E4 - Bola no Plano

universidade de aveiro
theoria poiesis praxis

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

# Phase III Report

## Eletrónica IV

Diogo Correia 76608

José Domingues 76328

TABLE OF CONTENTS

Page

Page

Page

# LIST OF FIGURES

Figure Page

Figure Page

# ABBREVIATIONS

UART - Universal asynchronous receiver transmitter

ADC - Analog to Digital Converter

PWM - Pulse Width Modulation

# Part I

# Phase 1

# 1. INTRODUCTION

## 1.1 PIC32

Our project, ball in the plane, consists on a system, based on a microcontroller, which controls a ball position on a plane surface according to a given setpoint. Considering that this system can only achieve the setpoint in an unstable equilibrium it is used a PID algorithm to control it.

In this report it is going to be explained the development

The microcontroller used for the project was the PIC32MX795F512L, with a 32-bit MIPS processor core running at 40MHz.

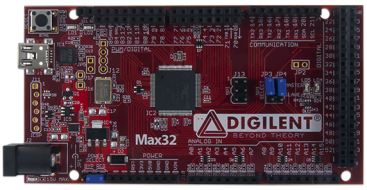



Figure 1.1. Max32 - Programmable PIC32 Microcontroller Board

The platform used to develop our code and program the microcontroller was MPLAB® X Integrated Development Environment (IDE) the in-system programming and debugging of applications was done using the chipKIT Programmer.

## 1.2 Original idea

The project consists on a system based on MCU, "Bola no plano", that controls a ball position on a plane surface. The desired ball setpoint is programmed by the PC, which is connected through a RS232C interface. This plane is coupled to two servo-motors whom allow to control its inclination according to two orthogonal axis. On the plane surface there is a touch


Figure 1.2. ChipKIT programmer

sensitive sensor, based on resistive technology, that allows to determine the ball position along both axis[1].

### 1.3 Add-on ideas

To complement the project, the ball setpoint will be able to be moved also with an analogical controller that will be made specifically for this purpose and will be after implemented with an wireless module, and also with an auxiliary touch panel, through which the user will be able to touch and indicate that way the desired setpoint of the ball.

### 1.4 Specifications of the project

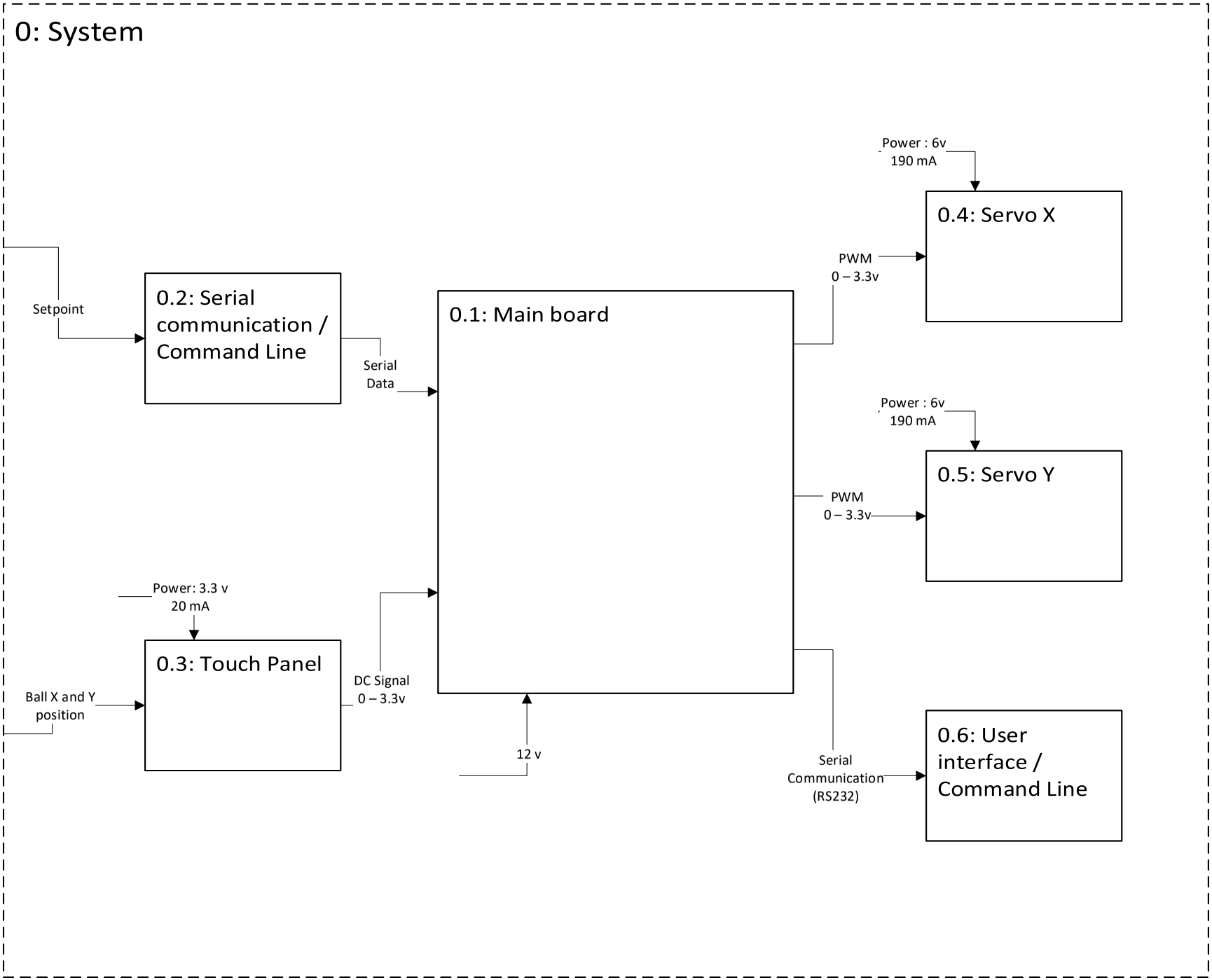
- Communication between the PC and the system via RS232C;
- Control area according to the physical dimensions of the sensor. The geometrical center of the plane will be on the origin;
- Setpoint resolution in mm;
- Ball position reading in mm ;
- Plane position control with a resolution equal or less than a degree;
- Ability to program the controller parameters through the PC.

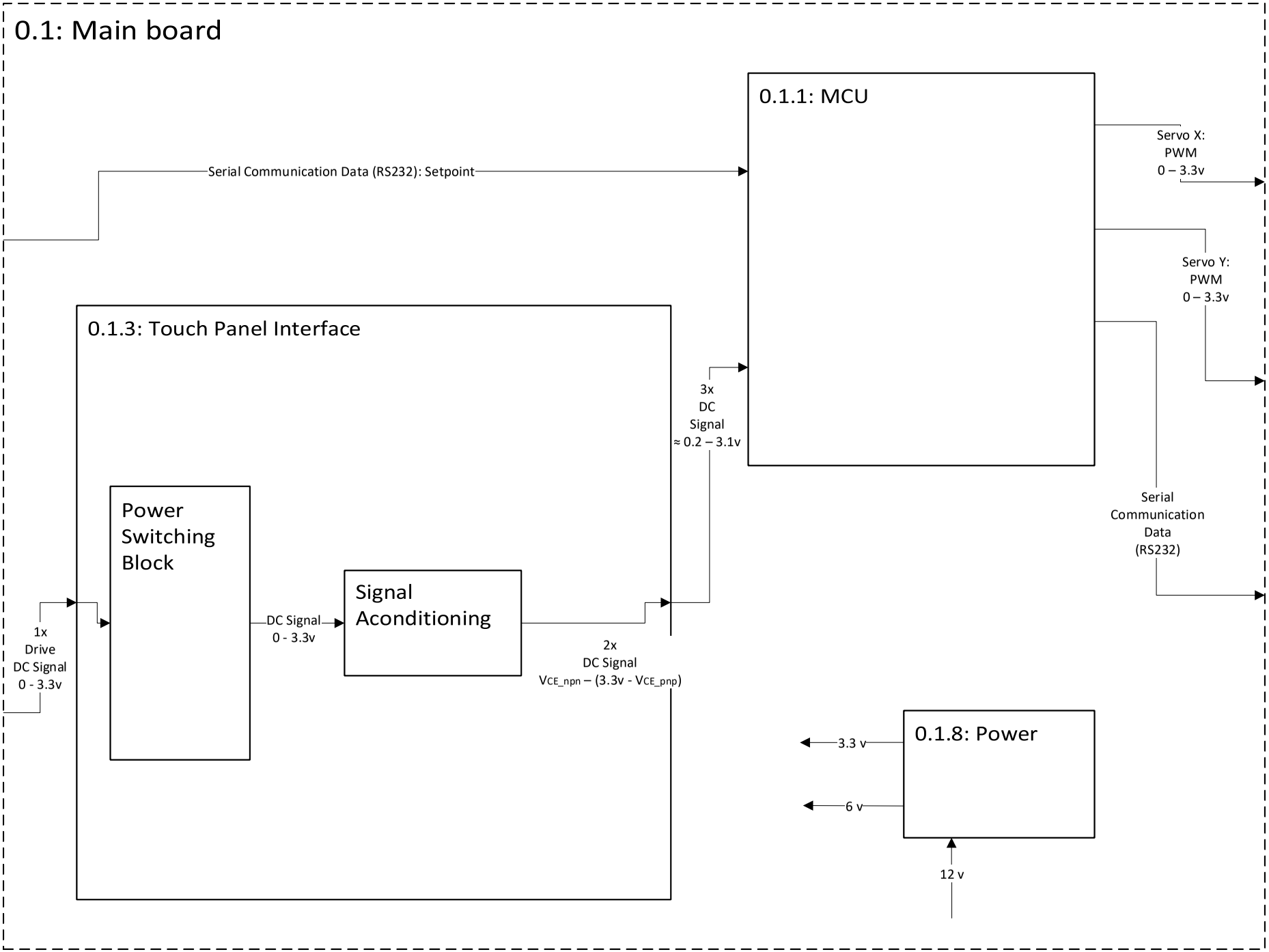
*Add-on specifications*:

- Ability to control the Setpoint through an analogical controller

- Ability to control the Setpoint through a secondary 4 Wire Touch Screen Panel with dimensions of 215x165mm, Unbranded/Generic

### 1.5 Block diagrams


0: System
Power : 6v
190 mA
0.4: Servo X
PWM
0 – 3.3v
Setpoint
0.2: Serial communication / Command Line
Serial Data
0.1: Main board
Power : 6v
190 mA
0.5: Servo Y
PWM
0 – 3.3v
Power: 3.3 v
20 mA
0.3: Touch Panel
Ball X and Y position
DC Signal
0 – 3.3v
12 v
Serial Communication (RS232)
0.6: User interface / Command Line


0.1: Main board
0.1.1: MCU
Serial Communication Data (RS232): Setpoint
Servo X:
PWM
0 – 3.3v
Servo Y:
PWM
0 – 3.3v
Serial
Communication
Data
(RS232)
0.1.3: Touch Panel Interface
3x
DC
Signal
≈ 0.2 – 3.1v
Power
Switching
Block
Signal
Aconditioning
1x
Drive
DC Signal
0 - 3.3v
DC Signal
0 - 3.3v
2x
DC Signal
VCE_npn – (3.3v - VCE_pnp)
0.1.8: Power
3.3 v
6 v
12 v

## 1.6 Work planning

In this section it will be presented the PERT chart alongside with the Gantt chart, these dates were the suggested on the beggining of the project.

The dates were partial completed in time. Due to delays in components supply the touch-screen circuit, touchscreen drivers and dependents tasks suffered a delay of one and half week. The current state of the project is as showed in the Gant graph of the figure 1.3.

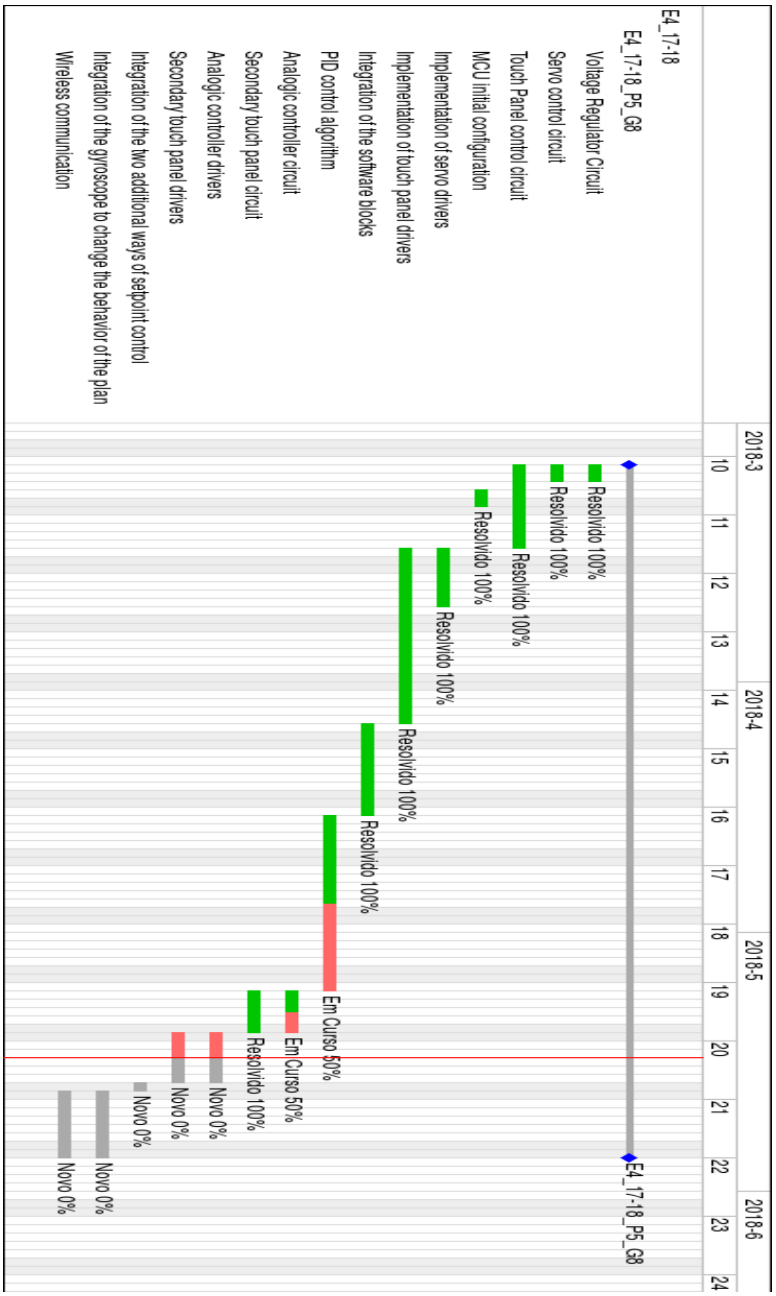
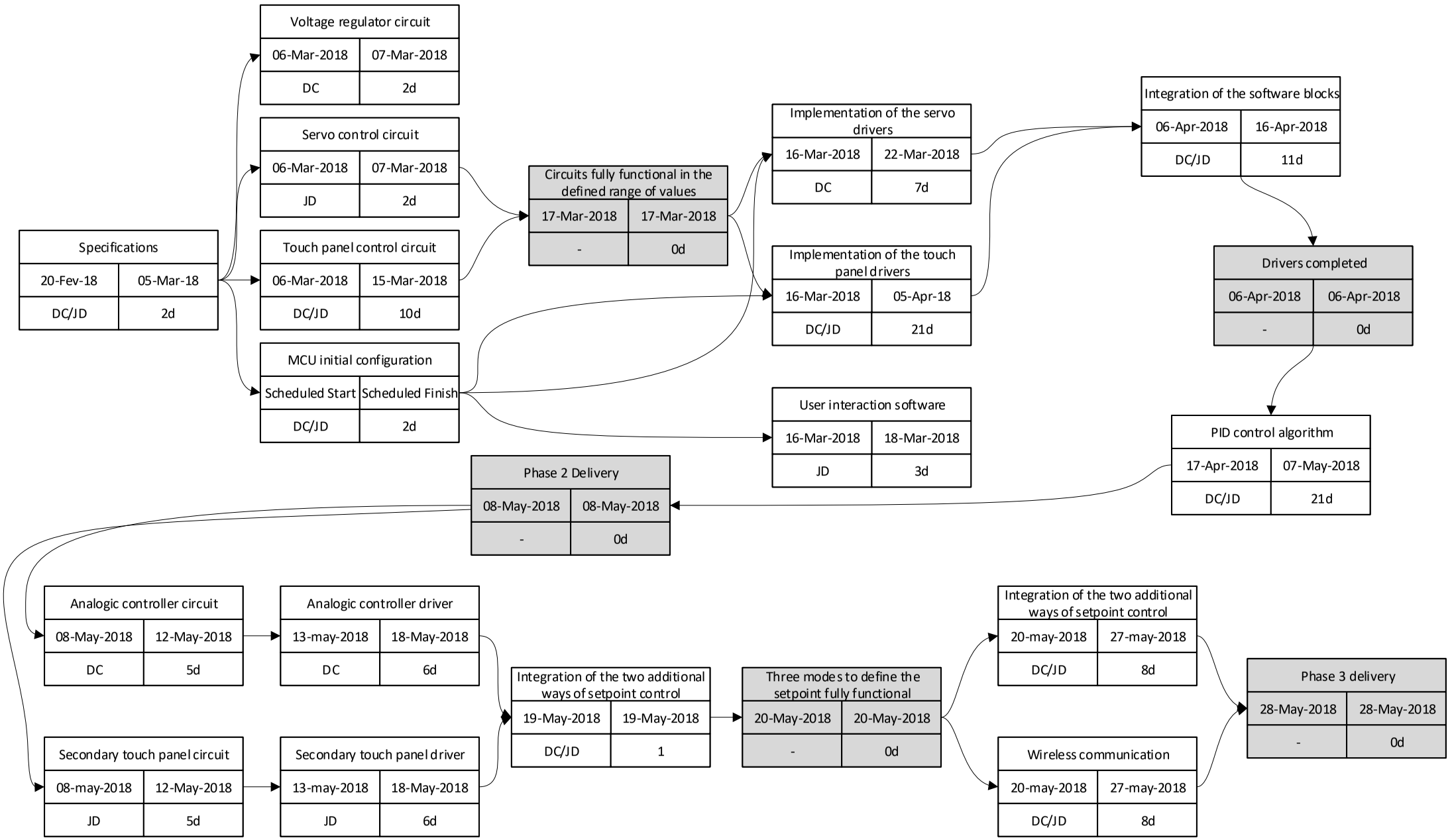



Figure 1.3. Current Gant progress

# PERT Chart

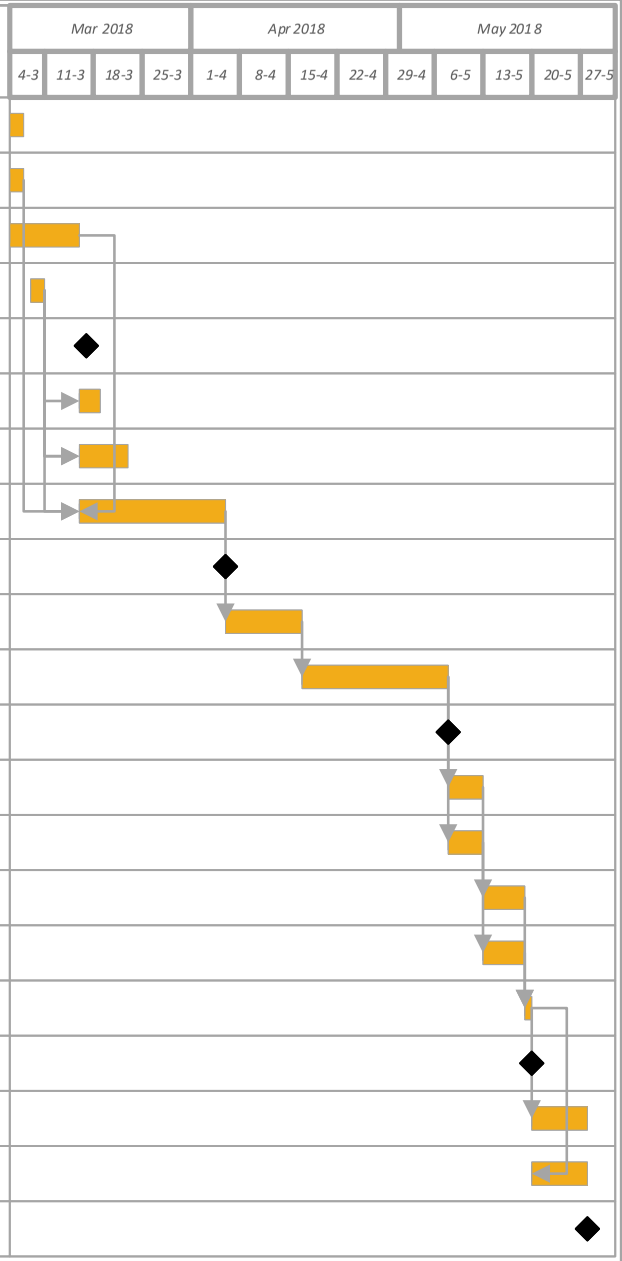
**March 6, 2018**



# Gantt chart

**March 6, 2018**

| ID | Task Name | In charge | Start | Finish | Duration |
|---|---|---|---|---|---|
| 1 | Voltage regulator circuit | Diogo Correia | 06-Mar-18 | 07-Mar-18 | 2d |
| 2 | Servo control circuit | José Domingues | 06-Mar-18 | 07-Mar-18 | 2d |
| 3 | Touch panel control circuit | Diogo Correia<br>José Domingues | 06-Mar-18 | 15-Mar-18 | 10d |
| 4 | MCU initial configuration | Diogo Correia<br>José Domingues | 09-Mar-18 | 10-Mar-18 | 2d |
| 5 | Circuits fully functional in the Defined range of values | - | 17-Mar-18 | 17-Mar-18 | 0d |
| 6 | User interaction software | José Domingues | 16-Mar-18 | 18-Mar-18 | 3d |
| 7 | Implementation of the servo drivers | Diogo Correia | 16-Mar-18 | 22-Mar-18 | 7d |
| 8 | Implementation of the touch panel drivers | Diogo Correia<br>José Domingues | 16-Mar-18 | 05-Apr-18 | 21d |
| 9 | Drivers complete | | 06-Apr-18 | 06-Apr-18 | 0d |
| 10 | Integration of the software blocks | Diogo Correia<br>José Domingues | 06-Apr-18 | 16-Apr-18 | 11d |
| 11 | PID control algorithm | Diogo Correia<br>José Domingues | 17-Apr-18 | 07-May-18 | 21d |
| 12 | Phase 2 delivery | - | 08-May-18 | 08-May-18 | 0d |
| 13 | Analogic controller circuit | Diogo Correia | 08-May-18 | 12-May-18 | 5d |
| 14 | Secondary touch panel circuit | José Domingues | 08-May-18 | 12-May-18 | 5d |
| 15 | Analogic controller drivers | Diogo Correia | 13-May-18 | 18-May-18 | 6d |
| 16 | Secondary touch panel drivers | José Domingues | 13-May-18 | 18-May-18 | 6d |
| 17 | Integration of the two additional ways of setpoint control | Diogo Correia<br>José Domingues | 19-May-18 | 19-May-18 | 1d |
| 18 | Three modes to define the setpoint fully functional | - | 20-May-18 | 20-May-18 | 0d |
| 19 | Integration of the gyroscope to change the behavior of the plan | Diogo Correia<br>José Domingues | 20-May-18 | 27-May-18 | 8d |
| 20 | Wireless communication | Diogo Correia<br>José Domingues | 20-May-18 | 27-May-18 | 8d |
| 21 | Phase 3 delivery | - | 28-May-18 | 28-May-18 | 0d |



# Part II

# Phase 2

# 2. PERIPHERALS

## 2.1 Resistive Touch Screen

FTAS00-104AS4 is a 4-wire Resistive Touch Screen. Resistive Touch Screens locate the coordinates where something touches its surface. In this project its application is to read the coordinates of a steel ball placed on its surface.


