EIE3105 Integrated Project Final Report Rev 1.1

The Hong Kong Polytechnic University Electronic and Information Engineering Year 3

Student Name: Cheah Yeok Tatt
Student ID: 18078696D
Group: Rescue Robot L09

Table of Contents

Overview 2
1. Mechanical Hardware Design 2
Overview 2
Technical Specifications 3
Design Process and Challenges 4
Stab V1 4
Stab V2 4
2. Electronic Hardware Design 5
Overview 5
Technical Specifications 5
Design Process and Challenges 6
3. Software Design 6
Overview 6
Code Flow Diagram 6
Writing a Test Program 6
Filtering IMU Data 7
Simple Kalman Filter 7
Combine IMU Data: Complementary Filter 8
Servo Motor Control 9
Control Flow Diagram of each P controller, one for each servo 9
4. Data Visualization and Tuning 10
Overview 10
Graphing 10
Numerical Values 10
3D Model 10
Input For Tuning 10
5. Discontinued Work 11
Image Recognition and Tensorflow Lite 11
Learning TFLite 11
Trying out YOLOv4 11
6. Appendix 12
Github Link 12
Design References 12

Overview

This is the final individual report for my part in the creation of L09's Rescue Robot. I'm in charge of the hardware and software design of the stabilizer. The stabilizer's task is to balance a 100 g 3D printed cylinder on the top as the robot passes through obstacles such as stairs and uneven ground. The design is a simple platform on a 2 rotational DOF actuator. This report will go through the technical specifications of each aspect and explain the major challenges faced, solutions chosen and the reasoning behind it.

