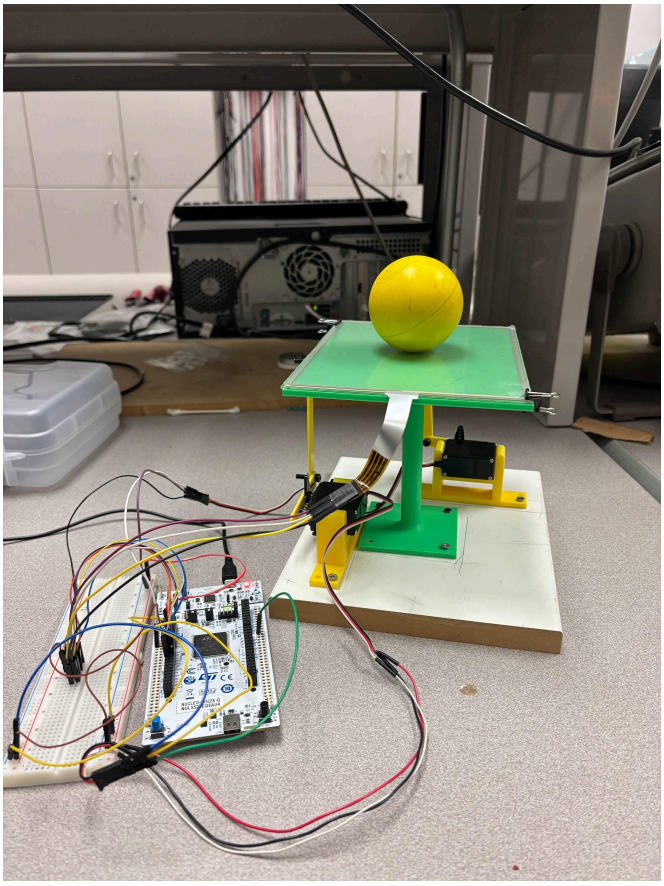
PID Balancing Platform

Michael Sportelli, Keegan Nelson
ECE 433 – Embedded and Portable Computing Systems
3 May 2025



I. INTRODUCTION

This report outlines the ECE433 project we designed and created. The project is a PID controlled platform, which consists of a platform controlled by servo motors on two axes in order to balance a ball. The system works when a (heavy enough) ball is placed on top of the panel. The panel will tilt and attempt to center the ball in the midpoint of the panel. The response of the motor is smoothed and tunable with PID control. The PID control provides a feedback loop to predict future error, and allows the system to operate smoothly and accurately, without oscillations.

II. SUMMARY

The three major components in making this project operational were 3D modeling, PID tuning, and the hardware setup. A lot of work had gone into the 3D modeling because there needed to be some structural support other than just the 2 arms controlled by the servo motors, and we were very limited on time and money. Another hardship was programming the resistive touch panel. There was no datasheet included with the order, so there was a process of scouring the internet for examples of people using similar devices. Sources found are listed in the References section. This project heavily relies on accurate and fast ADC sampling to receive the X / Y coordinates for the panel control.

The communication protocols used consist of: **PWM** (Pulse Width Modulation) to specify which angle to move the servo to, **4x GPIO** (general purpose input output) for switching and sequencing the values from the resistive touch panel, and **UART** for debugging and representing the X / Y coordinates visually.

This project was chosen due to the relevance of what was learned in class, and its range of disciplines required. This project taught us how to interface with new sensors such as the touch panel and servo motors, as well as implement software algorithms that can be translated into real-life applications.

III. SYSTEM DESIGN

The ball balancing system works in this manner: The resistive touch panel gives X / Y coordinates from 0-4096 that are read by the ADC. The microcontroller filters and scales the X / Y coordinates, which are then processed by two PID controllers. The output of the PID controllers are sent to the PWM controlled servo motors to give an appropriate angle in order to balance the ball.