Figure 2.1. FTAS00-104AS4 - 4 Wire Touch Screen (Source: NKK Switches - Resistive Touch Screens)

### 2.1.1 Operation Principle

This touch screen has two resistive layers that do not touch themselves while there in rest, one for the Y axis and another for the X axis. It works when there is applyed a voltage across the vertical or horizontal network. A measurement of the Y position of the pointing device can be made by turning on the Y+ and Y-, to 3.3V and GND respectively, and reading the measure

taken from the point of contact of the two layers in the input X+ with a ADC module. For a measurement of the X axis the inverse procedure can be applyed.

### 2.1.2 General Specifications

| | **Eletrical** |
|---|---|
| **Power Level** | 1mA @ 5V DC(resistive load) |
| **Insulation Impedance** | 10MΩ minimum @ 25V DC |
| **Linearity** | 3% Maximum (analog) |
| | **Mechanical** |
| **Touch Activation Force** | 1.4N maximum |
| **Key Area Dimensions** | 211.2mm x 158.4mm |
| **External Dimensions** | 225.6mm x 171.4mm |

Table 2.1.
General specifications for the project resistive touch screen (source: NKK Switches - Resistive Touch Screens)

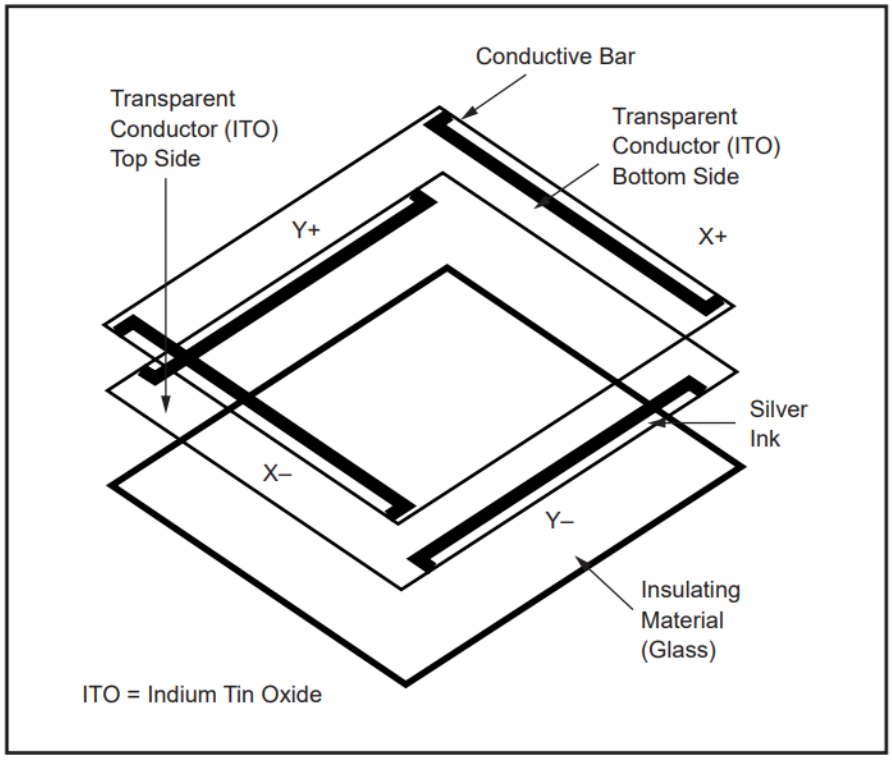



Figure 2.2. 4 Wire Touch Screen construction (Source: BURR-BROWN TOUCH SCREEN CONTROLLER TIPS)

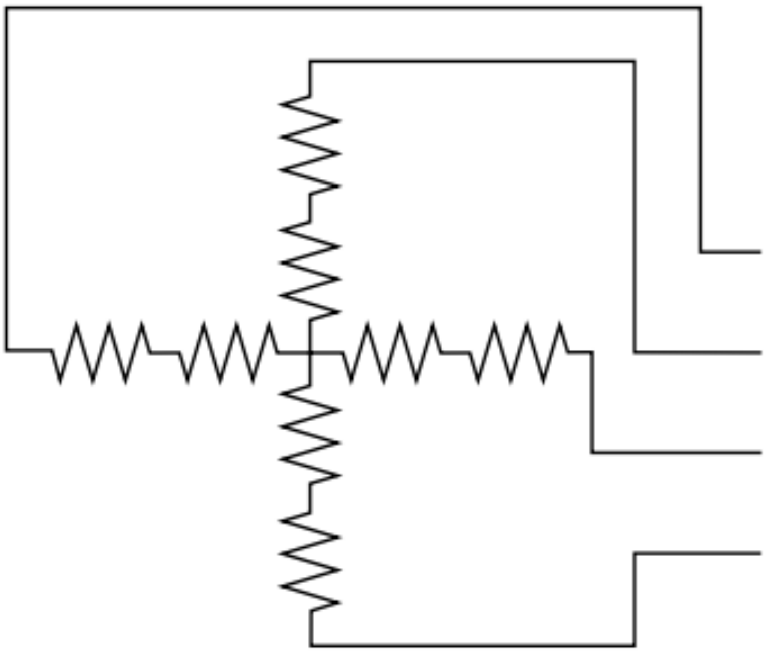


Figure 2.3. 4 Wire Touch Screen Circuits (Source: BURR-BROWN TOUCH SCREEN CONTROLLER TIPS)

### 2.1.3 Linearization

The Touchscreen linearization points were measured with the touchscreen drive and reading circuit obtaining the results in figures 2.4 e 2.5.

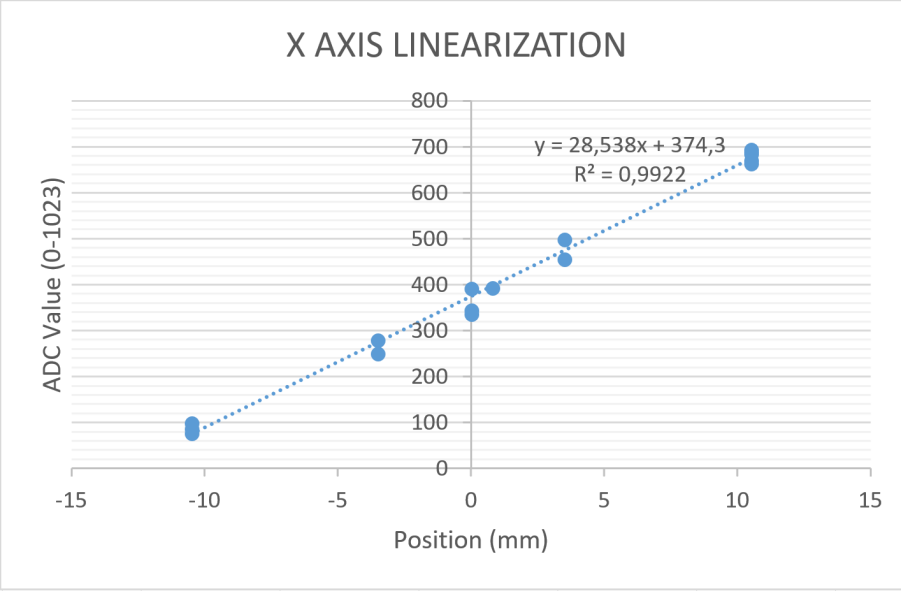



Figure 2.4. Linearization of touchscreen X axis

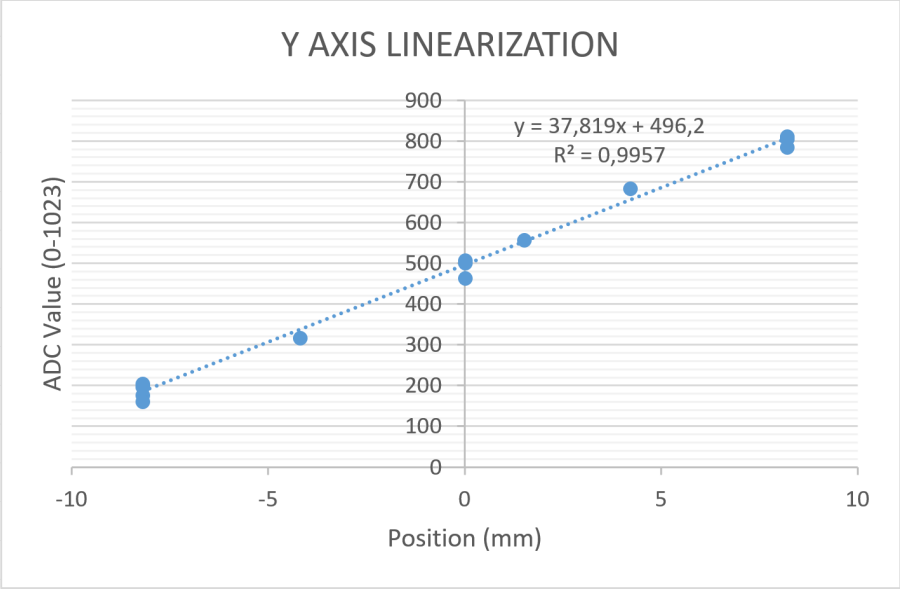



Figure 2.5. Linearization of touchscreen Y axis

## 2.2 Servos

For the project setup there were used two Parallax Standard Servos (900-00005) in order to provide the inclination of the plane and make the ball reach its setPoint. These servos have good precision that makes them desirable for our project.


Figure 2.6. Parallax Standard Servo (Source: PARALLAX Standard Servo 900-00005 Datasheet)

### 2.2.1 Operation Principle

A servo is characterized by lack of speed but good torque and a precise angle. These are characterized for having a feedback loop that controls the angle of the servo. In order to have more torque and precision the speed is lowered by gears. They receive a PWM signal that will correspond to a certain angle output and will apply it to the motor in order to get that position.

### 2.2.2 General Specifications

- Power requirements: 4 to 6 VDC; Maximum current draw is 140 ± 50 mA at 6 VDC when operating in no load conditions, 15 mA when in static state.
- Communication: PWM, 0.75-2.25 ms high pulse, 20 ms intervals

| PIN | Name | Description | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| **1(White)** | Signal | Input;TTL or CMOS | 3.3 | 5.0 | Vservo + 0.2 | V |
| **2(Red)** | 10MΩ minimum @ 25V DCVservo | Power suplly | 4.0 | 5.0 | 6.0 | V |
| **3(Black)** | Vss | Ground | | 0 | | V |

Table 2.2.
General specifications for the project servos (source: Parallax Standard Servo DataSheet)

# 3. BASE CONFIGURATION

The chipKIT Max32 has a primary oscillator with frequency of 8MHz, that can be multiplied until 80MHz depending on the project specification demands. For our project we defined that the system clockwould have a frequency of 40MHz and the peripheral clock of 20MHz. In order to set these values we recurred to a Configuration Bits built-in GUI in MPLAB which allows us to set the bits referring to the oscillator block. All the configuration bits were generated to code and implemented as library "xc32.h"

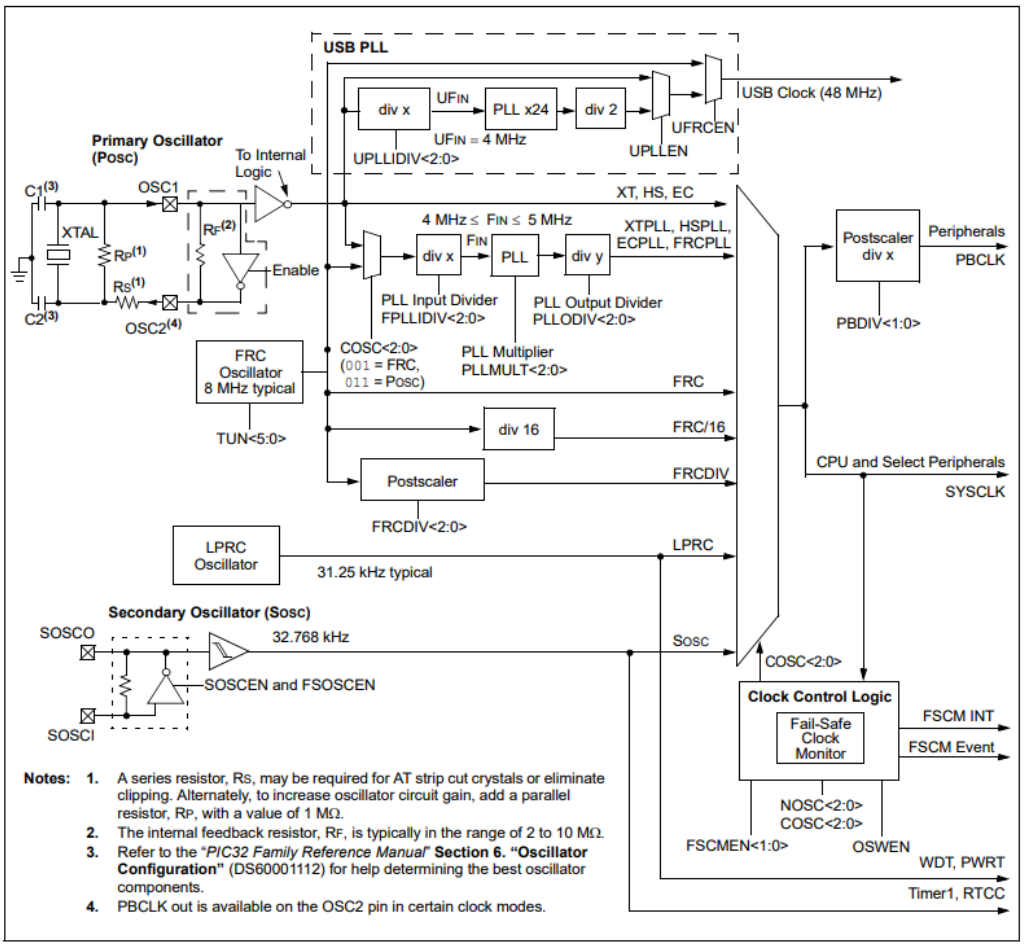



Figure 3.1. Oscillator Block Diagram (Source: PIC32MX5XX/6XX/7XX Family Data Sheet)

As it can be seen from the image, starting from the primary oscillator at 8MHz, with the phase-locked loop multiplying the frequency to 40MHz, it makes the SYSCLK(System Clock) to have a 40MHz frequency while the PBCLK(Peripheral Bus Clock) that is divided by a postscaler of 2 sets at 20Mhz.

# 4. UART - UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER

## 4.1 Operation principle

The system has requirements imposing that the setPoint value is defined by the user via computer and has to return the information from the microcontroller to the computer, which use RS232 as communication protocol. This protocol is implemented by the UART1 module available in the microcontroller, which is converted to USB by an FTDI. More details on the UART module are provided in the UART reference manual **here**

The UART uses NRZ format (non-return-to-zero), with one Star bit, eight or nine data bits, configurable by the user in the init function. It has parity, either even or odd. The UART module is set to 9600 baudRate, a value more than enough to the project needs. It is working currently by polling.


Figure 4.1. FTDI of MAX32

## 4.2 Configuration

The UART configuration is presented in the function *uartInit*, presented in *uartLib.h*. It takes 3 attributes: *baudrate*; *parity* - that can function in 4 modes, being them 8-bit data no parity, 8-bit data even parity, 8-bit data odd parity and 9-bit data no parity; and *stopbits* - than can be either 1 or 2 bits. This function starts with the calculation of U1BRG, taking in count the decimal round, going then in two switch cases that configure the bits in accord with the

attributes *parity* and *stopbits* passed to the function, enabling the transmitter module, receiver module and UART at the end. There was no need to use other advance features of UART, so they were initialized off.

### 4.3 Device Drives

The UART library, *uartLib.h*, comes with 3 functions been them:

- *putChar* : sends a char to UART;
- *getChar* : receives char from UART;
- *putString* : sends a string througth UART with an end of line character at the end.
- *printInt10* : sends an Integer in base 10
- *getCoordinatesXX* : returns an integer with the setPoint coordinates in the XX axis specified by the user
- *getCoordinatesYY* : returns an integer with the setPoint coordinates in the YY axis specified by the user

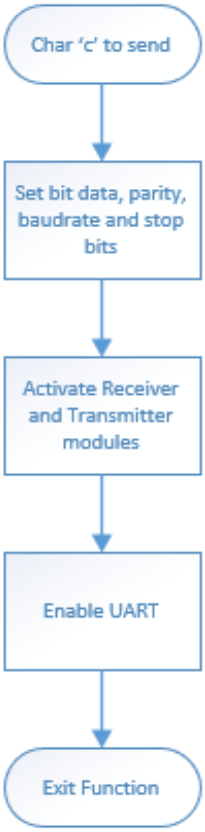



Figure 4.2. putChar high-level fluxogram

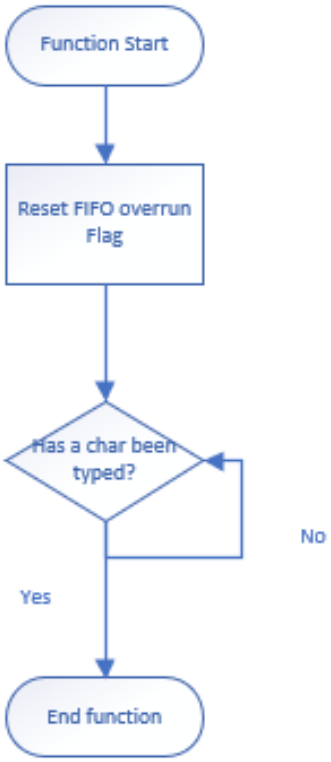



Figure 4.3. getChar high-level fluxogram

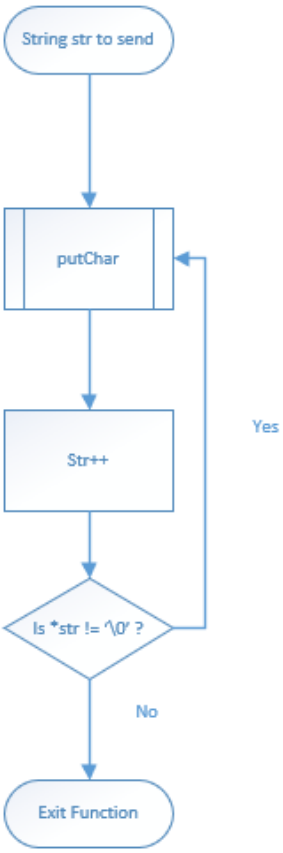



Figure 4.4. putString high-level fluxogram

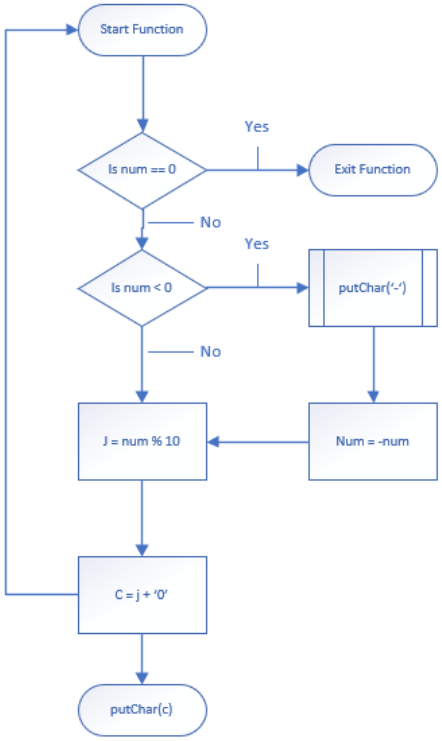



Figure 4.5. printInt10- high-level fluxogram

## 4.4 Tests

In order to test the sturdiness of the module it was made a series of tests, using the software *putty* for this purpose, below there are print screens made of the communication between the UART and the PC, where the function of the test is to ask for the setPoint coordinates both in XX axis and YY axis, and save them into variables.