1. Mechanical Hardware Design

Overview

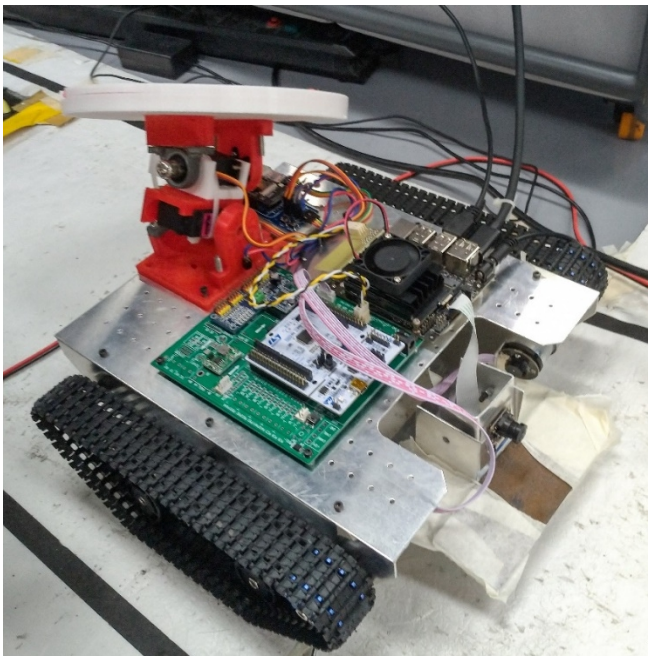


Figure 1 Rescue Robot with StabV2 on top

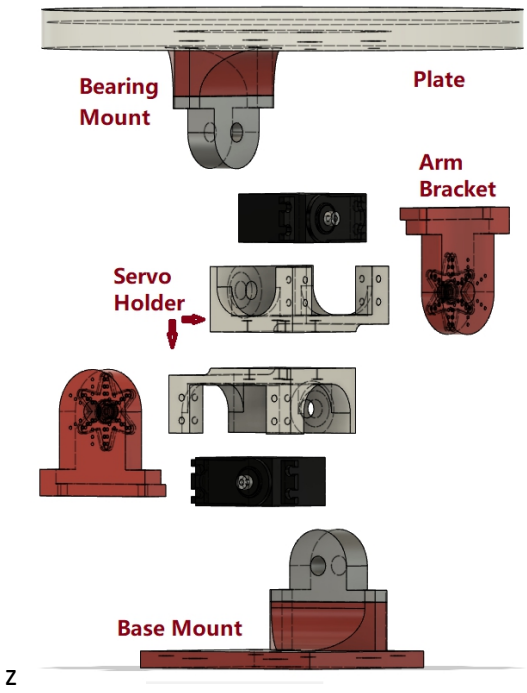


Figure 2 StabV2 Exploded View, Parts Labelled

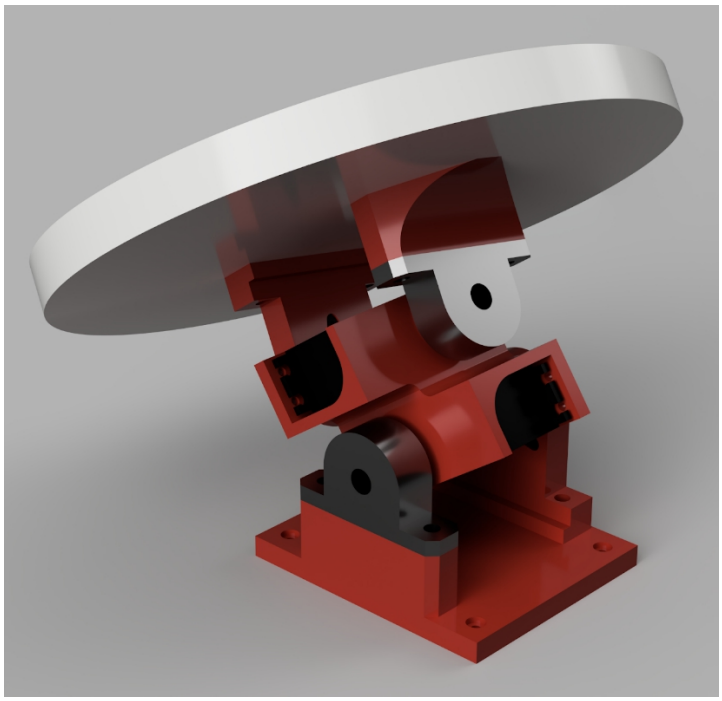
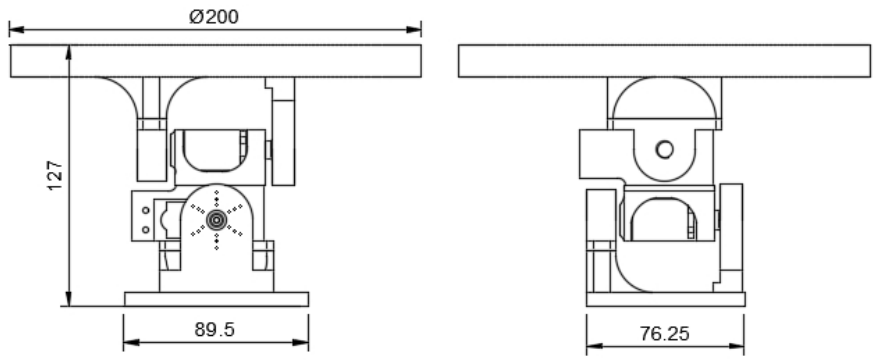
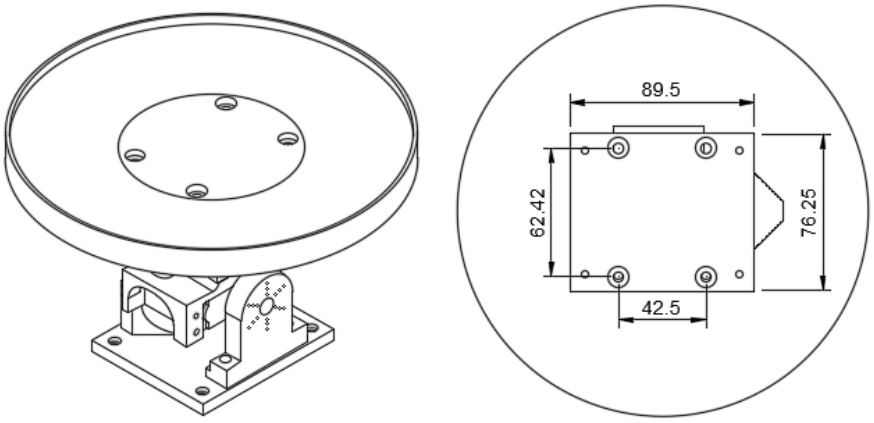