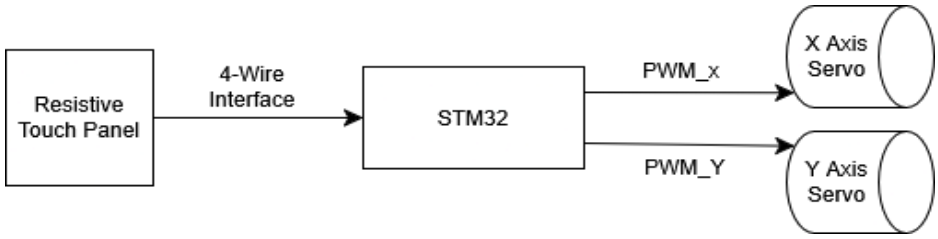


Figure 1: Overview of the System Design

III. HARDWARE ARCHITECTURE AND PIN ASSIGNMENTS

The project is wired as follows:

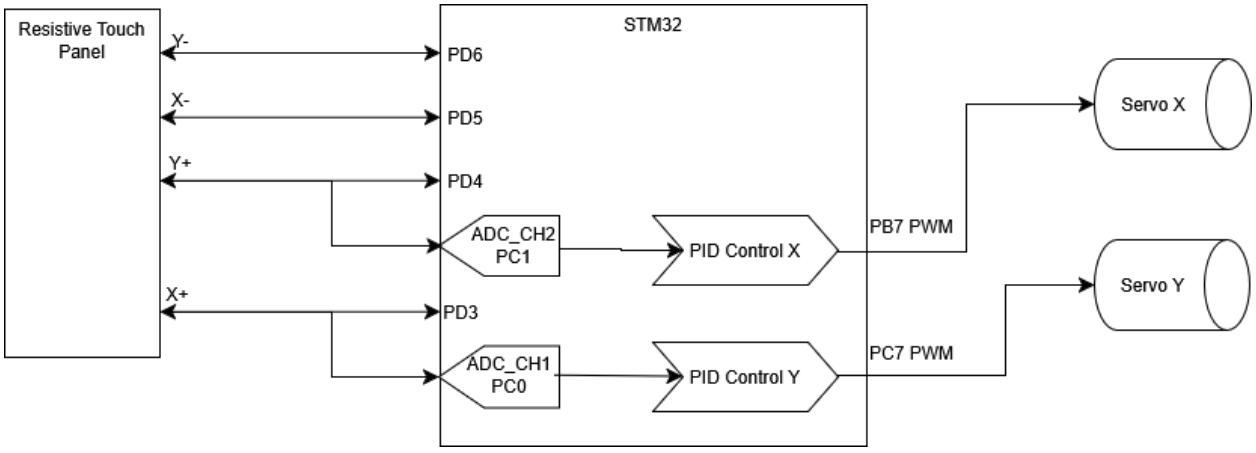


Figure 2: Hardware Design

COMMUNICATION	SIGNAL Direction	SIGNAL TYPE
x+ -> PD3	Input/Output	General Purpose
x+ -> PC0	Input	ADC1_IN1
x- -> PD5	Input/Output	General Purpose
y+ -> PD4	Input/Output	General Purpose
y+ -> PC1	Input	ADC1_IN2
y- -> PD6	Input/Output	General Purpose
PB7 -> ServoX	Output	TIM4_CH2 (AF2)
PC7 -> ServoY	Output	TIM3_CH2 (AF2)

IV. SOFTWARE ARCHITECTURE

In the current setup, nothing runs through the main. Before anything happens, timer2, the system clock, the LPUART1 clock and the ADC are initialized. Timer2 is set to generate an update flag every 50ms, this timing is important because the updates are fast enough for the PID to respond accurately to the ball placement and not too fast for the servos to change as well. The system clock is set to 16 Mhz, which was fast enough for the purposes of this project. LPUART1 was

used for debugging and to make sure that the values read from the resistive touch screen were coming accurately from the ADC. The resistive touch panel produces analog signals, which require an ADC to read and convert them for later manipulation. Global variables of the ADC's and PID's output are necessary to accurately update the functions in the Timer2_Handler. These variables are also necessary for the scaling and filtering done on the ADC's output before any handling of the PID.