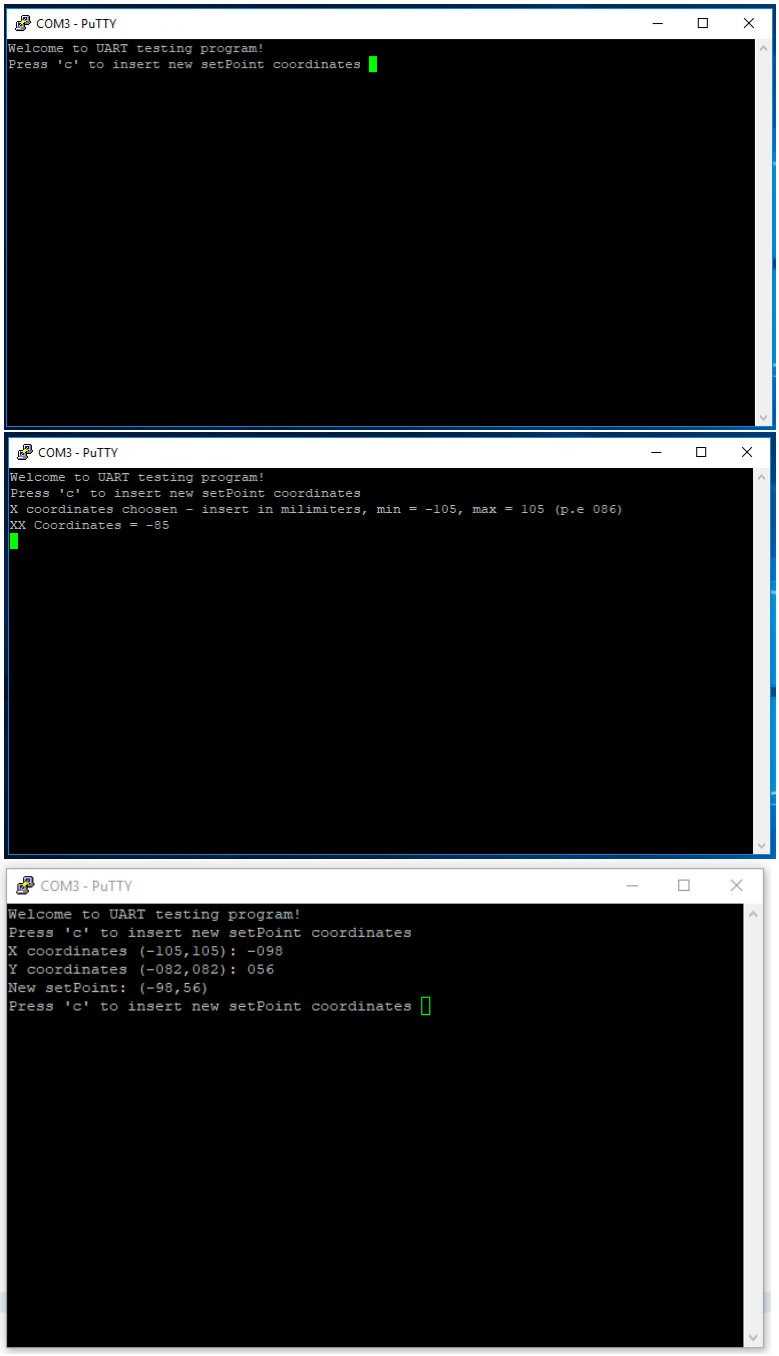



Figure 4.6. UART test

# 5. PWM - PULSE WIDTH MODULATION

## 5.1 Operation Principle

PWM is a technique used in many applications to encode information, control power supplied to devices like motors, loads and specially servos, that will be focus of this application.

In order to produce a precise PWM signal the max32 development board has a output compare module used primarily to generate a pulse/pulses in response to selected time base events. The diagram of the output compare module can be seen in figure 5.1

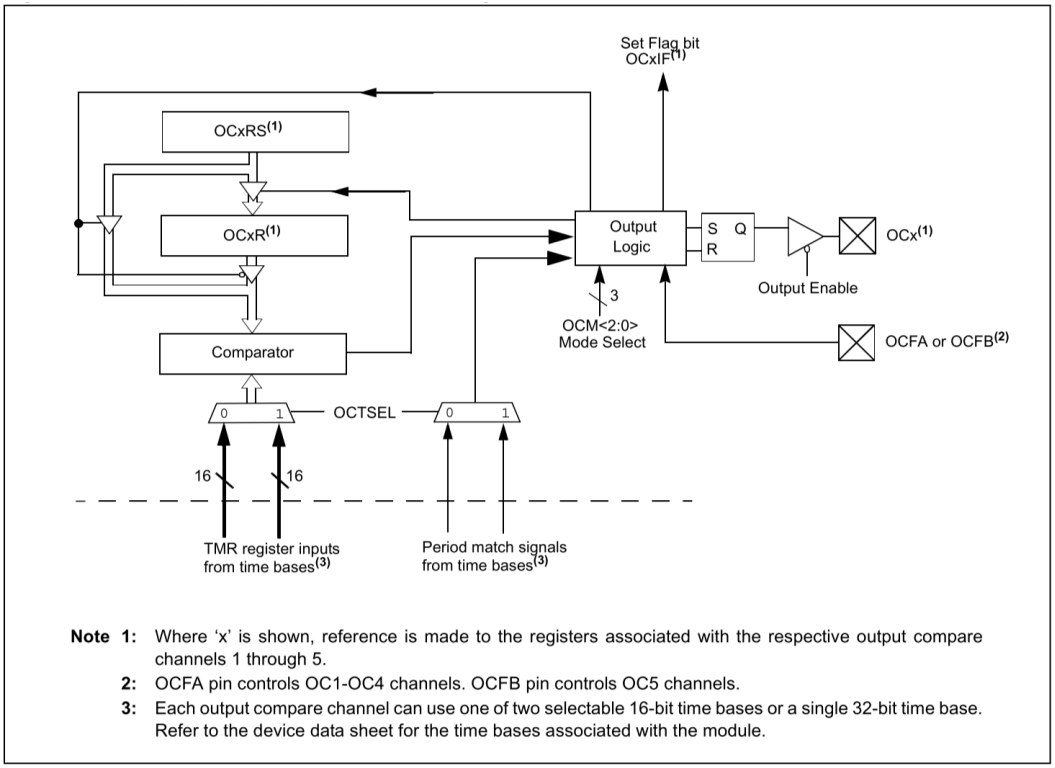



Figure 5.1. Output Compare Module Block Diagram [from PIC32 Family Reference Manual]

## 5.2 Configuration

To generate the events to trigger the output compare module we used the Timer2, a type B timer, provided in the PIC Micro Controller. The block diagram can be seen in figure 5.2.

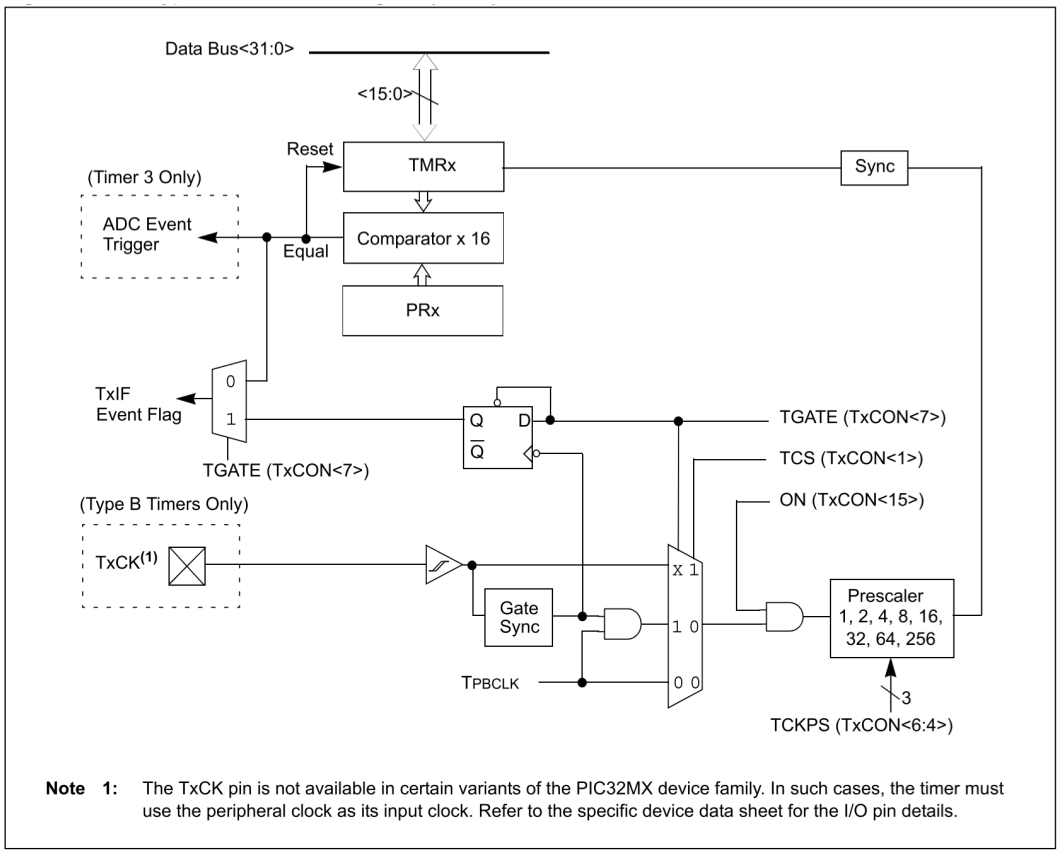



Figure 5.2. Type B Timer Block Diagram (16-Bit) [from PIC32 Family Reference Manual]

Since the servo as to receive a pulse every $25ms$ we can configure the timer2 with the proper parameters were with a prescale value of 1:32 and a PR2 of 12500 we can archive the pretended period within 16bit as can be seen in equation 5.1.

$$PR2 = \frac{\frac{20e10^6}{32}}{50} = 12500 < 2^{16} - 1 \tag{5.1}$$

With the timer configured is now possible to define the PWM $t_{ON}$. The servo datasheet refers, as we can see in figure 5.3, the minimum, center and maximum $t_{ON}$ values.

| BASIC Stamp Module | 0.75 ms | 1.5 ms (center) | 2.25 ms |
|---|---|---|---|
| BS1 | 75 | 150 | 225 |

Figure 5.3. Positions $t_{ON}$ values [from Parallax Standard Servo Datasheet]

With those values is possible to calculate the correspondent PWM in percentage and correspondent OCXRS has showed in equation 5.2 and results in the table in figure 5.4.

$$OCXRS = (12500 + 1) * PWM\% \tag{5.2}$$

| | Minimum | Center | Maximum |
|---|---|---|---|
| PWM (%) | 3.75% | 7.5% | 11.25% |
| OCXRS | 468.8 | 937.5 | 1406.3 |

Figure 5.4. Table with the results of PWM configuration

Knowing the minimum and maximum values of OCXRS and verifying that the servo position is linear with $t_{ON}$ is possible to change the OCXRS linearly between the minimum and the maximum. We decided to change it between 0 and 100%, being 0% the minimum position possible and 100% the maximum position possible. The linear equation to set the servo position follows equation 5.3.

$$OCXRS = (93758 * POSITION\% + 4690000)/10000 \tag{5.3}$$

The PWM pins are referenced in the max32 development board as pin 3 (for y axis) and 5 (for x axis).

All PWM configuration is done with the function *initPWM* presented in *pwmLib*.

## 5.3 Features

The PWM library, *pwmLib.h*, comes with 3 functions:

- *void pwmInit*: Initialize PWM/Timer Configuration Bits an Starts PWM for both Servos;
- *void setPWM(servo, pwm)*: Sets selected servo position;
- *uint*16_*t getPWM(servo)*: Returns selected servo actual setpoint position.

## 5.4 Device Drives

For more detail about the device driver, consult the doxygen and source code commented in code.ua repository.

### 5.4.1 pwmInit

The *pwmInit* implements the configuration described in the Configuration section. As can be seen in the fluxogram in figure 5.5, the function starts by configuring the timer 2 bits, the prescaler and the load period register (PR2), following by configuration of the output compare module, starting by choosing PWM mode, configuring the timer source and $t_{ON}$ constant. It end with a the enabling of the output compare peripheral.

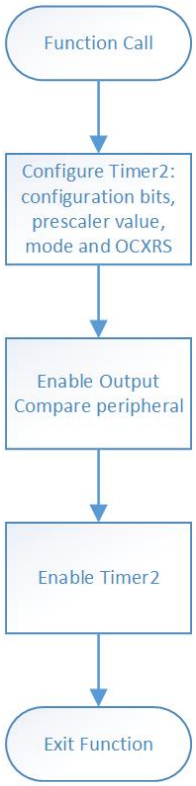



Figure 5.5. High level fluxogram of *initPWM*

### 5.4.2 setPWM

The function *setPWM* only asserts the pwm atribute value to prevent a $t_{ON}$ not supported by the Parallax default servo that could cause damage to it. Then it sets the OCXRS for the chosen servo has showed in equation 5.3 in the section Configuration.


Figure 5.6. High level fluxogram of *setPWM*

### 5.4.3 getPWM

The function *getPWM* only return a static variable presented in *pwmLib* that stores que current PWM for each of the servos.

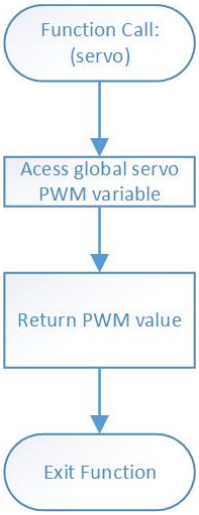



Figure 5.7. High level fluxogram of *getPWM*

## 5.5 Tests

To test the PWM library, was created a function that can increment and decrement both pwm axis through UART *getChar()*. Char 'w' and 's' increments and decrements y's axis pwm and 'a' and 'd' increments and decrements x's axis pwm.

With this test is possible to calibrate both axis since between the servo and the touchscreen plane there's a mechanical connection that introduces a variation between the angle of the servo and the real angle of the plane.

The PWM for the minimum and maximum values were measured, validating the implementation. The result can be seen in figures 5.8 and 5.9.

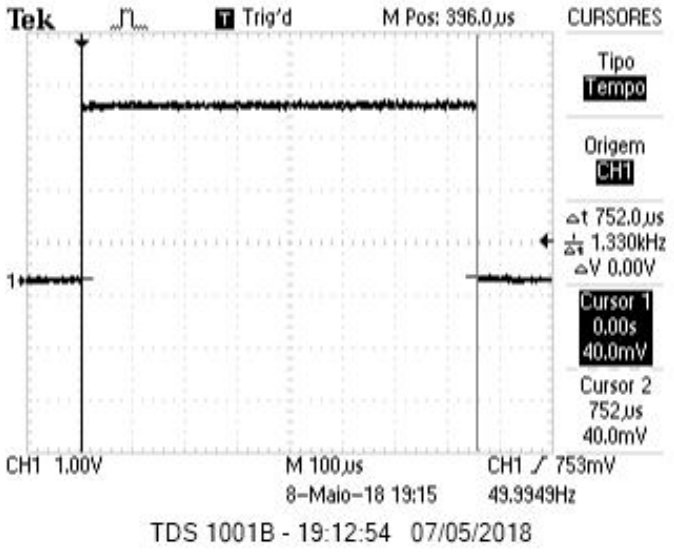



Figure 5.8. Practical results of the minimum servo PWM signal

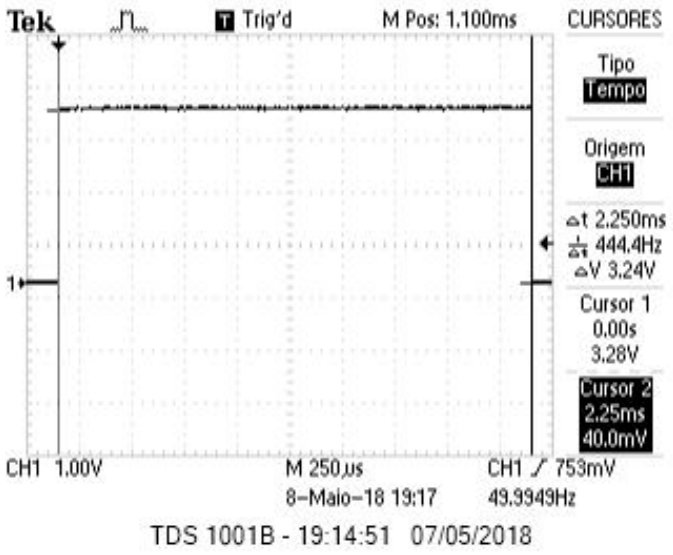



Figure 5.9. Practical results of the maximum servo PWM signal

# 6. ADC - ANALOG TO DIGITAL CONVERTER

## 6.1 Operation Principle

Because in the real world the signals are in analog forms there is the need to use an ADC to convert these to digital values, in order for the microprocessor to process and take decisions according to these inputs.

### 6.1.1 Successive approach method

This is the most used technique, it uses binary search. It consists of a high speed comparator, DAC(digital to analog converter), and control logic.

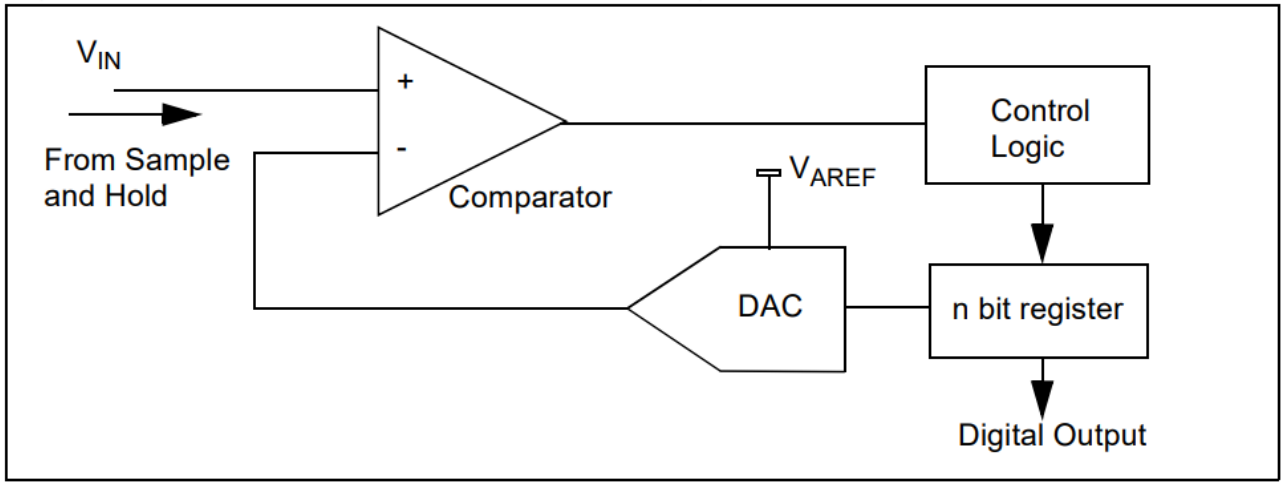



Figure 6.1. Successive Approximation

The Sucessive approach method consists in forcing the most significant bit high, and the DAC converts it to $V_{AREF}/2$. Then the input voltage is compared to $V_{AREF}/2$. If the input voltage is greater than the voltage corresponding to the most significant bit, the bit is left set, otherwise it is reset. [4]

This procedure is done for all the bits, and in the end we get the digital value that corresponds to the analog input.

