL_OUTPUT = PID(CTRL_INPUT,CTRL_PARAM);
% OUTPUT: ACCELERATION TARGET
PITCH_CONTROL = -CTRL_OUTPUT.CONTROL; %Mando invertido para
encajar con modelo (la k es negativa)

% STATE
PIT_INT_STATE = CTRL_OUTPUT.INT_STATE;
PIT_DER_STATE = CTRL_OUTPUT.DER_STATE;
otherwise
end
end
%% ROLL CONTROL: PID CONTROLLER
switch CONTROL_MODE
case 5 % Normal operation
% INPUTS
CTRL_INPUT.TARGET = single(0);
CTRL_INPUT.MEASUREMENT = single(EULER_ANG(1));
CTRL_INPUT.DERIVATIVE = single(EULER_RATE(1));
CTRL_INPUT.INT_STATE = ROLL_INT_STATE;
CTRL_INPUT.DER_STATE = ROLL_DER_STATE;
% PARAMETERS
CTRL_PARAM.Ts = SAMPLING_TIME;
CTRL_PARAM.K = CONTROL_OUT.PARAM.ROLL_K;
CTRL_PARAM.Ti = CONTROL_OUT.PARAM.ROLL_Ti;
CTRL_PARAM.Td = CONTROL_OUT.PARAM.ROLL_Td;
CTRL_PARAM.b = CONTROL_OUT.PARAM.ROLL_b;
CTRL_PARAM.N = CONTROL_OUT.PARAM.ROLL_N;
CTRL_PARAM.MAX_CONTROL = CONTROL_OUT.PARAM.ROLL_MAX_CONTROL;
CTRL_PARAM.MIN_CONTROL = CONTROL_OUT.PARAM.ROLL_MIN_CONTROL;

```

```

        CTRL_PARAM.INT_DISC_TYPE =
CONTROL_OUT.PARAM.ROLL_INT_DISC_TYPE;
        CTRL_PARAM.DER_DISC_TYPE =
CONTROL_OUT.PARAM.ROLL_DER_DISC_TYPE;
        CTRL_PARAM.DER_INPUT = CONTROL_OUT.PARAM.ROLL_DER_INPUT;
        CTRL_PARAM.ANTIWINDUP = CONTROL_OUT.PARAM.ROLL_ANTIWINDUP;
        % PID CONTROL
        CTRL_OUTPUT = PID(CTRL_INPUT,CTRL_PARAM);
        % OUTPUT: ACCELERATION TARGET
        ROLL_CONTROL = CTRL_OUTPUT.CONTROL;
        % STATE
        ROLL_INT_STATE = CTRL_OUTPUT.INT_STATE;
        ROLL_DER_STATE = CTRL_OUTPUT.DER_STATE;
    otherwise
end
end

%% POS_Z CONTROL: PID CONTROLLER
% switch CONTROL_MODE
%   case 5 % Normal operation
%       % INPUTS
%       CTRL_INPUT.TARGET = single()
%       CTRL_INPUT.MEASUREMENT = single(EARTH_POS_Z)
%       CTRL_INPUT.DERIVATIVE = single(EARTH_VEL_Z)
%       CTRL_INPUT.INT_STATE = POS_Z_INT_STATE;
%       CTRL_INPUT.DER_STATE = POS_Z_DER_STATE;
%       % PARAMETERS
%       CTRL_PARAM.Ts = SAMPLING_TIME;
%       CTRL_PARAM.K = CONTROL_OUT.PARAM.POS_Z_K;
%       CTRL_PARAM.Ti = CONTROL_OUT.PARAM.POS_Z_Ti;
%       CTRL_PARAM.Td = CONTROL_OUT.PARAM.POS_Z_Td;
%       CTRL_PARAM.b = CONTROL_OUT.PARAM.POS_Z_b;
%       CTRL_PARAM.N = CONTROL_OUT.PARAM.POS_Z_N;
%       CTRL_PARAM.MAX_CONTROL =
CONTROL_OUT.PARAM.POS_Z_MAX_CONTROL;
%       CTRL_PARAM.MIN_CONTROL =
CONTROL_OUT.PARAM.POS_Z_MIN_CONTROL;
%       CTRL_PARAM.INT_DISC_TYPE =
CONTROL_OUT.PARAM.POS_Z_INT_DISC_TYPE;
%       CTRL_PARAM.DER_DISC_TYPE =
CONTROL_OUT.PARAM.POS_Z_DER_DISC_TYPE;
%       CTRL_PARAM.DER_INPUT = CONTROL_OUT.PARAM.POS_Z_DER_INPUT;
%       CTRL_PARAM.ANTIWINDUP = CONTROL_OUT.PARAM.POS_Z_ANTIWINDUP;
%       % PID CONTROL
%       CTRL_OUTPUT = PID(CTRL_INPUT,CTRL_PARAM);
%       % OUTPUT: ACCELERATION TARGET
%       POS_Z_CONTROL = CTRL_OUTPUT.CONTROL;
%       % STATE
%       POS_Z_INT_STATE = CTRL_OUTPUT.INT_STATE;
%       POS_Z_DER_STATE = CTRL_OUTPUT.DER_STATE;
%   otherwise
% end