The get_xy function handles the pin configuration necessary for extracting the X and Y data from the resistive touch panel. The raw X and Y values given by this function are filtered by an EMA (exponential moving average) algorithm so the PID isn't sensitive to immediate changes.This data is made into raw X and Y values given to be scaled considering the range of the ADC's value and what would be useful data for debugging purposes and meaningful data for the PID. The scaling is shown below:

```
float posX = 200 * ((float)filtX – ADC_MID) / ADC_SPAN;
float posY = 200 * ((float)filtY – ADC_MID) / ADC_SPAN;
```

The scaled X and Y values give the PID an idea about where the ball is physically located on a scale of -100 to +100 along each axis. These posX/posY values then serve as the PID's process variables—0 means "perfectly centered," positive means "tilted right/up," and negative means "tilted left/down." The PID uses the scaled X and Y values to calculate the error of the ball's current position and ball's balanced position. By tuning the PID's constants Kp, Ki, and Kd the servos will move the ball to a balanced position by the servos. The PID's output is then sent to the PWM controlled servos to move the platform to the most balanced position relative to the ball's current position.

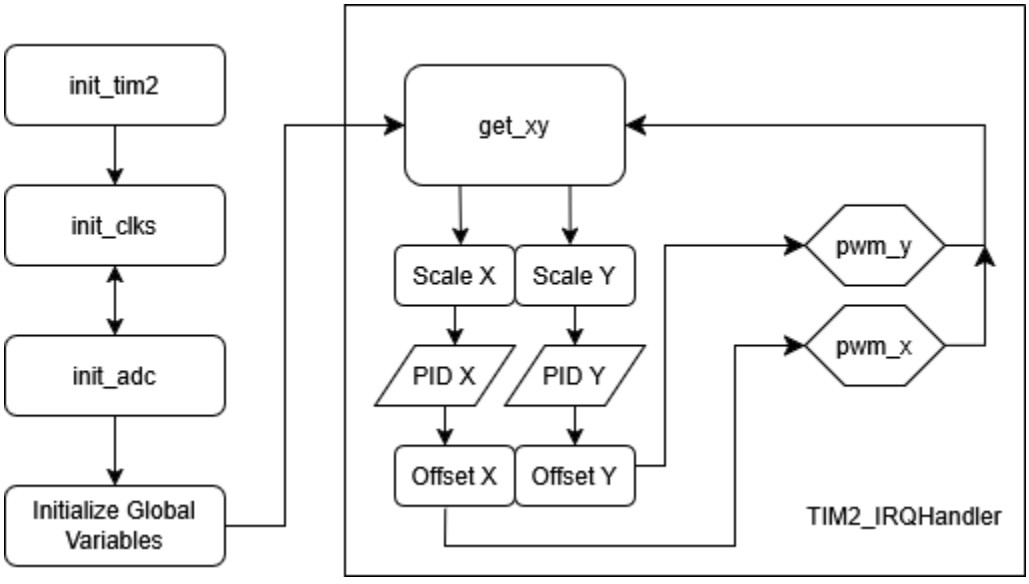


Figure 3: Software Design

V. COMPONENTS

Servo Motors x 2

Stepper motors were originally considered for their accuracy and torque, but we substituted these for servo motors due to their speed and power requirements. Using stepper motors would have required an external power supply that we didn't have in order to drive them.

Figure 4: Parallax Standard Servo

Key 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: Pulse-width modulation, 0.75–2.25 ms high pulse, 20 ms intervals
- Operating temperature range: 14 to 122 °F (-10 to +50 °C)