Figure 3 StabV2 Render in Fusion 360



Front View

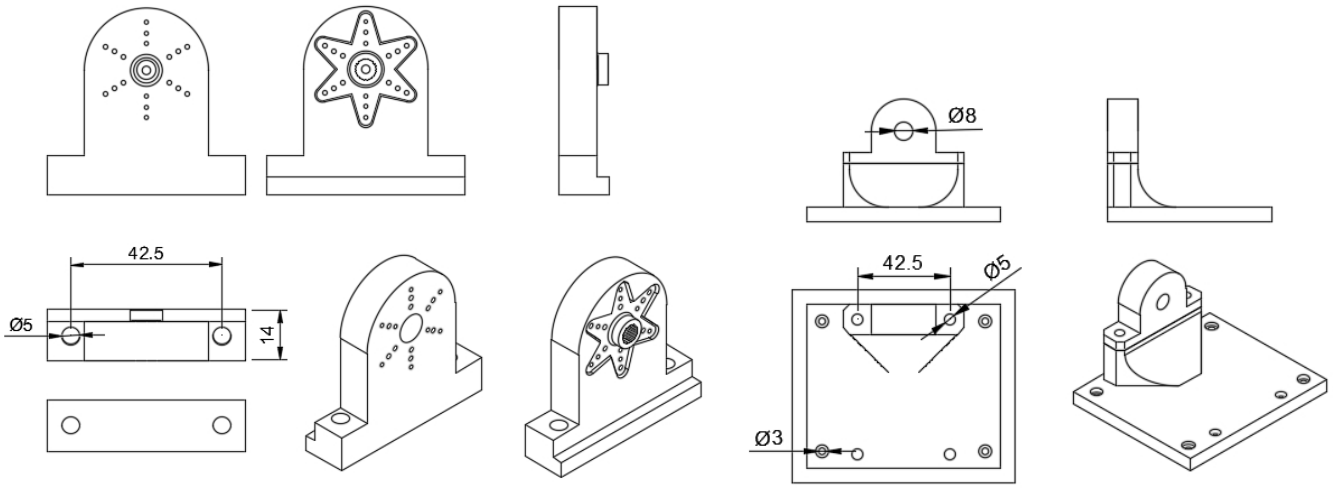
Side View



Isometric View

Bottom View

Stabilizer Overview



Servo Horn Holder for Standard 6-arm Plastic Horn

Bearing Holder

Design Process and Challenges

Stab V1

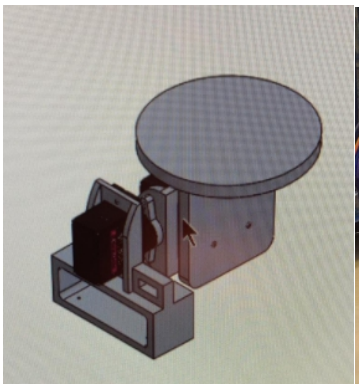


Figure 4. Previous Stab Design

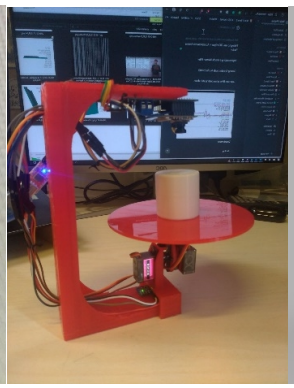


Figure 5. Stab V1, Functional

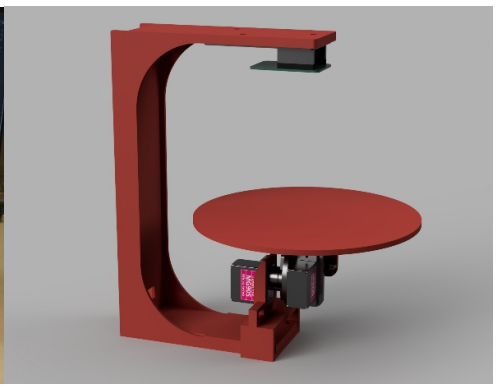


Figure 6. Stab V1, Render in Fusion 360

After improving on the previous year's design by adding several additional holes for easier assembly, enlarging the plate greatly and also adding a camera stand, I arrived at StabV1. The idea was to use the micro servos to balance the object in fast moving situations such as going up and down the stairs, and then use the ESP32-CAM to provide feedback regarding the object's position in the long term in order to move the object back to the centre by intentionally initiating tilts.

The stabilizer was designed to use micro servo motors because we believed that lighter servo motors will allow for a lighter design, which would require less current, be less punishing when errors occurred and is more fitting for a fully 3D printed design as 3D printed material is not very strong. After comparison, MG90D was chosen in the end due to its internal digital control signal and metal gears, which should provide the most robust control system and hardware performance.

Servo	Weight (g)	Signal Type	Gear Type	Torque (kgcm)	Speed (sec/60deg under no load)
SG90	9	Analog	Plastic	1.6	0.23
SG92R	9	Digital	POM/Carbon Fibre	2.5	0.1
MG90S	13	Analog	Metal	2.2	0.08
MG90D	13	Digital	Metal	2.4	0.1

Figure 7. Comparison Between Micro Servos

After several weeks however we were disappointed by this design's performance.