In our project we use two analog inputs, from the same ADC, and because of that, it is needed an Analog multiplexer to switch the channels by software and convert each one individually.

### 6.2 Configuration

In our project it was decided for the ADC to perform sampling only on command, when the SAMP bit was set to 1, which was always cleared automatically to end sampling and start conversion. This way we have more control over the module.

The acquisiton time is set by the SAMC bits, which make it take 20*$T_{AD}$, i.e, $20 * T_{PB} * 2$ = 20*100ns, where $T_{PB}$ is the peripheral clock period ($\frac{1}{20MHz}$), and the *2 is a result of the ADCS bits set to 0.

Since the ADC requires one ADC clock cycle to convert each bit of the result, plus two additional clocl cycles [**?**], the conversion time will take 12 $T_AD$ cycles to perform the conversion, and place the result in the ADC buffer.

The sample time is given by the acquisition time ($20T_{AD}$) and the conversion time ($12T_{AD}$), which results in a 32*100ns sample time.

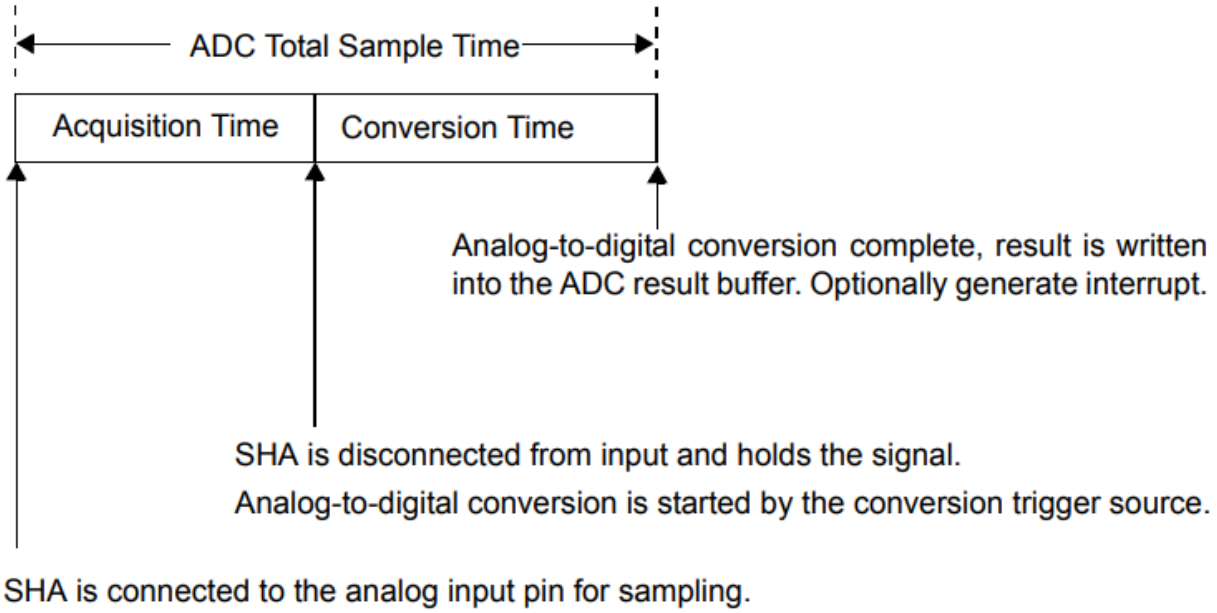



Figure 6.2. ADC Sample Sequence(Note: SHA - Sample and Hold Amplifier), source: PIC32 - ADC Reference Manual


Figure 6.3. initADC high-level fluxogram

## 6.3 Device Driver

For more detail about the device driver, consult the doxygen and source code commented in code.ua repository.

The ADC library, *uartLib.h*, comes with 4 functions being them:

- *readVoltageXX* : Returns the voltage read by the Analog pin of the XX axis (AN0);
- *readVoltageYY* : Returns the voltage read by the Analog pin of the YY axis (AN1);
- *readXX* : Returns the XX coordinates of the object on the touch screen;
- *readYY* : Returns the YY coordinates of the object on the touch screen.

### 6.3.1 ReadVoltageXX and ReadVoltageYY

This function was used for initial debugging of the ADC module, where in order to quickly check the veracity of the measurements we compared them with an oscilloscope to see if they were simillar.

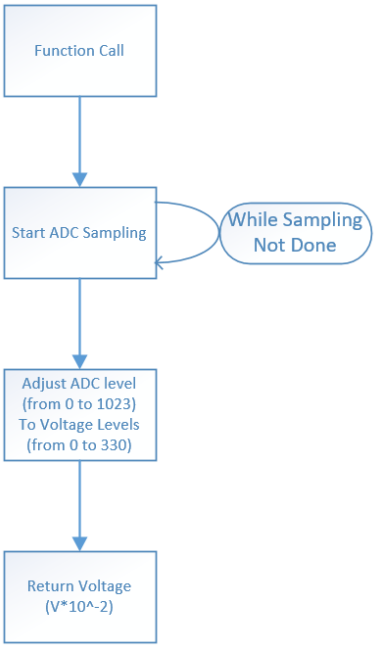



Figure 6.4. ReadVoltageXX and ReadVoltageYY high-level fluxogram

### 6.3.2 ReadXX

This function serves to measure the current position of the ball in the plane, with the center of the plane being the origin of both axis. At a first approach this function was defined to set the drive pin to 1 in order for the circuit in 7.2 to set the X+ and X- to 3.3V and GND respectively, and right after that, the ADC would sample through Y-, returning the value after properly treated to give the distance.

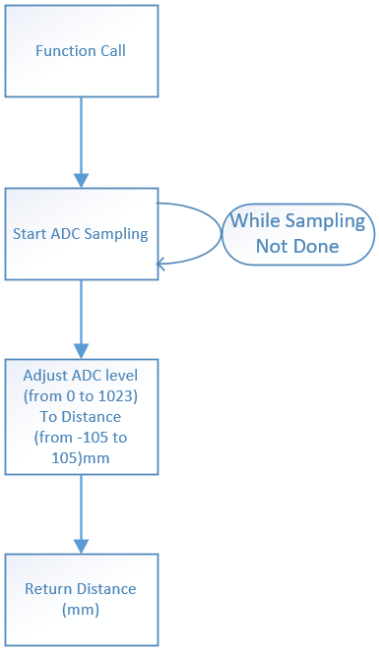



Figure 6.5. ReadXX high-level fluxogram, first iteration

Because this method was prone to errors due to the fact that the circuit in itself couldn't drive the screen panel fast enough before the ADC sampling, it was decided to impose a delay right after setting the Drive Pin, and only after the delay would the ADC sample the value.

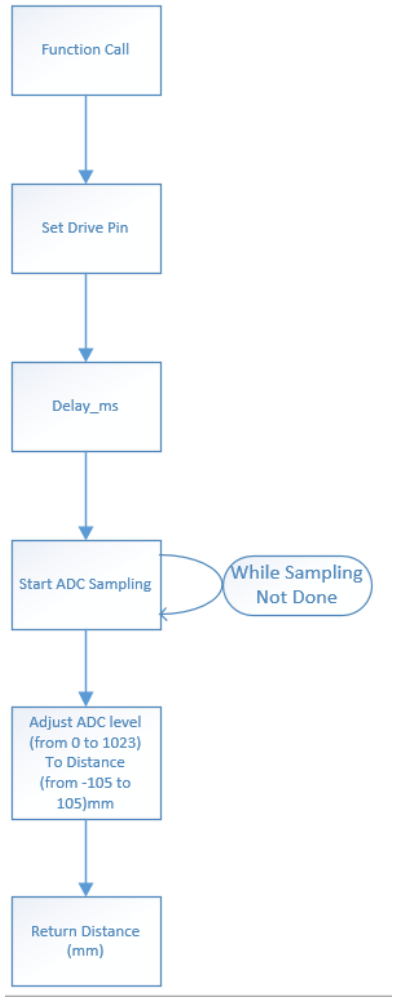



Figure 6.6. ReadXX high-level fluxogram, second iteration

#### 6.3.2.1 Resolution

Knowing that the touch panel in the X axis, i.e, the bigger side, has 210mm and the range of levels of the ADC is , maxLevel = 833 and minLevel = 340 ,the resolution will be $\frac{210mm}{833-340} = 0.436$

### 6.3.3 ReadYY

This function is used almost like the the one above, with the exception that the Drive pin is set to 0 instead of 1, in order to read the Y axis. And the linear regression that was made is different in order to return the proper values because the physical dimensions are smaller in the Y axis.

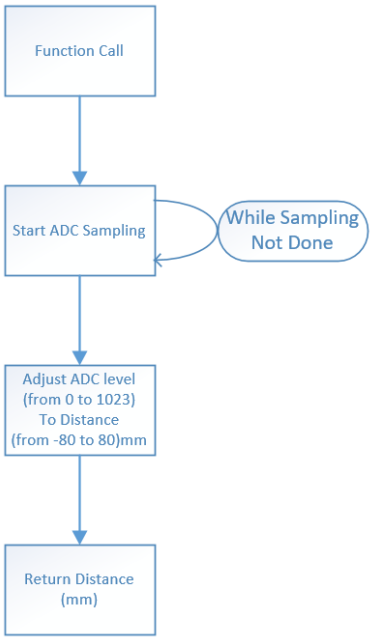



Figure 6.7. ReadYY high-level fluxogram, first iteration

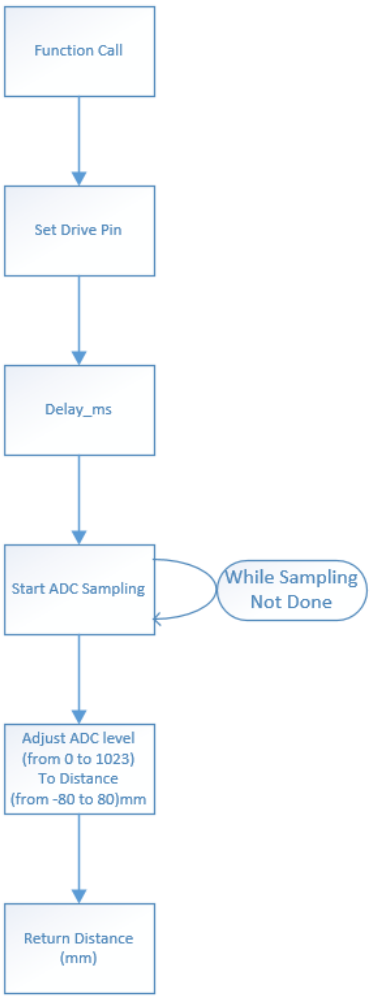



Figure 6.8. ReadYY high-level fluxogram, second iteration

#### 6.3.3.1 Resolution

The resolution of the YY axis is given the same way as the XX one, except for the fact that the physical length is smaller, 165mm, and the ADC levels are different: maxLevel = 220 and minLevel = 654 ,the resolution will be $\frac{210mm}{654-220} = 0.380mm$

### 6.3.4 Device driver tests

#### 6.3.4.1 Potentiometer

This was the first test made to test the ADC and to conclude if the values being taken were accurate enough. In this simple test each analog pin, AN0 and AN1 had a potentiometer connected to them, serving has a resitive divider between 3.3V and GND.

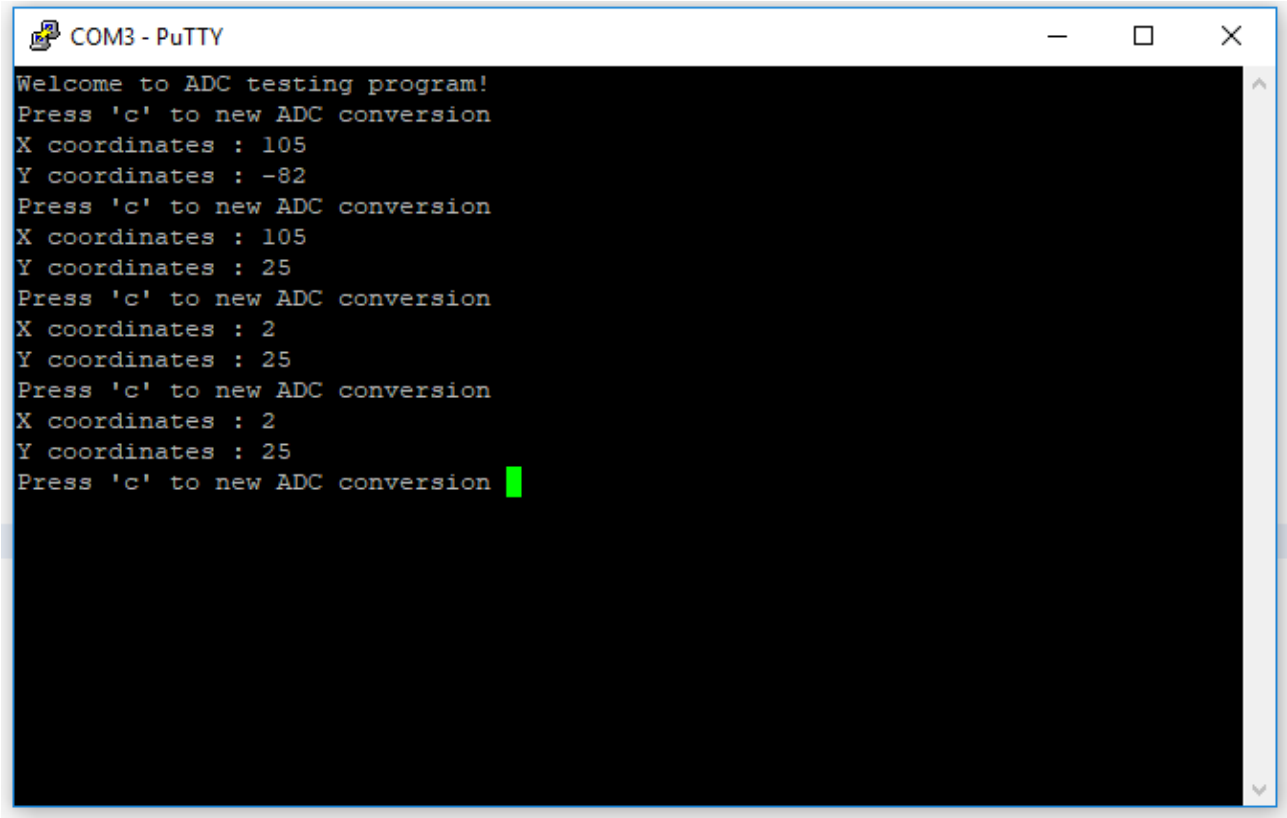



Figure 6.9. Potentiometer test

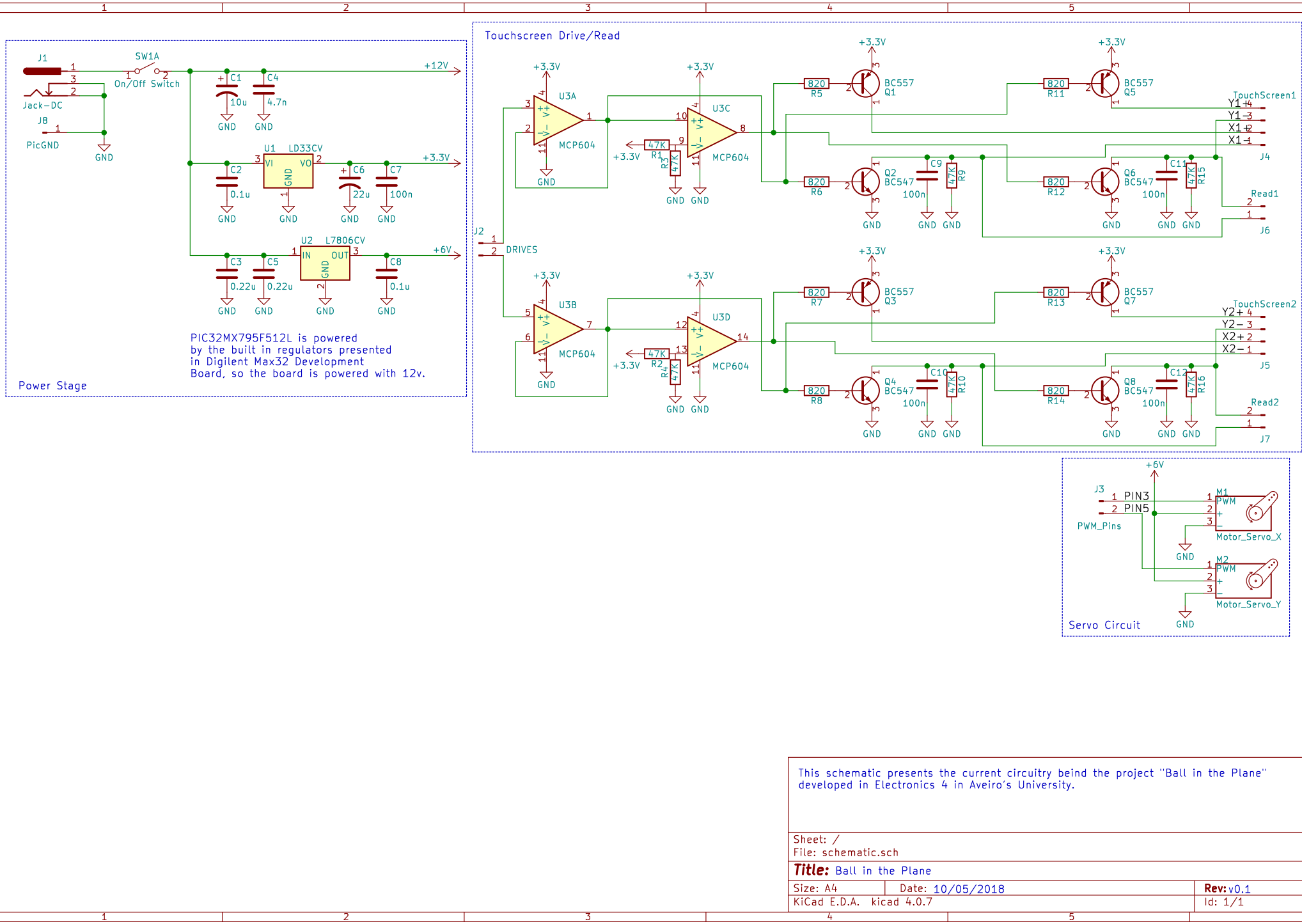
#### 6.3.4.2 Readings from TouchScreen

Since one of the project add-ons would be the possibility to read the desired setPoint through a secondary touch screen we obtained one similar to the one of the classroom, and with it we tested the circuit to read the touchScreen and the behaviour of the ADC module with it.

Single Mode Firstly we started with only a single axis reading, in order not to concern with delays needed for the circuit to stabilize. These measures permitted to make the linear regression and get an idea of the behaviour of the system.

Alternated Mode Finally it was used the circuit commuting between the XX axis and the YY axis to get the most close possible to the real touch screen.

# 7. CIRCUITS


Power Stage
PIC32MX795F512L is powered
by the built in regulators presented
in Digilent Max32 Development
Board, so the board is powered with 12v.
Touchscreen Drive/Read
DRIVES
TouchScreen1
TouchScreen2
Read1
Read2
Servo Circuit
PWM_Pins
Motor_Servo_X
Motor_Servo_Y
This schematic presents the current circuitry beind the project "Ball in the Plane"
developed in Electronics 4 in Aveiro's University.
Sheet: /
File: schematic.sch
Title: Ball in the Plane
Size: A4
Date: 10/05/2018
Rev: v0.1
KiCad E.D.A. kicad 4.0.7
Id: 1/1

## 7.1 Power Circuit

### 7.1.1 Concept

In order to power the circuitry and the max32 development board we decided to make a power stage circuit with the objective of easy powering through a universal DC jack $12v$. To do that was chosen two regulators: the LD33CV, a $3.3v$ regulator; and a L7806CV, a $6v$ regulator.

The $12v$ will power the development board through the built in DC jack, powering also all the regulators built in the board. To prevent noise induced by the board in the analog reading of the other components was placed some capacitors.

The $3.3v$ from the LD33CV will power all the touchscreen circuitry. The capacitors near the regulator serve as stabilizers recomended by the manufacter.

The $6v$ from the L7806CV will power the servo motors. The capacitors near the regulator serve aswell for stability proposes recomended by the manufacter.

The final power circuit can be seen in figure 7.1.