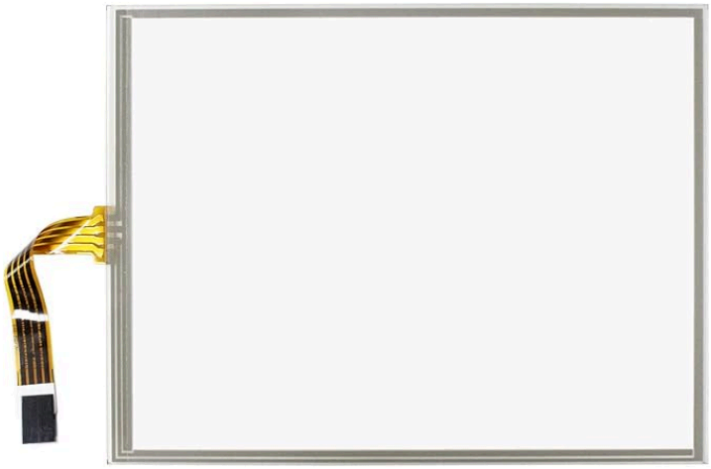
Our servos actually work in the 0.4 to 2.20 ms high pulse range. 0.4 ms translates to 0 degrees, and 2.20 ms translates to 180 degrees. In order to achieve this PWM range, we must set the pwm frequency to 50 Hz. This can be a calculated with the equation:

$$f_{PWM} = \frac{f_{clk}}{(ARR+1)(PSC+1)} \Rightarrow \frac{16{,}000{,}000}{(1999+1)(159+1)} = 50\ Hz$$

Having an ARR value of 2000 helps because it translates to the 20 ms period, and we can set the duty cycle exactly to the number of (ms / 100) we want the pulse to be high for. We achieved this by setting CCR2 = val + 40. This way, if the user passes in a val of 0, CCR2 = 40, and the servo will move to 0 degrees with a 0.4 ms pulse width. Similarly, if the user passes in a val of 90, CCR2 = 130, and the servo will move to 90 degrees with a 1.3 ms pulse width.

Resistive Touch Panel

The resistive touch panel is responsible for reading the X and Y values of the current ball position. It is powered by 4 different GPIOs, and read by two ADCs, one for the X axis and one for the Y axis.



Pins	Signal
1	Y_{UP}
2	Y_{LO}
3	X_{LE}
4	X_{RI}

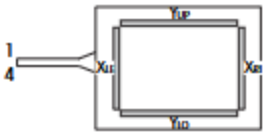
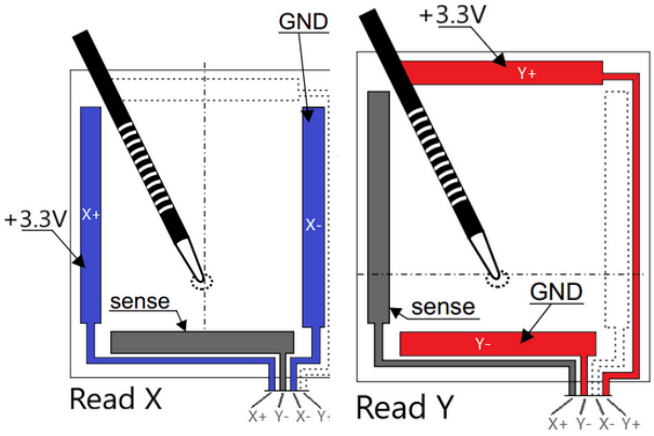


Figure 5: Resistive Touch Panel

The sequence of operations in order to read the X and Y values is as follows:

1) Set Y- and Y+ to tristate
2) Create a Voltage Divider between X+ and X-
3) Read the X Value from Y+

4) Set X+ and X- to tristate
5) Create Voltage Divider between Y+ and Y-
6) Read the Y value from X+



One important aspect of reading the X / Y coordinates is setting the unused pins to tristate (inputs / high impedance). This allows the unused pins to not interfere with the resistances currently being measured. After many hours of tinkering, this article gave us a thorough enough explanation to create our own implementation.

Another aspect of reading the ADC values was the speed at which the ADC was running. The values were not reading accurately when the ADC was running from its fastest speed all the way to 400 x slower. It wasn't until I slowed it down to a 700 x sample rate that the values were as accurate as I could get them. After some thought, I believe this is due to the RC time constant with the resistive touch panel, which grows as resistance increases, and the sampling capacitor in the SAR ADC. The panel normally has resistances from 0 - 500 Ohms, which is very high from the ADC's perspective.