1. The lack of support on the other side of the actuator meant that the 3D printed material would flex and wobble in the operation of the robot
2. Despite the metal gearing of the servos, the servos would still sometimes get jammed and lead to inconsistent performance. The only solution is to switch out the motor, which was time consuming. This was also an issue I wish to not face during demonstration day.
3. The plate is too slippery and the object may just slip out

Stab V2

StabV2 is the current version of the stabilizer. It is designed with robustness in mind, having support on the other side of the servo's turning axis using 628ZZ pillow block bearings. We used MG996R motors to control the much larger plate, and we also added non-slip material to the top of the plate to increase the grip. The performance proved much better however the additional weight and power cost. The additional weight of the stabilizer changed the CoG of the robot, which proved to make tasks such as stair climbing to be more difficult but was still overcome during the testing on Monday by sufficient tuning. The additional power cost of the MG996R motor caused some over-current problems, which we also overcame (more details in software design section).

2. Electronic Hardware Design

Overview

I chose to use a separate microcontroller to control the stabilizer. This is because our team wanted to delegate work better by separating the task of programming and building the stabilizer from the main program of the robot car. The microcontroller that we chose was ESP32-CAM, Ai-Thinker module. It has integrated WIFI, Bluetooth, a fisheye OV2640 Camera and a SD Card Reader. Relevant features:

- Up to 160MHz clock speed, summary computing power up to 600 DMIPS - important for the many floating point operations necessary for our implementation.
- Built-in 520 KB SRAM, external 4MPSRAM - important for simple on board image processing
- Supports UART/SPI/I2C/PWM/ADC/DAC - important for communications with servos and IMU

Hence a separate integrator PCB was designed in order to minimize the messiness of jumper connections and increase the robustness of the robot. ESP32 boards also require the user to pull low the DOWNLOAD and RESET pin at once in order to enter program download mode. Since ESP32-CAM modules do not have this implemented as a button circuit, I also included it into my PCB design.

Another goal of the PCB was also to provide a power delivery channel for the servos without passing through the PCB of the ESP32-CAM module. The thick 1.5mm power supply routes works flawlessly on a lab bench power supply, and was meant to connect directly to the 5V power supply bus of the main PCB. However I realized that the main PCB has no direct jumper connection to the 5V power supply, instead I connected it onto one of the servo outputs pins of the PCA9685 Module.

Technical Specifications

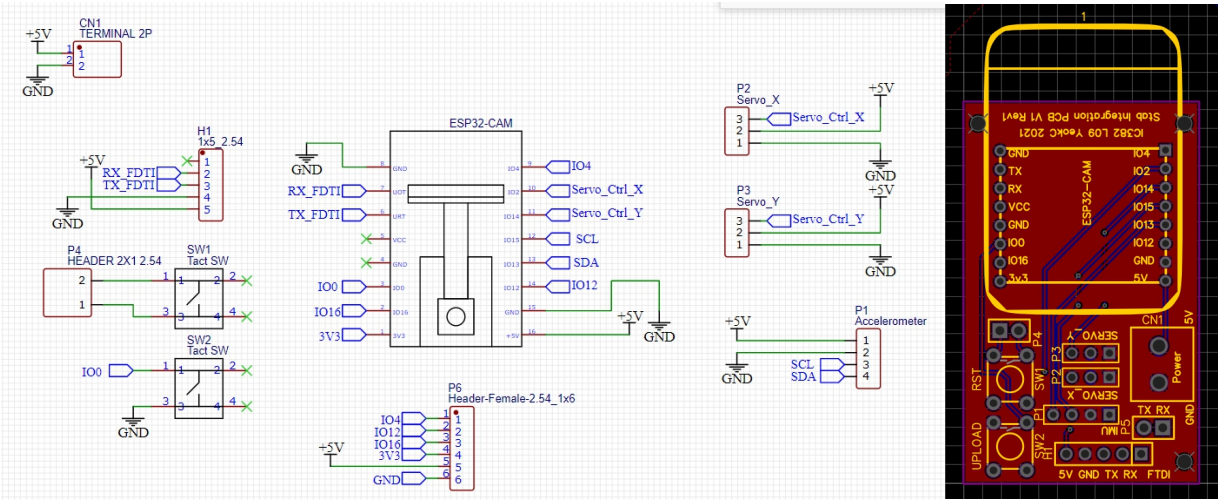


Figure 9. PCB Schematic *Figure 8. PCB Layout*

Design Process and Challenges

Initially, the first version of the PCB I designed had the camera facing the wrong way after being plugged in. Due to the short wire, I couldn't turn it either. I had to send a new version of the gerber file for manufacturing again. It was quite embarrassing but didn't really cost the overall project anything.