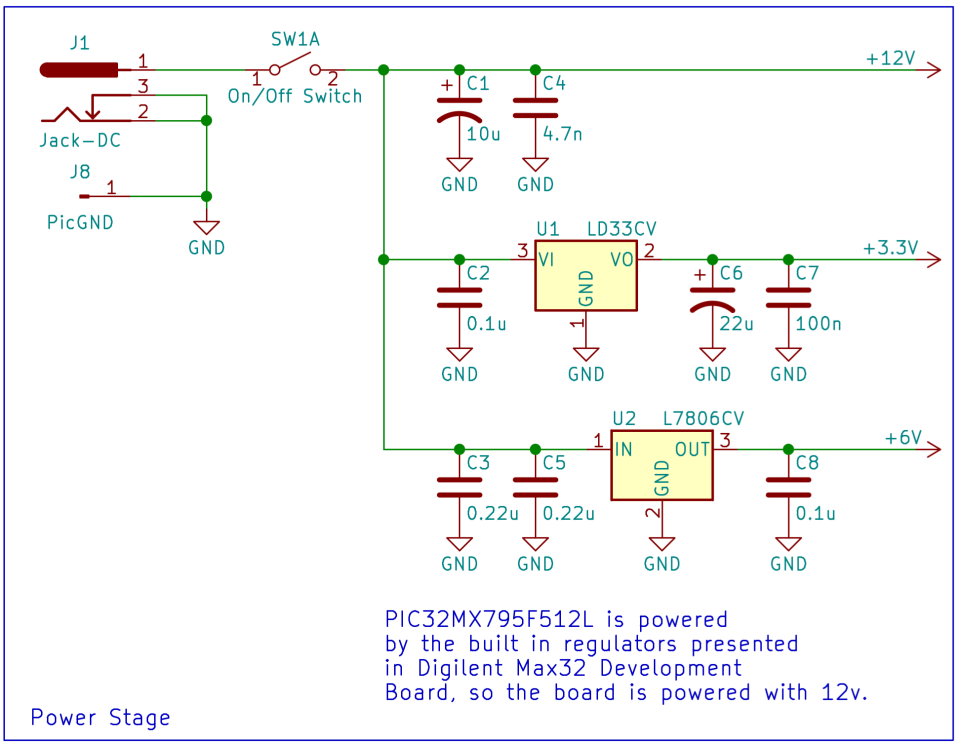



Figure 7.1. Power Circuit

### 7.1.2 Tests

All the regulators were tested in the laboratory independently and in conjunct with the same power supply, applying a load to validate a stable voltage output.

The L7806CV outputs a voltage a little higher than what specified in the datasheet, around $6.2v$, $0.2v$ higher than what was expected.

## 7.2 Touchscreen Circuit

### 7.2.1 Conception

In order to drive the touchscreen to switch power between axis, is necessary that the microcontroller pin sees an high output impedance in order to prevent over current through the pin.

With that in mind, the first concept of the circuit involved using four mosfets, two p-channel TP0604 and two n-channel TN0702, to switch the power to the axis in the same manner that was implemented in the final circuit with BJTs, that can be seen in figure 7.2.

These mosfets were chosen because of their low threshold voltage, fast switching speeds and high input impedance, making them ideally for analog switches and line drivers using a microcontroller, which are their intent.

Due to problems with component acquisition, the approach to the problem had to be changed. Instead of using high price mosfets that had to me ordered, we decided to emulate the pretended characteristics of those mosfets with more common and cheaper BJTs: pnp BC557 and npn BC547, with an common rail-to-rail two opamp package MCP602. Since space is not a concern in this project, the price and easier to find components make a great alternative choice to the initial approach.

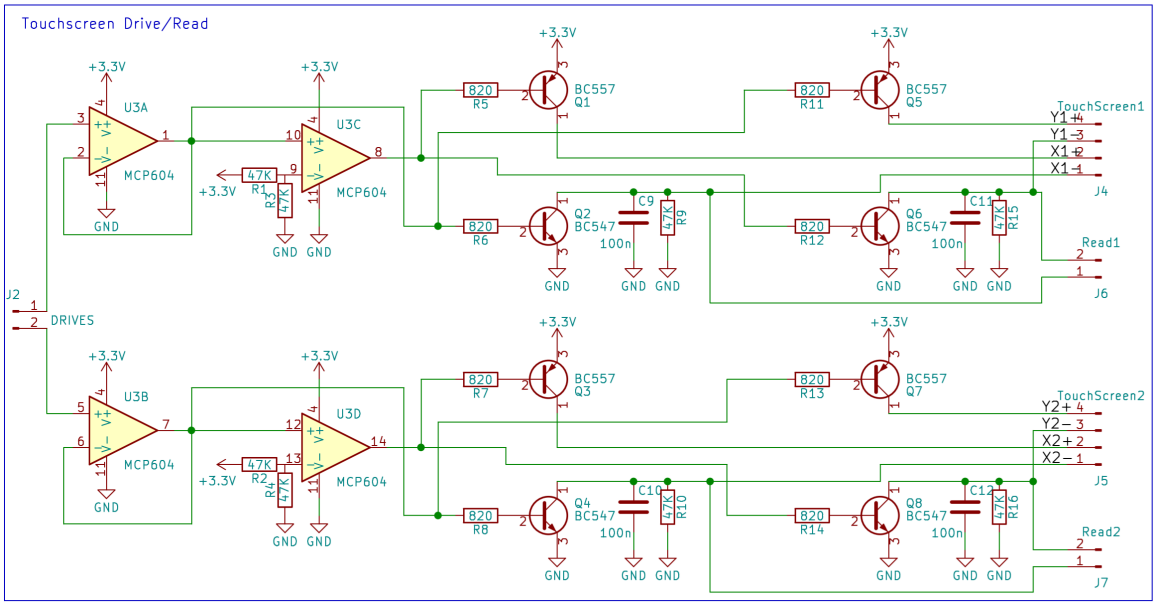



Figure 7.2. Touchscreen Drive/Read Circuit

Taking this approach forward, the final circuit can be seen in figure 7.2. The circuit had a few iterations being this the version 2.0.

The circuit started as showed in figure 7.3 with two a TL081 (since they were the only ones available) for testing purposes working as buffers for two drive signals generated by the microcontroller, resolving the issue of high impedance in the drive control pins. Putting to $3.3v$ one of the drive signal and the other to $0v$ was possible to switch the power between axis and reading the touch position. The low-pass filter in the reading pins was dimensioned to cut the main's induced noise at $50Hz$ and other high frequencies interference, so a common use $47K\Omega$ resistor and $100nF$ capacitor in parallel resulting in a $34Hz$ cutoff frequency is ideal.

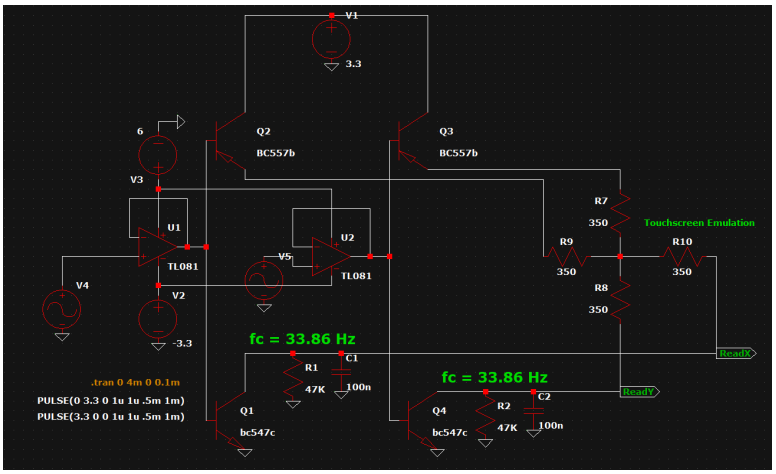



Figure 7.3. Spice: Touchscreen Drive/Read Circuit version 1.0

In this circuit we had some problems that could be improved on, namely the fact of having two drive signals when they were just the inverted signal of each another. With that we added two BJTs working as a inverter in version 1.1, that can be seen in figure 7.4.

There were some more improvements that could be done, so the final version came in iteration 2.0 that can be seen in figure 7.2. Instead of using a inverter made with BJTs we could use a two package opamp that would implement the buffer and a digital inverter with only a DIP8 footprint. There was another problem, with the TL082, we didn't had a rail-to-rail opamp, so we needed a negative power supply. That problem was fixed by using a common MCP602 opamp, powering it only with a $3.3v$ supply voltage in VDD and GND in VSS. Was added a resistor at the base of each BJT to prevent over current that could burn the transistors.

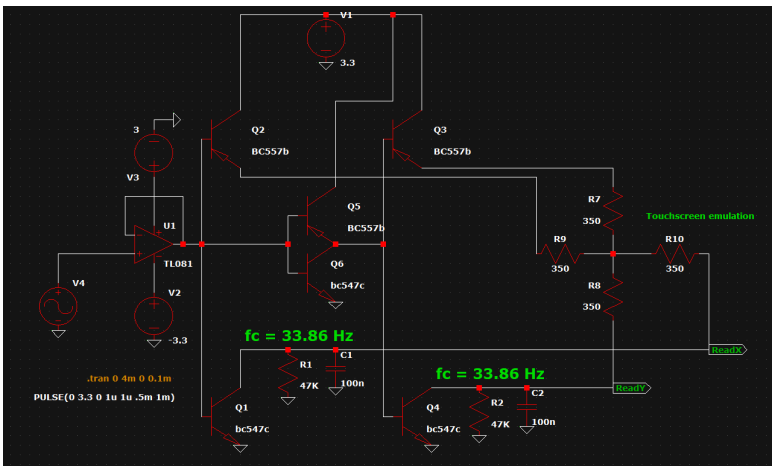



Figure 7.4. Spice: Touchscreen Drive/Read Circuit version 1.1

### 7.2.2 Simulation

The simulation of all the circuits was done in parallel with the conception, making sure that the circuits were function in every iteration of the concept process. With that will be presented the results archived in the simulation of the final version of the touchscreen circuit. The spice circuit can be seen in figure 7.5

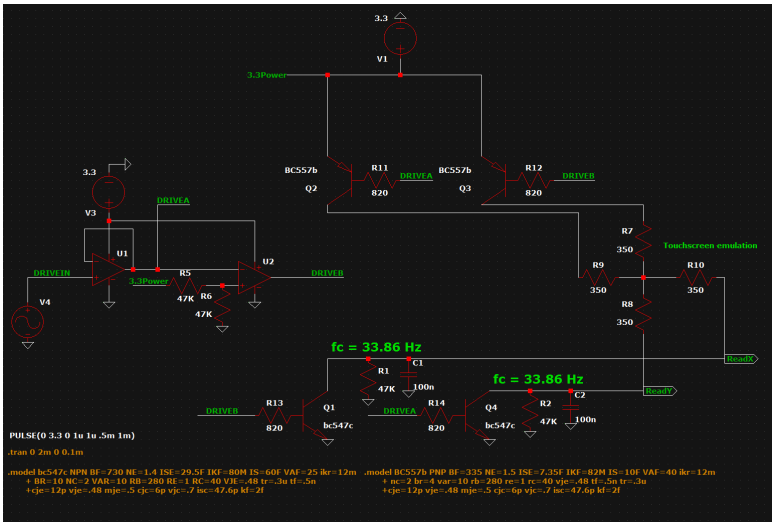



Figure 7.5. Spice final circuit for Touchscreen

In the figure 7.6 its possible to see the results of the drive signal and the axis reads. The readings are being done at $1KHz$, and with this results is possible to obtain a fastest sampling frequency if wanted, although it is not needed at all.

We can also see in figure 7.7 the drive signals in the base of the pnp and npn and the current in figure 7.8. A current of $2.3mA$ is sufficient to put the transistors in saturation.

These result are what we expected and fits the needs of the application, so the next step was pratical testing in the lab.

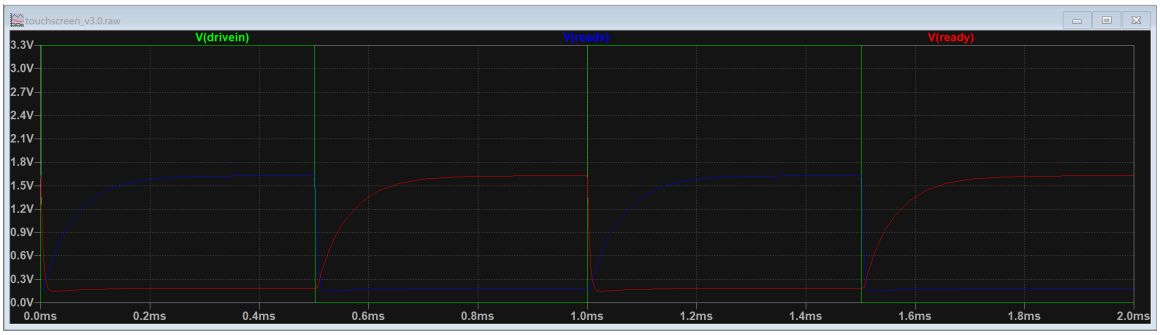



Figure 7.6. Simulation in Spice: Touchscreen Circuit - Drive signal, Read X axis and Read Y axis


Figure 7.7. Simulation in Spice: Touchscreen Circuit - BJTs Base Drive Signals


Figure 7.8. Simulation in Spice: Touchscreen Circuit - Current through the base of the pnp and npn BJTs

### 7.2.3 Testing

The circuit was built in a breadboard and tested using the same $1KHz$ square wave drive signal and with the same touchscreen impedance emulation as in spice.

The results can be seen in figures 7.9 to 7.12 and are what was expected. The scale of voltage possible goes from $V_{CE_{npn}}$ to $V_{cc} - V_{CE_{pnp}}$ with a base current of $2.66v/820\Omega = 3.2mA$, a bit higher than the simulation results.

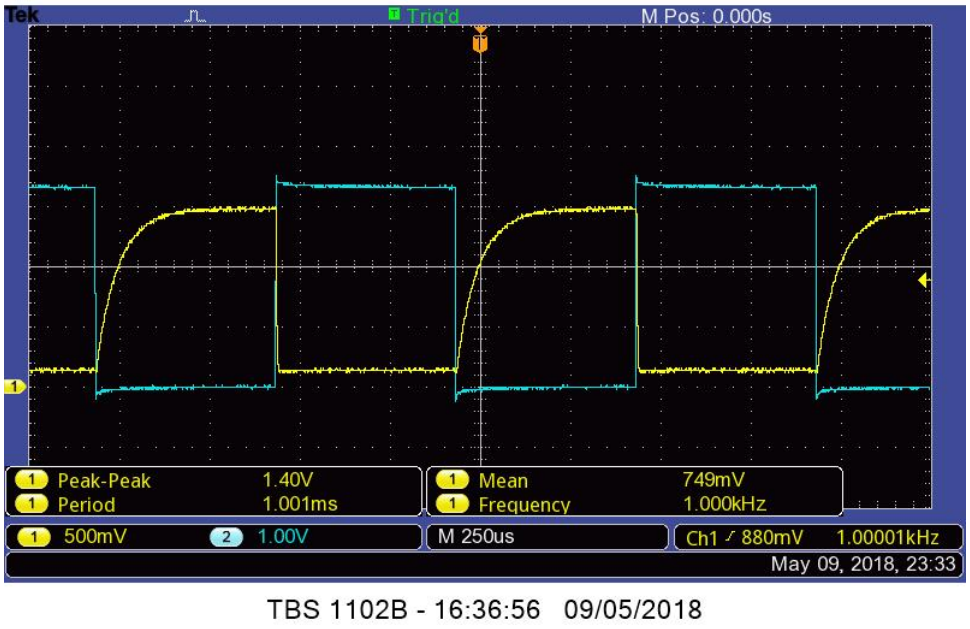



Figure 7.9. Drive and Read X Signals

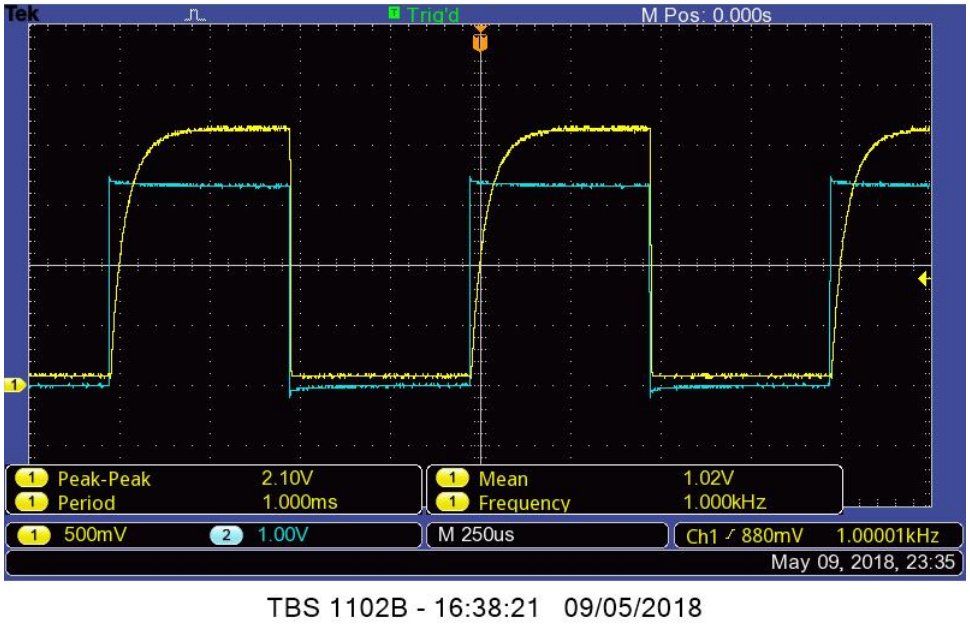



Figure 7.10. Drive and Read Y Signals

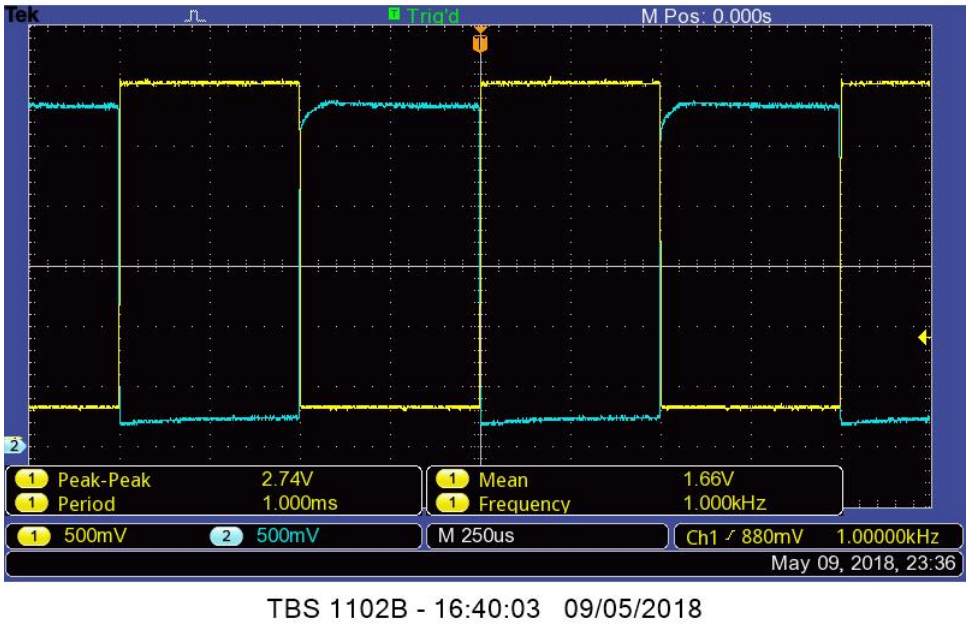



Figure 7.11. BJTs base Drive Signals

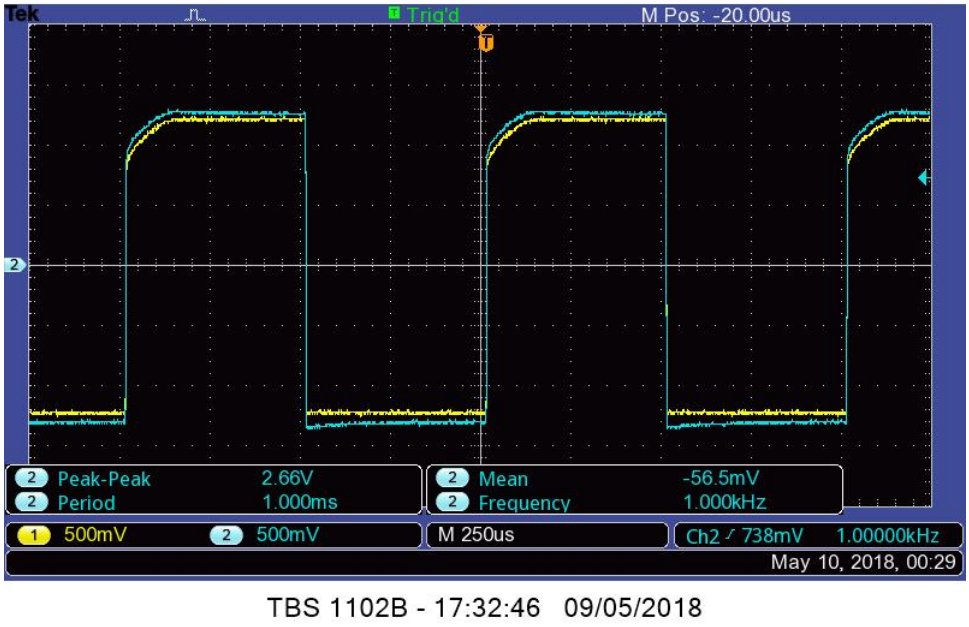



Figure 7.12. BJTs base current ($R = 820\Omega$)

## 7.3 Servos Circuit

### 7.3.1 Concept

The servo motors don't need any additional circuitry than $V_{DD}$ (power in), that should be between 4 and $6v$ and comes from the power stage circuit (from the L7806CV voltage regulator), a PWM Signal to control the servo position, that should have a peak voltage between $3.3v$ and $5v$ where it can't surpass $V_{DD} + 0.2v$, and ground.