%% CONTROL MIXER

if (RAW_CH(5)>1500)
    %SERVO 1
    ESC_PWM(4) =
((PITCH_CONTROL+ROLL_CONTROL+POS_Z_CONTROL)*250)+1800;
    %SERVO 2
    ESC_PWM(5) = ((PITCH_CONTROL-
ROLL_CONTROL+POS_Z_CONTROL)*250)+1800;

```

```

    %SERVO 3
    ESC_PWM(6) = ((-PITCH_CONTROL-
ROLL_CONTROL+POS_Z_CONTROL)*250)+1800;
    %SERVO 4
    ESC_PWM(7) = ((-
PITCH_CONTROL+ROLL_CONTROL+POS_Z_CONTROL)*250)+1800;
else
    %SERVO 1
    ESC_PWM(4) = (((single(RAW_CH(4))-1500)+(single(RAW_CH(1))-
1500))*(250/600))+1800;
    %SERVO 2
    ESC_PWM(5) = (((single(RAW_CH(4))-1500)-(single(RAW_CH(1))-
1500))*(250/600))+1800;
    %SERVO 3
    ESC_PWM(6) = ((-(single(RAW_CH(4))-1500)-(single(RAW_CH(1))-
1500))*(250/600))+1800;
    %SERVO 4
    ESC_PWM(7) = ((-(single(RAW_CH(4))-1500)+(single(RAW_CH(1))-
1500))*(250/600))+1800;
end
%% CONTROL BUS UPDATE
% TARGET
% CONTROL_OUT.INPUT.EARTH_POS_TARGET = EARTH_POS_TARGET;
% CONTROL_OUT.INPUT.EARTH_VEL_TARGET = EARTH_VEL_TARGET;
% CONTROL_OUT.INPUT.EARTH_ACCEL_TARGET = EARTH_ACCEL_TARGET;
% CONTROL_OUT.INPUT.EULER_ANG_TARGET = EULER_ANG_TARGET;
% CONTROL_OUT.INPUT.EULER_RATE_TARGET = EULER_RATE_TARGET;
% CONTROL_OUT.INPUT.EULER_ACCEL_TARGET = EULER_ACCEL_TARGET;
% CONTROL_OUT.OUTPUT.BLDC.THRUST_FORCES = THRUST_FORCES;
% CONTROL_OUT.INPUT.THROTTLE_TARGET = THR_TARGET;

% CONTROL VARIABLES
CONTROL_OUT.OUTPUT.BLDC.ESC_PWM = ESC_PWM;

return

```

CHAPTER 5: STATE MACHINE UPDATE FUNCTION

This is the code for the simple state machine that controls the RC car.