VI. EMA Filter

The exponential moving average filter (EMA filter) is a recursive filter that uses the algorithm:

$$Y[n]=\alpha X[n]+(1-\alpha)Y[n-1]$$

Y[n] describes the output of the ema filter, while X[n] describes the current data. Y[n-1] describes the filter's most previous output. α is known as the smoothing factor ($0.1 < \alpha < 0.9$) applied to the current input that determines how responsive or smooth the input will be scaled. This wasn't strictly necessary for PID to balance the ball, but applying the EMA filter gives the PID controller a smoother position input and in turn a better response to the ball's error. An initial Y[0] must be given so that the recursion of the EMA filter has a starting point. Since the EMA filter is recursive, if a Y[0] wasn't initialized it would create an initial spike on startup which would counter the EMA's purpose. This Y[0] is the ADC value for the midpoint otherwise known as ADC_MID seen earlier in the scaling equation. The midpoint as the initial position is reasonable since the ball is most likely set in the middle of the platform each time the program is run.

VII. PID Controller

A controller is essentially a formula that provides a prediction based on the error between wanted output - the current output, or in the scope of this project the ball's desired balanced position - its current position.

$$e(t) = error = setpoint(0,0) - current\ point\ (X,Y)$$

The actions that are taken to drive the error to the desired position are known as Proportional, Integral, and Derivative (PID).

$$u(t) = K_p * e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt}e(t)$$

$$Kp = proportional\ gain$$

$$Ki = integral\ gain$$

$$Kd = derivative\ gain$$

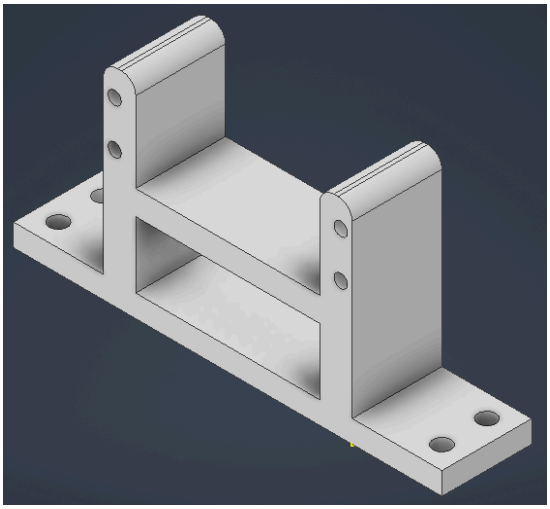
The proportional gain in this equation scales the current error to try and quickly offset the initial error. Integral gain scales the total accumulation of error over time to eliminate the ball wavering around the setpoint. While the derivative gain scales the error's rate of change to try and anticipate the balls next position. "t" is basically the sampling rate of the PID equation.

This was "integral" to the project since in order to keep the ball balanced, an idea of where it will be is necessary to apply an opposite force to maintain its place on the platform. This allowed the PID to continuously adjust the servos to where the ball really is in a responsive and accurate manner.

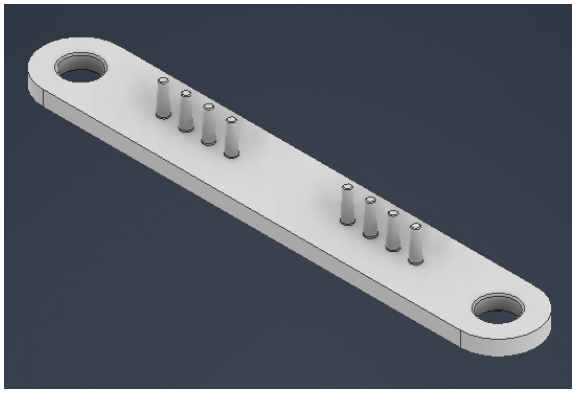
VIII. 3D MODELING

Due to not being able to find online designs that worked for the ball platform, we decided to 3D model our own mechanical parts. It did take some time to do this, but it was really fun doing everything from scratch, as well as probably saved us money from buying more expensive parts. We did all measurements with digital calipers, and used online youtube videos of similar projects for inspiration, however all parts are original designs.