The final circuit can be seen in figure 7.13.

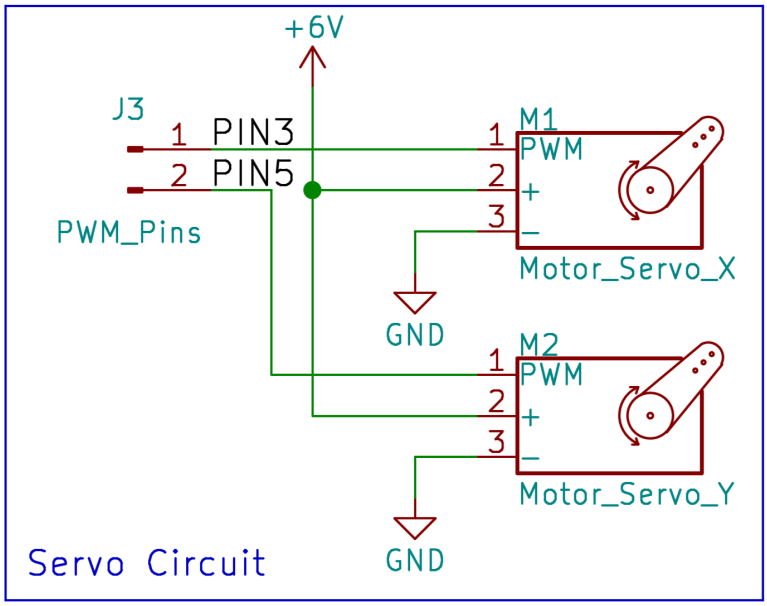



Figure 7.13. Servos Circuit

## 7.4 PCB

The current PCB design (version 1.0) can be seen in figures 7.14 and 7.15. Some space was left for future add-ons like the analog joystick. The second touchscreen hardware circuit for the setpoint input is already implemented in the board.

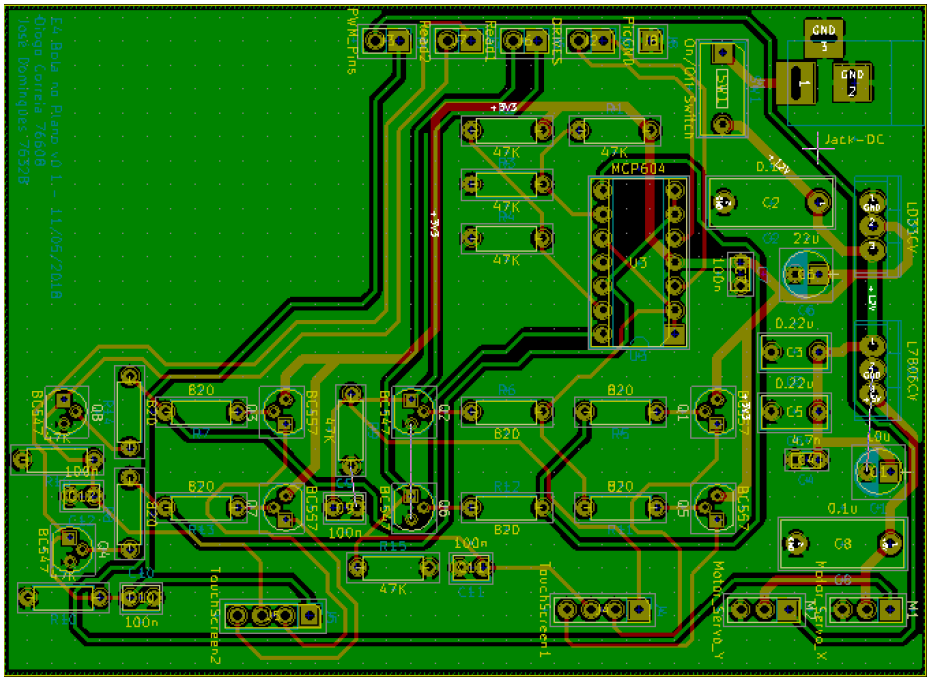



Figure 7.14. v1.0 PCB design

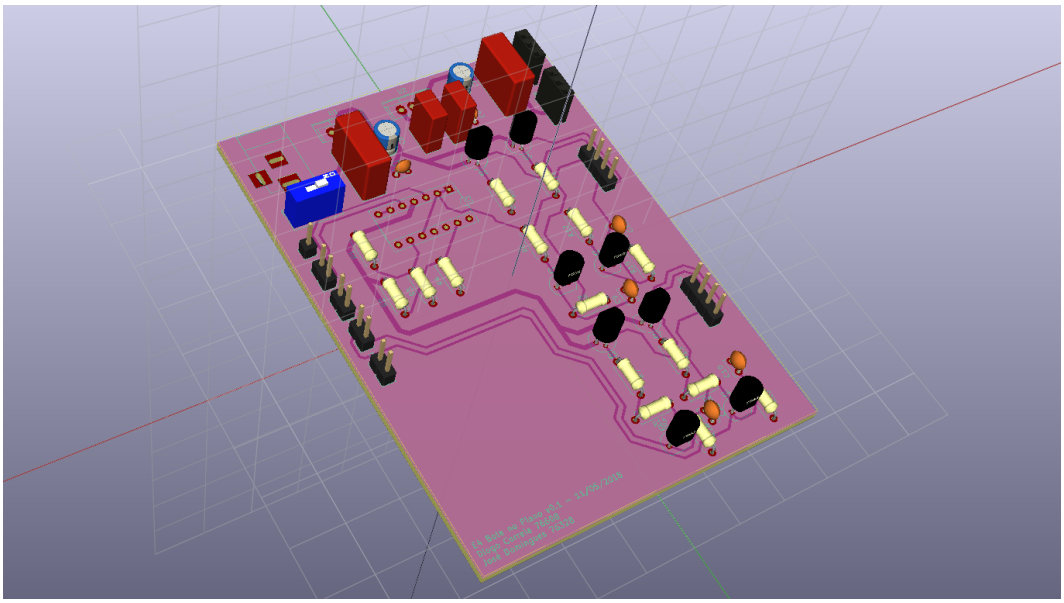


Figure 7.15. v1.0 3D PCB design

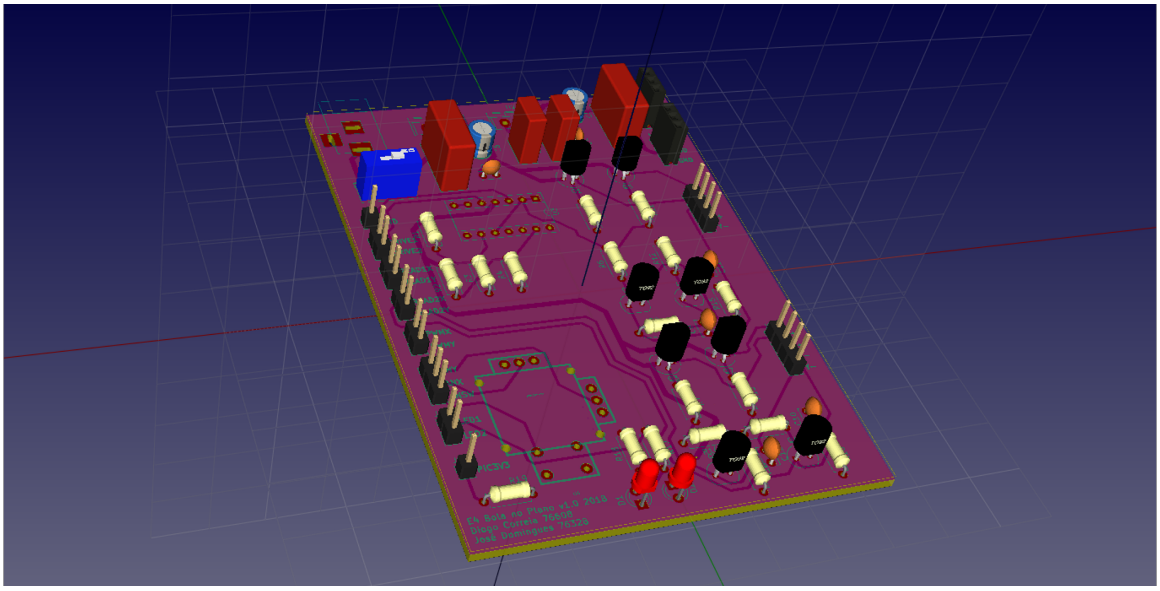


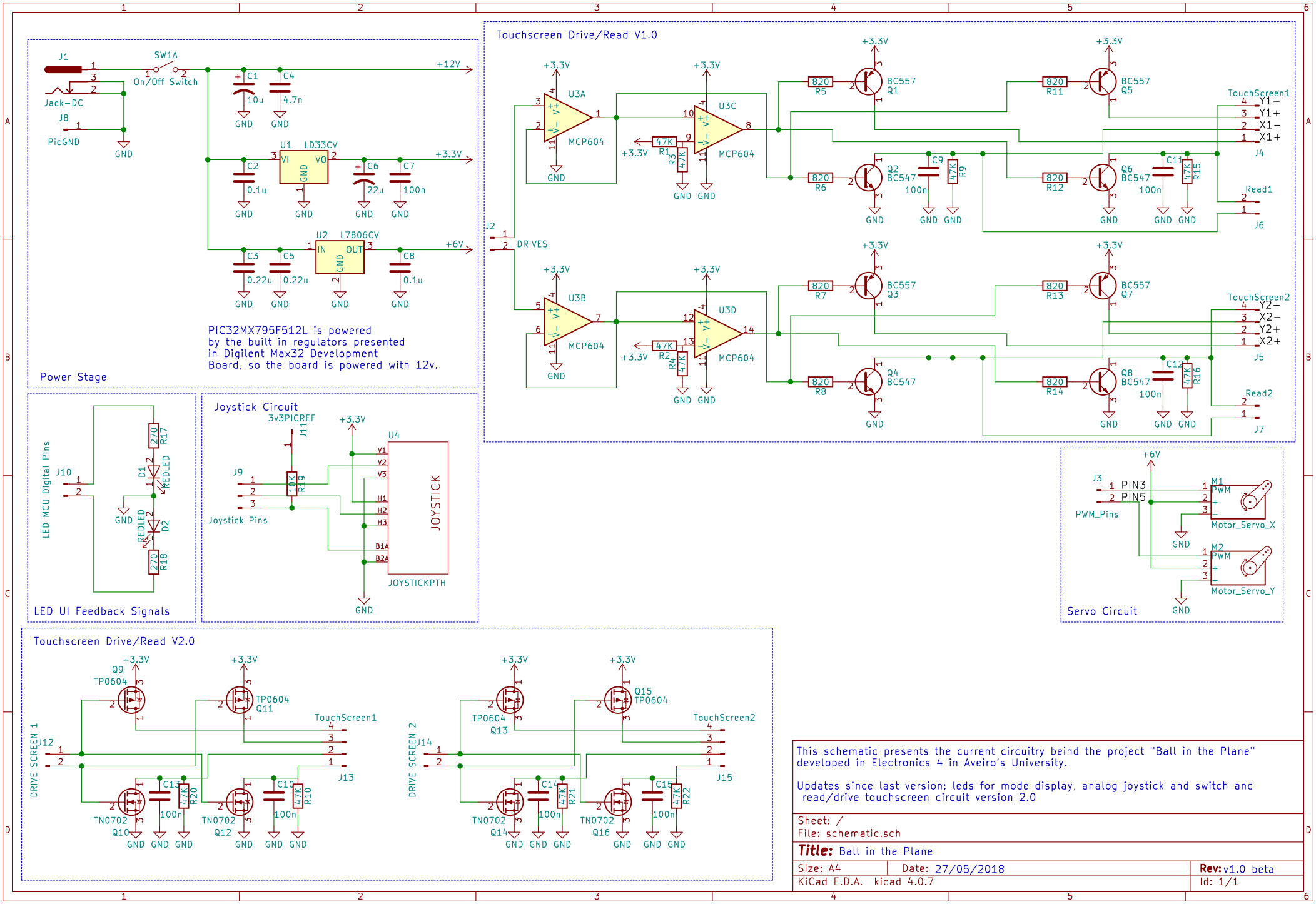
Figure 7.16. v1.1 3D PCB design already with the extra features circuitry

# Part III

# Phase 3

# 8. CIRCUIT UPDATE

## 8.1 Finished Circuit (v.1.0 beta)


Power Stage
PIC32MX795F512L is powered
by the built in regulators presented
in Digilent Max32 Development
Board, so the board is powered with 12v.
Touchscreen Drive/Read V1.0
LED UI Feedback Signals
Joystick Circuit
Servo Circuit
Touchscreen Drive/Read V2.0
This schematic presents the current circuitry beind the project "Ball in the Plane"
developed in Electronics 4 in Aveiro's University.
Updates since last version: leds for mode display, analog joystick and switch and
read/drive touchscreen circuit version 2.0
Sheet: /
File: schematic.sch
Title: Ball in the Plane
Size: A4
Date: 27/05/2018
Rev: v1.0 beta
KiCad E.D.A. kicad 4.0.7
Id: 1/1

## 8.2 Touch Screen Circuit Update

### 8.2.1 Reason

With the arrival of the mosfets needed for the first iteration of the circuit (presented in phase 1), a few inconveniences in the PCB design and a linear response when the ball isn't in the plane, we opted to return to the first iteration of the circuit, presented in the figure 8.1.

With this circuit we trade a couple of components (2 resistors and 2 op-amps) by 1 digital port in the microcontroller with the same effective cost. This is a good option since there are a good amount of digital ports still available in the MCU, making possible a construction of a simpler pcb with a smaller footprint.

Functionally, the high gate input impedance of the mosfets replace the op-amp buffer and the inverter is replaced with another digital output pin and software.

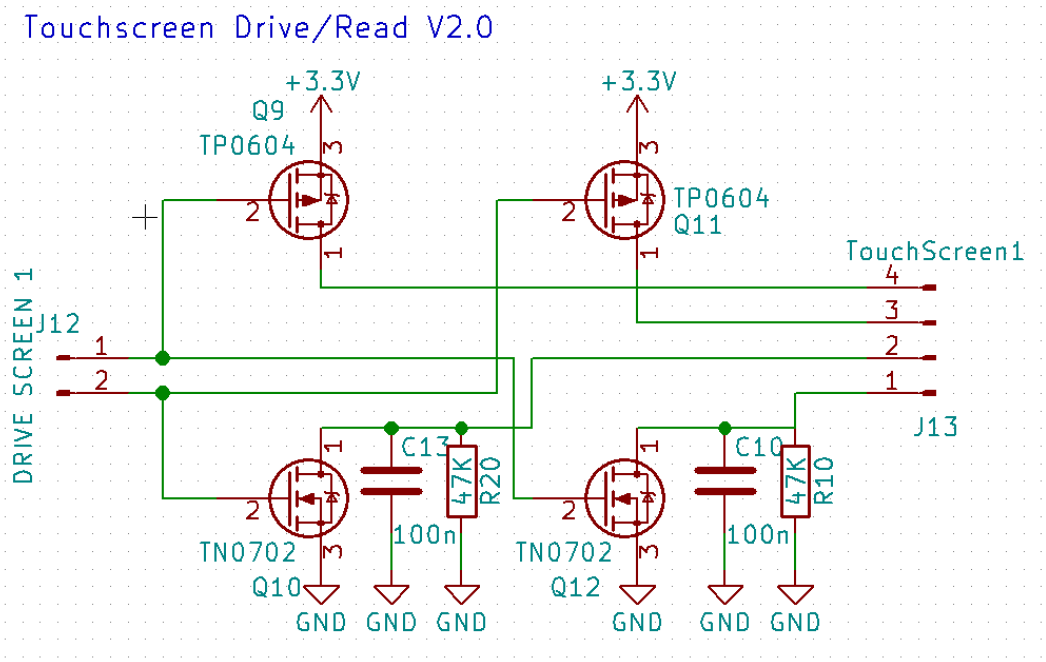



Figure 8.1. Touchscreen Mosfet drive circuit

# 9. USER INTERFACE

## 9.1 Objectives

In order to have a more natural and interactive way of interacting with the user, during the beginning of phase 2 were started two user interfaces: one developed in matlab; and another developed using web technologies.

Due to delays in development of the main project requirements both interfaces were abandoned. In the next sections is presented the work developed for each one.

## 9.2 Matlab User Interface

The matlab user interface can be seen in figure 9.1. The interface principal objective was interaction with the MCU showing the ball position in the graph and changing the setpoint.

Currently it can open a serial communication missing the protocol to handle the data duplex comunication.

## 9.3 Web based User Interface

The web based user interface was planned to show in 3D the plane changing angles and the ball movement in real time through RS232 serial communication. The UI can be seen in figure 9.2.

To render the 3D objects was used three.js library and for the setpoint GUI and serial communication p5.js librabry.

Currently it renders the plane and ball objects, renders the setpoint GUI and can extract the setpoint information. It misses the communication protocol to change information with the MCU, data processing and physics.

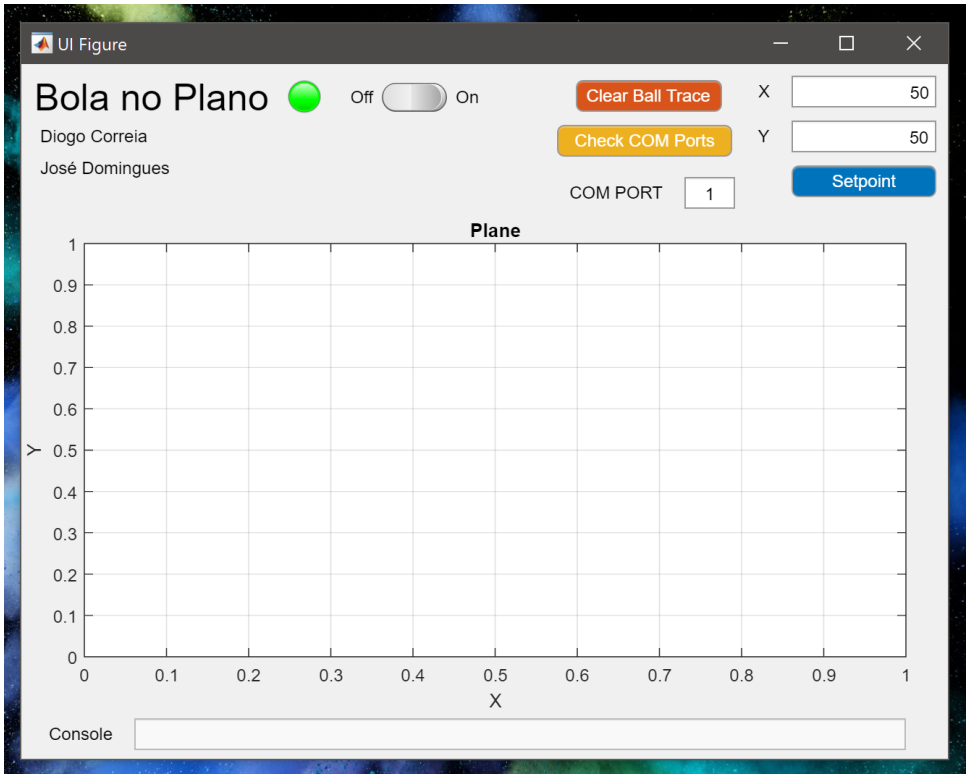



Figure 9.1. Matlab User interface

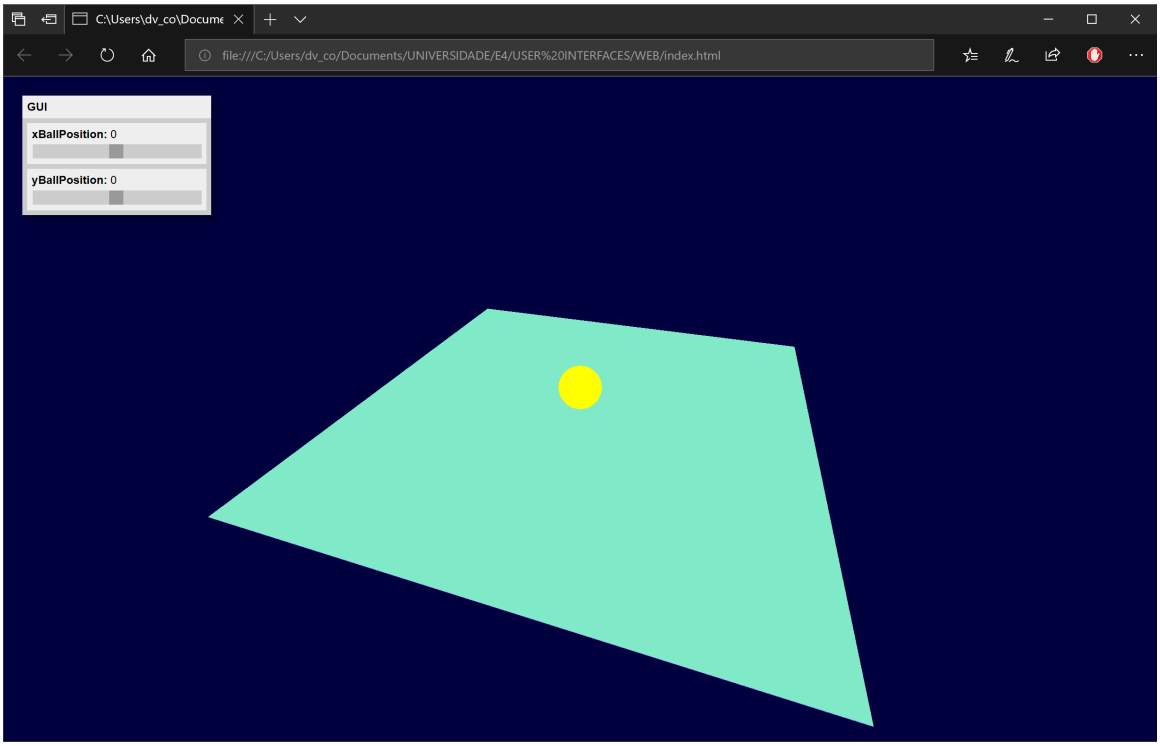



Figure 9.2. Web based User interface

# 10. ANALOG CONTROLLER

## 10.1 Operation Principle

This device is used for two dimensional input, in our specific case being used for the X and Y axis of the plane. It consists on two potentiometers that change its resistance according to the actual position of the protrusion relative to the default position, and therefore the value of the voltage drop across them alongside. Through the ADC a continuously measuring of the voltage is being done and it is possible to know which direction is the protrusion being pushed.