```
function CONTROL_OUT = STATE_MACHINE(CONTROL_IN)

%% COPY CONTROL BUS
CONTROL_OUT = CONTROL_IN;

%% TIME UPDATE
% SAMPLING TIME
SAMPLING_TIME = CONTROL_OUT.PARAM.SAMPLING_TIME; % s
% 6-bit SAMPLING COUNTER
SAMPLING_COUNT = CONTROL_OUT.STATE.SAMPLING_COUNT;
SAMPLING_COUNT = SAMPLING_COUNT + uint8(1);
if SAMPLING_COUNT > uint8(63)
    SAMPLING_COUNT = uint8(0);
end
CONTROL_OUT.STATE.SAMPLING_COUNT = SAMPLING_COUNT;
% BOOTING TIME IN MILLISECONDS
TIME_BOOT_MS = CONTROL_OUT.STATE.TIME_BOOT_MS;
TIME_BOOT_MS = TIME_BOOT_MS + uint32(1000*SAMPLING_TIME);
CONTROL_OUT.STATE.TIME_BOOT_MS = TIME_BOOT_MS;
% UNIX TIME IN MICROSECONDS
TIME_UNIX_US = CONTROL_OUT.STATE.TIME_UNIX_US;
TIME_UNIX_US = TIME_UNIX_US + 1e6*double(SAMPLING_TIME);
CONTROL_OUT.STATE.TIME_UNIX_US = TIME_UNIX_US;
AUX = TIME_UNIX_US*2^32;
CONTROL_OUT.STATE.TIME_UNIX_US_LSB = uint32(AUX);
AUX = TIME_UNIX_US - double(uint32(AUX)*2^32);
CONTROL_OUT.STATE.TIME_UNIX_US_MSB = uint32(AUX);
TIMER = CONTROL_OUT.STATE.TIMER;

%% OPERATION MODES
CURRENT_STATUS = CONTROL_OUT.STATE.CURRENT_STATUS;
PREVIOUS_STATUS = CONTROL_OUT.STATE.PREVIOUS_STATUS;
% / 0. BOOTING / 1. SENSOR CALIBRATION / 2. SETTING OPERATION
HEIGHT
% / 3. STANDBY / 4. LOCKED MOTORS / 5. SYSTEM
OPERATION
% CONTROL_MODE:
% / 0. INITIALIZATION / 1. ESTIMATION / 2.
SETTING OPERATION POINT
% / 3. TESTING VELOCITY CONTROL / 4. VELOCITY CONTROL ONLY / 5. FULL
CONTROL
% / 6. MODEL TESTING
% EKF_MODE:
% / 0. NOT ENABLED / 1. PITCH, ROLL AND POS Z
EKF_MODE = CONTROL_OUT.STATE.EKF_MODE;
MOTOR_MODE = CONTROL_OUT.STATE.MOTOR_MODE;
% / 0. % SHUT DOWN / 1. ARMED MOTORS / 2. THROTTLE UNLIMITED /
SENSOR_CALIB_MODE = CONTROL_OUT.STATE.SENSOR_CALIB_MODE;
```

```

% / 0. UNCALIBRATED / 1. CALIBRATING / 2. CALIBRATED /

% INPUTS
SAFETY_SWITCH = CONTROL_OUT.INPUT.SAFETY_SWITCH;
RESET = CONTROL_OUT.INPUT.RESET;
STEERING = single(CONTROL_OUT.INPUT.RCRX.RAW_CH(3));
THROTTLE = single(CONTROL_OUT.INPUT.RCRX.RAW_CH(2));