Part 1: Servo Mount

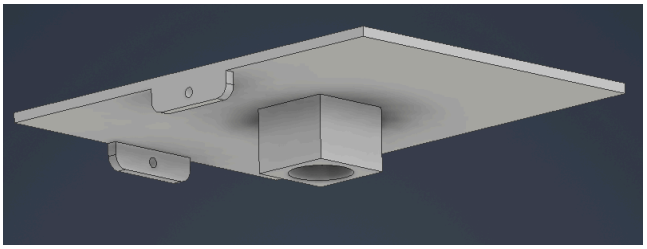


Part 2: Servo Horn

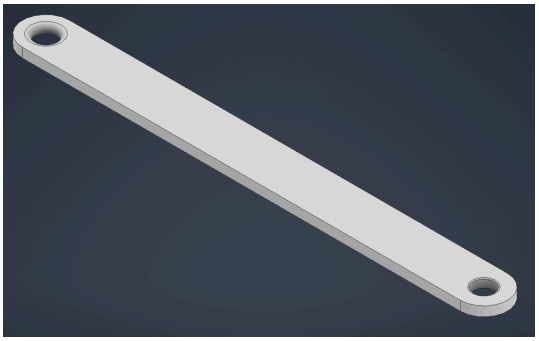


Part 3: Post

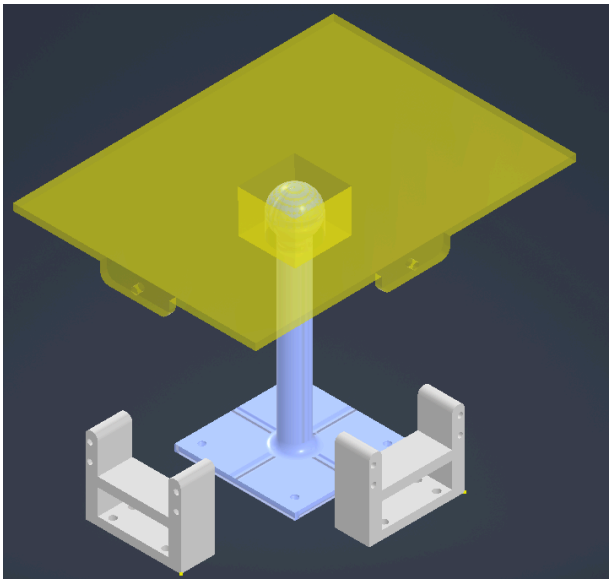
Part 4: Panel Platform



Part 6: Arm



Partial Assembly



IX. CONCLUSION

The PID controlled self-balancing robot was a fun project for us. We met the initial specifications of the project by getting the ball to balance even when we pushed it around, which was exciting to get. We also tried to implement a way to give the robot a set of positions to follow so that the ball would move in a pattern like a circle or a line, but the problem became more difficult than we originally thought and with time constraints we couldn't implement it in the demo.

Looking back on the project, it would have been to have a bigger platform to see the ball really move across, but we're very stoked with how it turned out. It was a great learning experience and it was cool to implement a PID controller after taking SMAC. Plus the experience in writing code and completing the project is always great to add to our portfolios.

X. REFERENCES

Servos: https://docs.rs-online.com/0e85/0900766b8123f8d7.pdf

Touch-screen https://www.instructables.com/4-Wire-Touch-Screen-Interfacing-with-Arduino/ / https://media.digikey.com/pdf/Data%20Sheets/NKK%20PDFs/FT_Series_4-Wire_Ds_Oct_2017.pdf

EMA filter:
https://fedevel.com/blog/the-simplest-digital-filter-stm32-implementation-phils-lab-92

PID controller: https://acrome.net/post/pid-controller-design-for-stm32-microcontrollers