## 10.2 Configuration

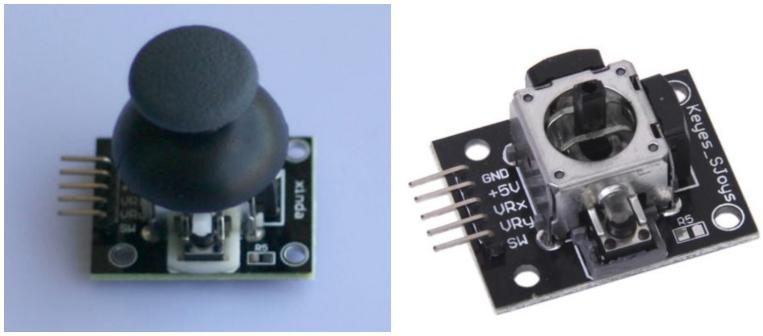



Figure 10.1. Analog controller

### 10.2.1 External interrupt

In order for the system to function correctly with these alternative modes it had to be configured with external interrupts, which allow the system to go to an interrupt service routine every time a certain input was on its rising edge. This was what allowed to, every time that the analog button was pressed, to change modes, which would be impossible by polling since nothing would guarantee that the pressed button would be detected.

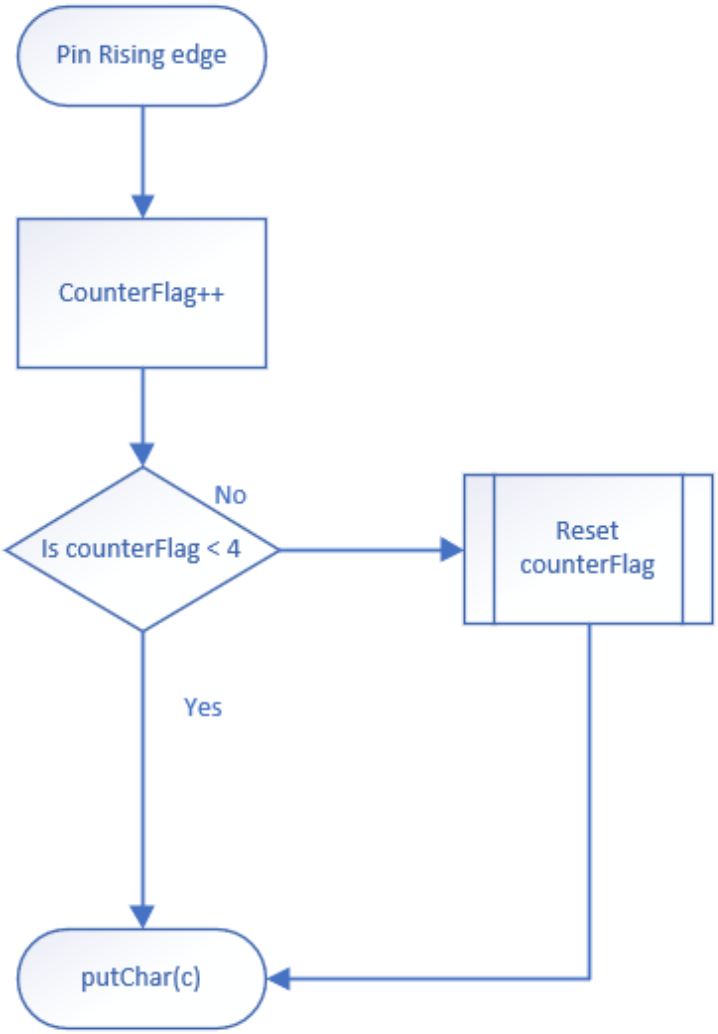



Figure 10.2. High Level Fluxogram of the External Interrupt Service Routine

## 10.3 Application Use

There are 4 different modes that are reachable through this analog controller. To get to each one of them one must press the controller button in order to cause the external interrupt that will change the mode.

### 10.3.1 1º Mode: Default Mode

In the default mode the plate will start to send the ball's coordinates in both X and Y axis to the microcontroller, and the microcontroller will respond accordingly to these with a control signal. The setpoint will be defined by the user through an input on the terminal.

### 10.3.2 2º Mode: Analog Setpoint Mode

In this particular mode the setpoint will be defined by the analog controller, i.e, the user will move the controller to reach the desired setpoint and the ball will follow the coordinates.

### 10.3.3 3º Mode: Analog Inclination Mode

This particular mode won't have a PID control, its purpose is to give the user a different experience, i.e, the user will be able to change the inclination of the plane with the analog controller in order to try to balance the ball in the center.

### 10.3.4 4º Mode: Secondary Touch Panel Setpoint Mode

Within this one, the user can choose the SetPoint through the secondary touch panel. This makes the user capable of exactly pinpointing the location of the ball since the two panels are of equal dimensions. In order to return to the default mode again one must press the analog button one more time.

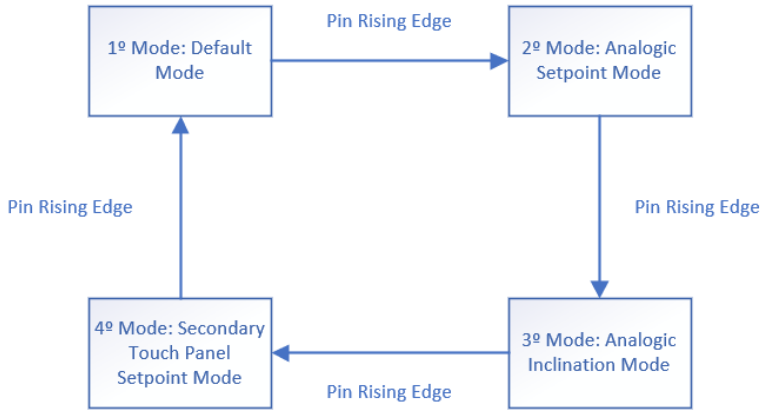



Figure 10.3. High Level Fluxogram of the 4 different system modes

# 11. SECONDARY TOUCHSCREEN

Like the main touch panel, this too has 210mmx165mm of dimensions. It is used to allow a pair of coordinates to be read from it like the main one, but the purpose of this is to get these coordinates as a new setpoint.

The implementation of the circuit was the same as for the first one and to read the coordinates two different channels of the ADC are used and the linearization is done inside the microcontroller.

Figure 11.1. Secondary touch panel

# 12. PID CONTROLLER - PROPORTIONAL INTEGRAL DERIVATIVE CONTROLLER

## 12.1 Operation Principle

In order to control the ball position in the plane, the best option controlling a system with this characteristics is a widely common PID controller.

A PID, proportional–integral–derivative, controller is a feedback control loop that periodically calculates the error value between a desired setpoint and the output system value, applying a correction based on proportional, integral, and derivative coefficients.

## 12.2 Dicrete PID Implementation

The discrete PID implementation is a PID controller *optimized* for discrete systems like microprocessors and computers.

In order to calculate the control signal, it need 3 parameters ($a_0$, $a_1$ and $a_2$), that are calculated as follows in equations 12.1 to 12.3, being $h$ the sampling period and $K_p$, $T_i$ ans $T_d$ the PID proportional, integral, and derivative coefficients.

$$a_0 = K_p * (1 + \frac{h}{2 * T_i} + \frac{T_d}{h}) \tag{12.1}$$

$$a_1 = K_p * (-1 + \frac{h}{2 * T_i} - \frac{2 * T_d}{h}) \tag{12.2}$$

$$a_3 = \frac{K_p * T_d}{h} \tag{12.3}$$

The control signal is showed in equation 12.4, were $k - x$ means the past $x$ value, being $u(k)$ the control signal and $e(k)$ the error signal.

$$u(k) = u(k-1) + a_0 * e(k) + a_1 * e(k-1) + a_2 * e(k-2) \tag{12.4}$$

In order to facilitate the calculations and maintain up to 2 decimal cases, we changed the equations to the following 12.5 to 12.8 equations. The sampling period was changed to sampling frequency ($f$) to avoid float operations. To maintain the two decimal cases, the PID coefficients

($K_p$, $T_i$ and $T_d$) were passed multiplied by $DP$, that in this case is 100. The "$a$" parameters are then passed multiplied by $DP$ to $u(k)$, being $a_0$, $a_1$ and $a_2$ always a int number. With this configuration the decimal cases are maintained during each operation.

$$a_0 = K_p * (DP + DP * DP/(2 * T_i * f) + T_d * f)/DP \tag{12.5}$$

$$a_1 = K_p * (DP * DP/(2 * T_i * f) - 2 * T_d * f - DP)/DP \tag{12.6}$$

$$a_2 = K_p * T_d * f/DP \tag{12.7}$$

$$u(k) = u(k-1) + (a_0 * e(k) + a_1 * e(k-1) + a_2 * e(k-2)/DP \tag{12.8}$$

This control signal is them applied to each servo, existing a PID control signal for each one of the axis, X and Y. The code implementation will be described in the following section, were will be explained the main loop.

# 13. MAIN PROGRAM IMPLEMENTATION

## 13.1 Main Loop

Following a necessity of improving the code performance of the main code developed in an early stage of the project, a second version was created and presented in this section. The second version was implemented with discrete PID and new algorithm options. The code can be checked in the project *mainDiscretePID.X* in the *code.ua* repository.

The main loop of the implementation follows the flowchart presented in figure 13.1. When the main loop is called, *initWorld* is called as well and then it enters an infinite loop were 4 interruptions can happen:

- *ext_SW* is the function called upon external interruption from the Analog switch (Priority 4, the highest in the program);
- *isr_PID* is the PID control loop (Priority 2);
- *isr_Analog* is the analogic joystick reading interruption loop (Priority 3);
- and *isr_SecondTouchScreen* is the interruption loop that reads the secondary touchscreen for setpoint user input (Priority 3).

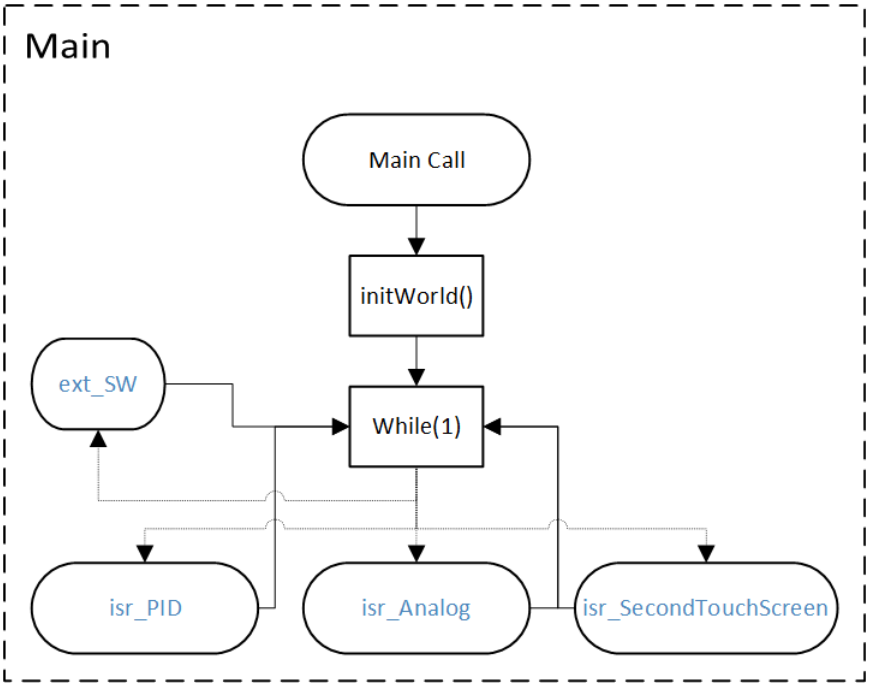



Figure 13.1. Main Function Discrete PID Implementation

## 13.2 initWorld

In *initWorld*, the function initializes all parameters, configurations, variables and registers needed to run the program. The fluxogram can be seen in figure 13.2.

Is starts by initializing UART, ADC and PWM libraries used to interact with the user, read the analog values and controlling servos correspondingly. Then follows with the calculations of the $a_0$, $a_1$ and $a_2$ PID parameters for each axis, configuration of the touchscreen circuit drive pins, configuration of all interruptions and finally enabling globally the interruptions and disabling each individual interruption with the exception of the analog switch ($ext\_SW$). The interruptions will be turn on and off depending on which mode is active. The comportment is controlled by the analog switch external interruption $ext\_SW$.

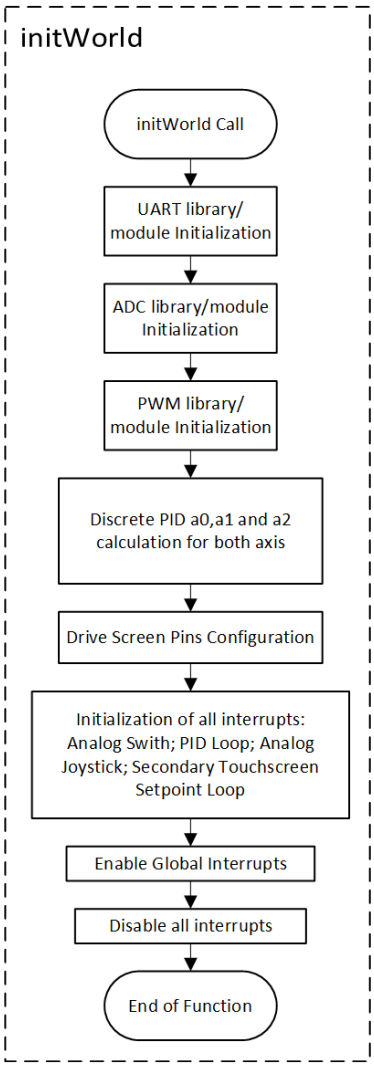



Figure 13.2. *initWorld* Function fluxogram

### 13.3 ext_SW

The system modes described previously in the document are all controlled thought an external interrupt function call activated when the external interrupt pin goes high. The function handling the interruption call is *ext_SW*. The function's fluxogram can be seen in figure 13.3.

On function call, it runs a debounce software delay followed by a switch condition that turns on and off the correspondent interruptions for each one of the modes. Each of the modes was described and explained previously in this document in section 10.3.

The mode is incremented at each call till the last mode (mode 4), returning to the first operation mode (mode 1).

The Setpoint user input for mode 1 is implemented inside the interruption, being asked for setpoint input through UART upon mode activation.

At each mode activation, it displays in the terminal through UART a feedback string to let the user know which mode is active.

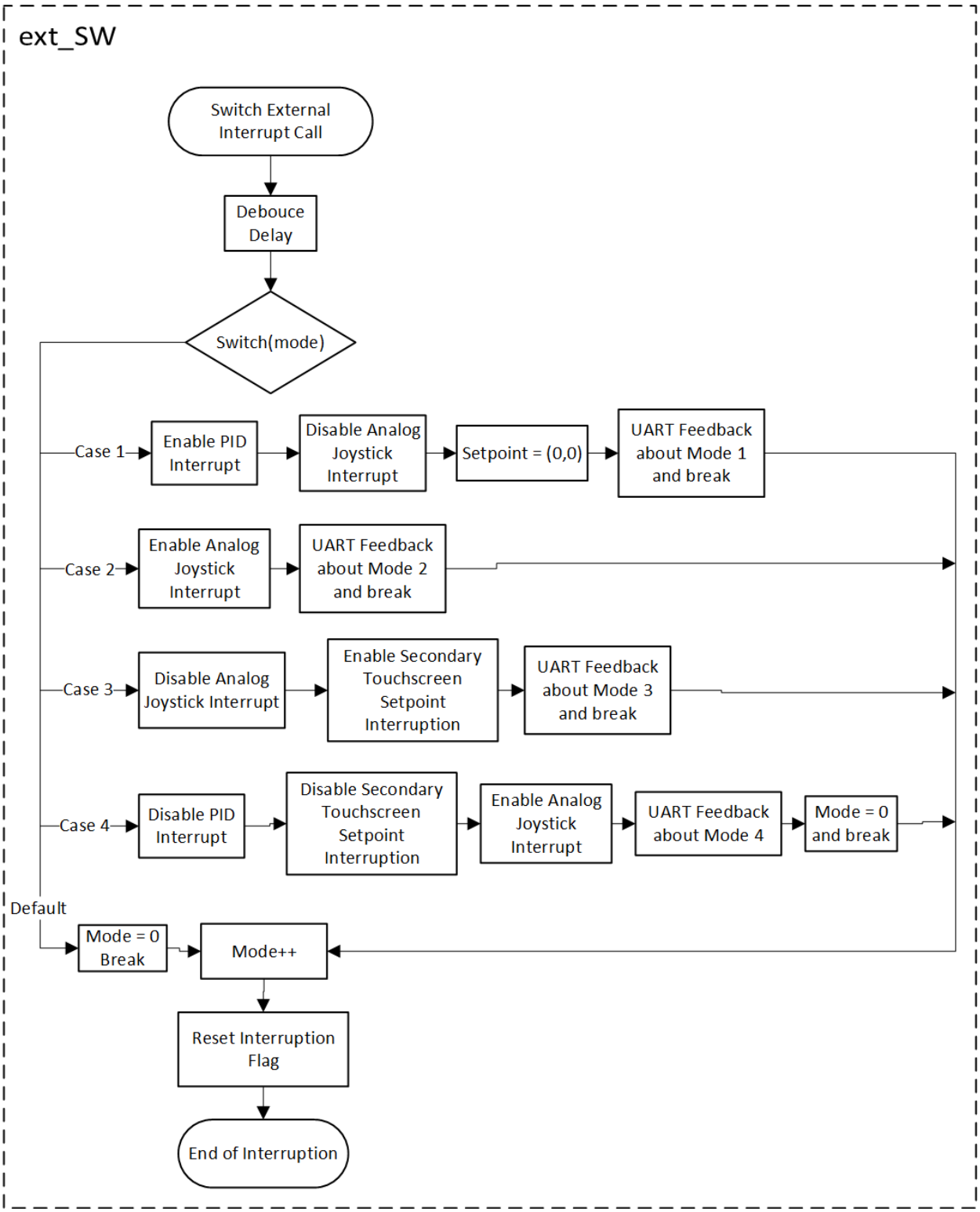



Figure 13.3. *ext_SW* Interruption Function call fluxogram

### 13.4 isr_PID: PID Control Loop

The PID Control Interruption loop is one of the most important modules of the code, running in all modes of the system with the exception of mode 4. The full fluxogram of the function can be check in figure 13.5.

The interruption has a frequency of $40Hz$, controlled by timer 4, calling the interruption handler function in a $0,025$ seconds interval.

The function is composed of a switch condition loop that runs through 4 states. The two first states handle Y axis control loop and the two other states handle the X axis control loop.

In **State 1** the drive pins are put in "reading Y axis" configuration and is sent to UART all Y axis information. The information is sent in this state in order to remove processing cycles from **State 2**, which handles Y axis PID control and touch detection of the ball, without any compromises, distributing the work efficiently through the states.