% STATUS UPDATE
switch CURRENT_STATUS
%-----
-----
case 0 % BOOTING
if TIMER >= 3
    NEXT_STATUS = uint8(1); % SENSOR CALIBRATION
    TIMER = single(0);
else
    NEXT_STATUS = uint8(0); % CURRENT STATUS GOES ON
end
% UPDATE TIMER
TIMER = TIMER + SAMPLING_TIME;
% CONTROL MODE: INITIALIZATION
CONTROL_MODE = uint8(0);
% EKF MODE: NOT ENABLED
EKF_MODE = uint8(0);
% MOTOR MODE: SHUT DOWN
MOTOR_MODE = uint8(0);
% SENSOR CALIBRATION MODE: UNCALIBRATED
SENSOR_CALIB_MODE = uint8(0);
%-----
-----
case 1 % SENSOR CALIBRATION
if TIMER >= 30
    NEXT_STATUS = uint8(2); % SETTING OPERATION POINT
    % SENSOR CALIBTRIAON MODE: CALIBRATED
    SENSOR_CALIB_MODE = uint8(2);
    % TIMER RESET
    TIMER = single(0);
else
    NEXT_STATUS = uint8(1); % CURRENT STATUS GOES ON
    % SENSOR CALIBRATION MODE: CALIBRATING
    SENSOR_CALIB_MODE = uint8(1);
    % UPDATE TIMER
    TIMER = TIMER + SAMPLING_TIME;
end
% CONTROL MODE: INITIALIZATION
CONTROL_MODE = uint8(0);
% EKF MODE: NOT ENABLED
EKF_MODE = uint8(0);
% MOTOR MODE: SHUT DOWN
MOTOR_MODE = uint8(0);
%-----
-----
case 2 % SETTING OPERATION HEIGHT
if (CONTROL_OUT.INPUT.DIST_Z>0.051)
    NEXT_STATUS = uint8(3); % STANDBY
else
    NEXT_STATUS = uint8(2); % CURRENT STATUS GOES ON
end
% CONTROL MODE: SETTING OPERATION POINT

```

```

CONTROL_MODE = uint8(3);
% EKF MODE: PITCH AND ROLL
EKF_MODE = uint8(1);
% MOTOR MODE: ACTIVE
MOTOR_MODE = uint8(1);
%-----
-----
case 3 % STANDBY
if SAFETY_SWITCH
    NEXT_STATUS = uint8(4); % DISARMED MOTORS
else
    NEXT_STATUS = uint8(3); % CURRENT STATUS GOES ON
end
% CONTROL MODE: ESTIMATION
CONTROL_MODE = uint8(1);
% EKF MODE: PITCH AND ROLL
EKF_MODE = uint8(1);
% MOTOR MODE: SHUT DOWN
MOTOR_MODE = uint8(0);
%-----
-----
case 4 % LOCKED MOTORS
if THROTTLE == 1500 && STEERING == 1500
    NEXT_STATUS = uint8(5); % UNLOCKING MOTORS
else
    NEXT_STATUS = uint8(4); % LOCKED MOTORS
end
% CONTROL MODE: ESTIMATION
CONTROL_MODE = uint8(1);
% EKF MODE: EULER ANGLES
EKF_MODE = uint8(1);
% MOTOR MODE: SHUT DOWN
MOTOR_MODE = uint8(0);
%-----
-----
case 5 % SYSTEM OPERATION
if SAFETY_SWITCH
    NEXT_STATUS = uint8(5); % ARMED MOTORS
else
    NEXT_STATUS = uint8(3); % READY
end
if RESET
    NEXT_STATUS = uint8(0);
end
% MOTOR MODE: ACTIVE
MOTOR_MODE = uint8(1);
% CONTROL MODE: RUN (5)
CONTROL_MODE = uint8(5);
% EKF MODE: EULER ANGLES
EKF_MODE = uint8(1);
%-----
-----
%-----
-----
otherwise
    % NEXT STATUS
    NEXT_STATUS = uint8(3);
    % CONTROL MODE: ESTIMATION

```

```
CONTROL_MODE = uint8(0);
% EKF MODE: EULER ANGLES
EKF_MODE = uint8(1);
% MOTOR MODE: SHUT DOWN
MOTOR_MODE = uint8(0);

end

%% CONTROL BUS UPDATE
% UPDATE PREVIOUS STATUS
CONTROL_OUT.STATE.PREVIOUS_STATUS = CURRENT_STATUS;
% UPDATE CURRENT STATUS
CONTROL_OUT.STATE.CURRENT_STATUS = NEXT_STATUS;
% UPDATE OPERATION MODES
CONTROL_OUT.STATE.CONTROL_MODE = CONTROL_MODE;
CONTROL_OUT.STATE.EKF_MODE = EKF_MODE;
CONTROL_OUT.STATE.MOTOR_MODE = MOTOR_MODE;
CONTROL_OUT.STATE.SENSOR_CALIB_MODE = SENSOR_CALIB_MODE;
CONTROL_OUT.STATE.TIMER = TIMER;
```