Both states are not combined because the circuit needs time to stabilize as can be seen in figure 13.4, where the sampling frequency is increased to $1KHz$ to demonstrate the causes in this decision, assuring that the circuit is stabilized when reading the ball position.

The **State 3** and **State 4** are the equivalent algorithm of State 1 and 2 but for X axis.

In figure 13.4 can be seen the different states of the loop, being state1 and state3 the drive pins changes to read correspondingly Y and X, and state 2 and 4 the readings and control of each axis in the same order.

Effectively, the control loop actuates in each axis once in each 4 iterations of the interruption, being the effective PID controller sampling time equal to $0.1s$ with a sampling frequency of $10Hz$.

This method allows a variation in sampling frequency without compromising the system stability when reading the ball position.

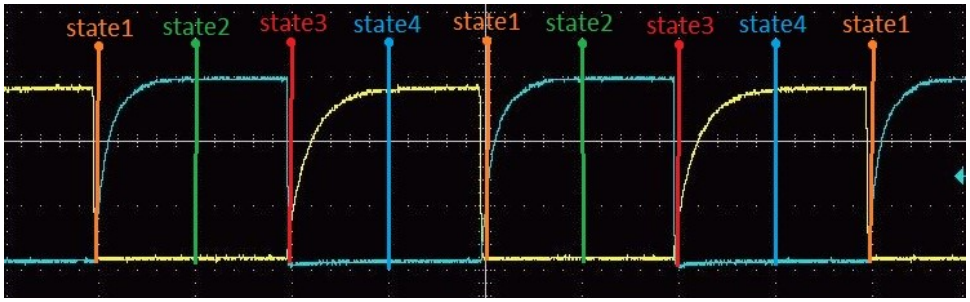



Figure 13.4. PID Control Loop States Example (Blue - Y axis readings ; Yellow - X axis readings

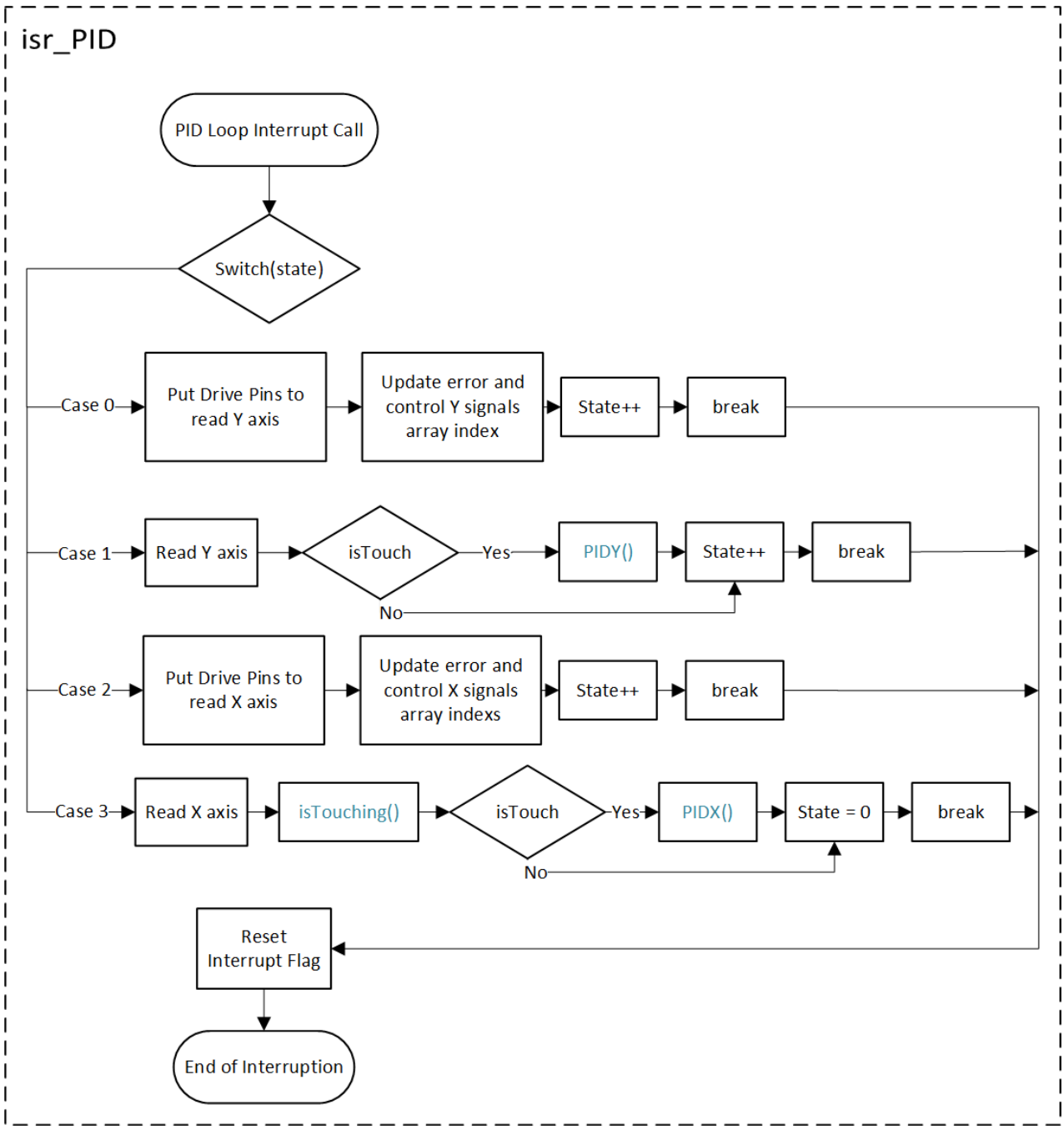



Figure 13.5. *isr_PID* Interruption Function call - Control Loop fluxogram

## 13.5 PIDY and PIDX

The functions *PIDY* and *PIDX* handle the PID control algorithm calculations for each axis, applying the algorithm presented in the fluxogram of the figure 13.6 and calculations presented in section 12.

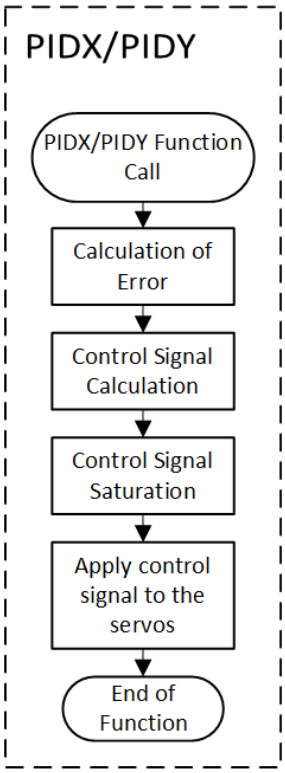



Figure 13.6. *PIDX* and *PIDY* Function fluxogram

### 13.6 isTouching

The function *isTouching* checks if the main plane is being touched by the ball. In case that isn't, it levels the plane and puts the global variable **isTouch** to zero that will be used in *isr_PID* to jump the PID loop.

The fluxogram of the function can be checked in figure 13.7.

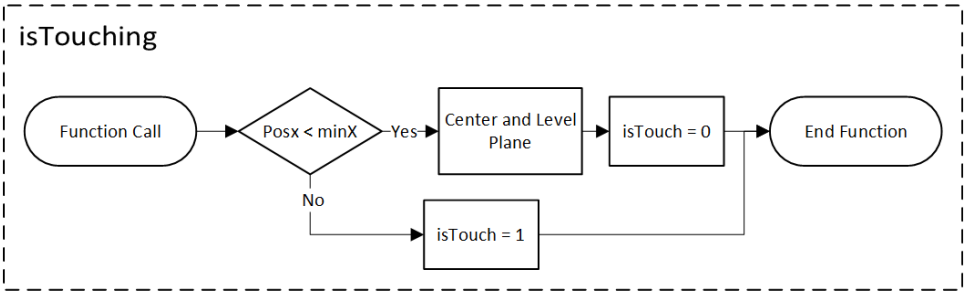



Figure 13.7. *isTouching* Function fluxogram

### 13.7 isr_Analog

The interruption *isr_Analog* handles the algorithms involving analog joystick position readings. This interruption is active in mode 2 and 4 (check section 10.3 for mode of operation specifications).

This interruption is generated by timer 3, running at $5Hz$.

As can be seen in the function's fluxogram in figure 13.8, the interruption function handler runs one of the two blocks depending of the mode of operation.

In mode 2 the function changes the ball setpoint and in mode 4 changes the PWM signal controlling the servos, actuating directly at the servo's angle.

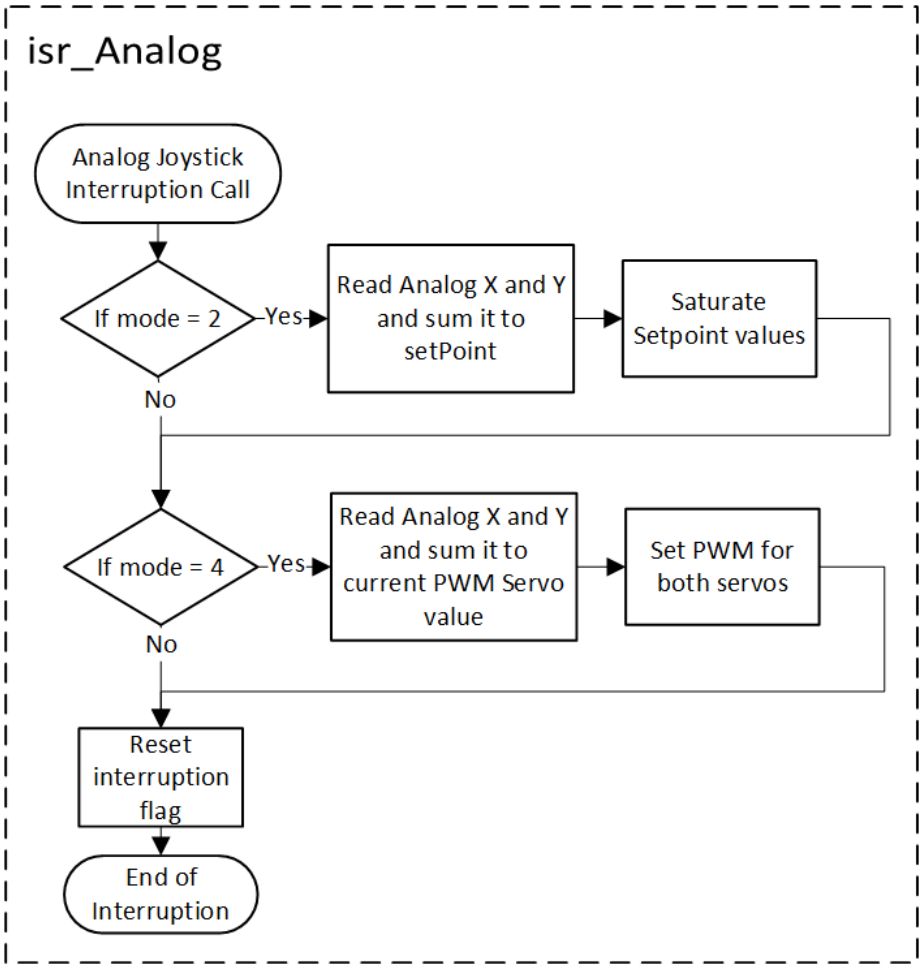



Figure 13.8. *isr_Analog* Interruption Function call

### 13.8 isr_SecondTouchScreen

Handling the secondary touchscreen features is a interruption handler of timer 5 running at $2Hz$. This function, when active (in mode 3), starts by reading the X touchscreen position following with a function that checks if the user is touching the screen. If the user is touching the screen, reads the Y axis position and sets the setpoint to the position in the secondary screen. In case the user isn't touching the screen, sets the setpoint to (0,0).

The fluxogram of this function can be seen in figure 13.9.

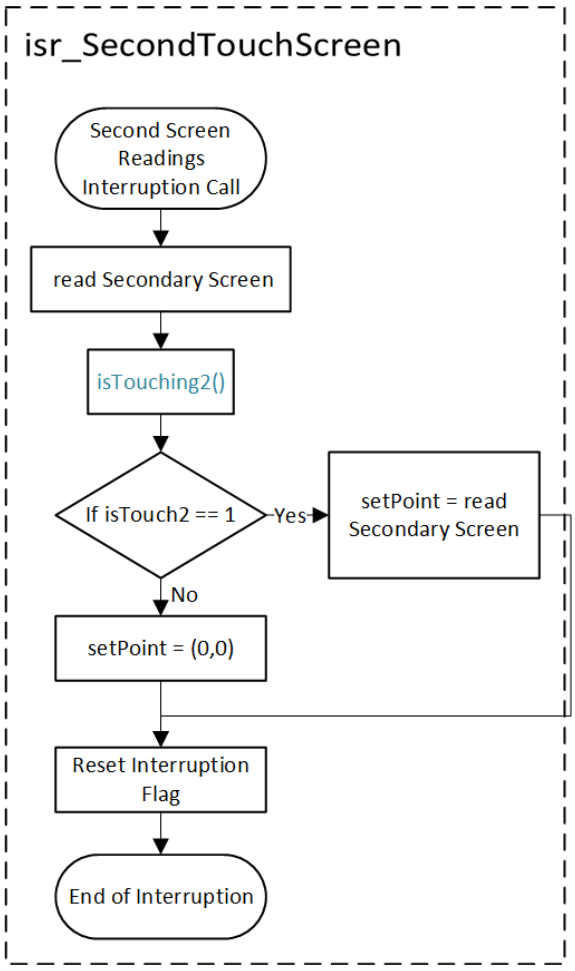



Figure 13.9. *isr_SecondTouchScreen* Interruption Function call

# 14. FINAL RESULTS

## 14.1 Phase 2 Results

All objectives in phase 2 were completed, being present functions that test the features for each library and the correspondent documentation presented in phase 2 of this document.

## 14.2 Phase 3 Results

### 14.2.1 Requirements Status

The phase 3 was in part completed. The integration of all modules was completed with success, been created an algorithm that involves all agents needed to complete the main specification.

The PID tuning was not completed with success, since the X axis servo was not responding properly to the control signal.

The latest version of the code with the discrete PID implementation was not fully tested with hardware.

### 14.2.2 Extras

The extra features were partially completed too. The analog joystick with the features presented in section 10.2 were fully completed. The secondary touchscreen for user setpoint input was completed as well. The graphical user interfaces were not completed, missing the points explicit in section 9.

## 14.3 Next Steps

To properly finish the project there are some points that need to be worked on, being them:

- improve the PID controller tuning;
- fully test the last version of the main project code;
- update the PCB circuit with the latest touchscreen control circuit.

# REFERENCES

# REFERENCES

[1] https://elearning.ua.pt/pluginfile.php/241845/mod_resource/content/1/sbaa036.pdf

[2] https://eu.mouser.com/datasheet/2/295/nkkswitches_4-wireTouch-1187718.pdf

[3] https://www.parallax.com/sites/default/files/downloads/900-00005-Standard-Servo-Product-Documentation-v2.2.pdf

[4] http://www.st.com/content/ccc/resource/technical/document/application_note/9d/56/66/74/4e/97/48/93/CD00004444.pdf/files/CD00004444.pdf/jcr:content/translations/en.CD00004444.pdf

[5] http://ww1.microchip.com/downloads/en/DeviceDoc/61104E.pdf

universidade de aveiro
theoria poiesis praxis

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

# User Manual

## Eletrónica IV

Diogo Correia 76608

José Domingues 76328

## 1 Introduction

The "bola no plano" system is based on a self controlling system that moves a ball on a plane to a given setpoint chosen by the user. All the text needs to be sent and received through a terminal emulator such as **Putty** or equivalent. Its propellers are two servos that change the inclination of the plane according to the desired destination, and a touch panel that reads continuosly the correct location of the ball.

This operating instruction covers information on safety and caution. Please read relevant information carefully.

## 2 Safety Note

To avoid possible damage to the system adhere to the following rules:

- Use an appropriate DC power supply voltage to power the PCB
- Be careful when placing the ball or other objects in the plane, since if enough force is applied, it can damage it
- Do not force the servos when turned on

## 3 Specifications

The system has the following operation modes available:

1 PID controller enabled - SetPoint defined by the UART module

2 PID controller enabled - SetPoint defined by the analogic controller

3 PID controller disable - Plane angles defined by the analogic controller

4 PID controller enabled - SetPoint defined by the secondary touchscreen

It also detects if the ball is touching or not the plane and resets its position if not.

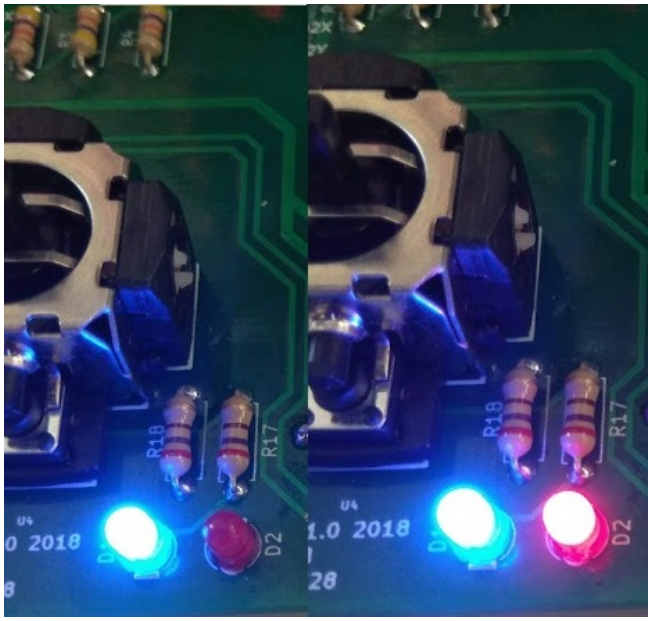
## 4 Explanation of menu and indicators

Mostrar foto da pcb, leds e assim falar do debugging

Falar do menu que tenho que acrescentar ao programa

Falar do que pode ser printed no terminal

On the PCB the user may see two different LEDs, one blue and other red. These are made for visual information. Since they are two there are four different lighting schemes. While the system has a UART module that prints to the terminal every information needed to the user these can provide a simpler anda faster way to see basic information about the modes of operation.




Blue and Red Leds

The following table portraits the corresponding modes to the ligthing schemes

| Blue Led | Red Led | Mode |
|---|---|---|
| 0 | 0 | 1º |
| 0 | 1 | 2º |
| 1 | 0 | 3º |
| 1 | 1 | 4º |

Indication of modes through Leds

## 5 System

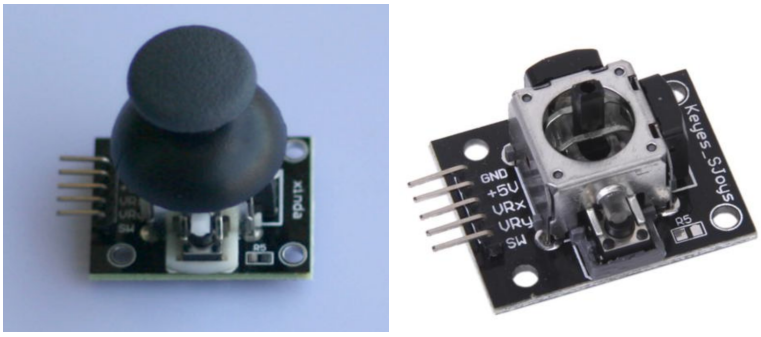
Upon starting the system, a string will appear on the terminal and the system will remain on standby until the key 's' is pressed to start the control.

### 5.1 1º Mode: Default Mode

Right after the key is pressed, the plate will start to send the ball's coordinates in both X and Y axis to the microcontroller, and the microcontroller will respond accordingly to these with a control signal. The setpoint will be defined by the user through an input on the terminal. While on the control loop, a string will be displayed on the terminal stating that for a new setpoint the user must press 'n'. After pressing 'n' the user must insert the coordinates according to this example: 013 056

### 5.2 2º Mode: Analogic Setpoint Mode

To reach this and the other alternative modes, the user must press the analogic button. In this particular mode the setpoint will be defined by the analogic controller, i.e, the user will move the controller to reach the desired setpoint and the ball will follow the coordinates.




Analog controller

### 5.3 3º Mode: Analogic Inclination Mode

This particular mode won't have a PID controll, its purpose is to give the user a different experience, i.e, the user will be able to change the inclination of the plane with the analogic controller in order to try to balance the ball in the center.

### 5.4 4º Mode: Secondary Touch Panel Setpoint Mode

This is the last mode. Within this one, the user can choose the SetPoint through the secondary touch panel. This makes the user capable of exactly pinpointing the location of the ball since the two panels are of equal dimensions. In order to return to the default mode again one must press the analogic button one more time.

Secondary touch panel

### 5.5 No touch detected

When the plane is not being touched, the microcontroller receives this information and proceedes to set the default angle to the servos, which will reset the plane inclination to $0^{o}$ on both axis.