An interesting note is that although jumper wires, when tied up with zip-ties and taped together at the connections are quite stable, it would've been better if I could've substituted the headers with 2.54mm Molex Crimps. That way, the connections would look more professional and have even better stability.

Another interesting note was that having the mounting holes within the outline of the ESP32-CAM module was a pain as I couldn't screw/unscrew it when the MCU is connected – giving priority for accessibility and assembly concerns over design efficiency can also be a future improvement.

3. Software Design

Overview

The goal of the software is to

1. Filter the IMU sensor data
2. Combine the IMU sensor data
3. Control the servo motor

Code Flow Diagram

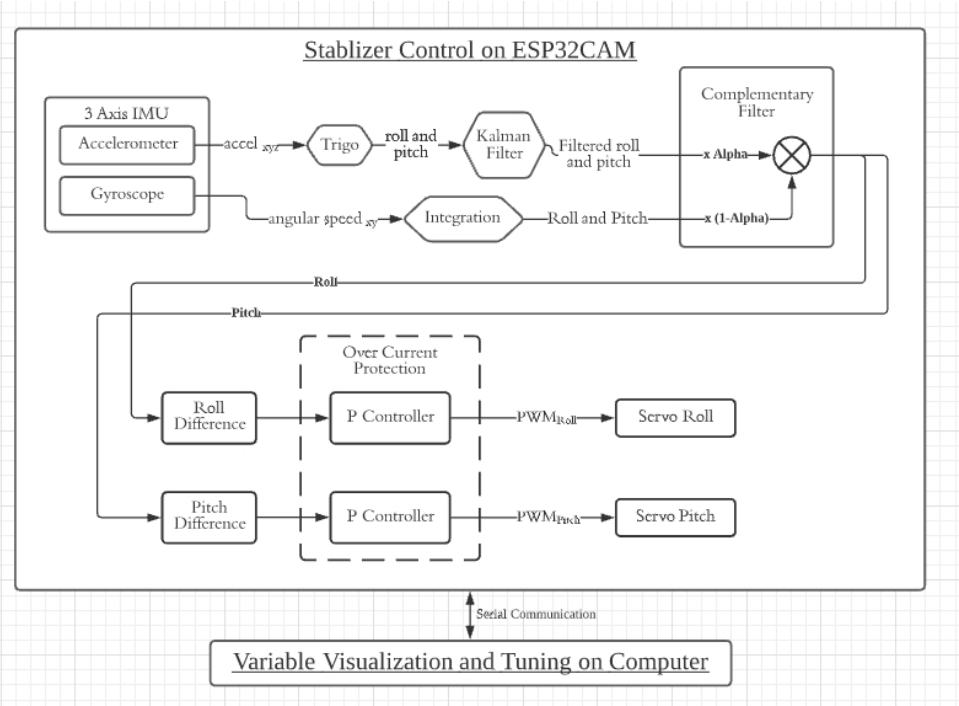


Figure 10. Code Flow Diagram

Writing a Test Program

The goal of the program is to figure out how to control servo motors using our MCU (ESP32-CAM) and read IMU data without them interfering with each other, as both the I2C communication of the MPU6050 and the PWM signal for controlling servos may rely on the same timers.

I first achieved basic servo control that I can control with serial input by typing on the keyboard. By using an external bubble level, I wrote a function that uses serial input from the keyboard to adjust and calibrate

the servos so the plate is at a flat position. I then used simple I2C code to read from my MPU6050 and unparsed the accelerometer data, and use serial port to print it onto my terminal on the computer.

Filtering IMU Data

I first attempted several filters before settling on a simple Kalman filter. The list includes:

- **Madgwick AHRS Algorithm** – Response was too slow, for some reason. However it was very stable.
- **RTMULib** – Requires at least 9DOF to be computed but would prove useful otherwise.
- **Extended Kalman Filter** – Unable to be tuned, somehow.
- **TJK's Kalman Filter** – Takes in both accelerometer and gyroscope values for roll and pitch and uses matrix equations to do both quarternion conversion and filtering – I found it over complicated

My attempts included changing the necessary parameters such as sampling time, measurement uncertainty as close to known values as possible, and then tuning the parameters. My conclusion from all the attempts is that I do not understand Kalman filter well enough to write my own implementation or even tune it. Therefore I went ahead studied some online lectures.

Simple Kalman Filter

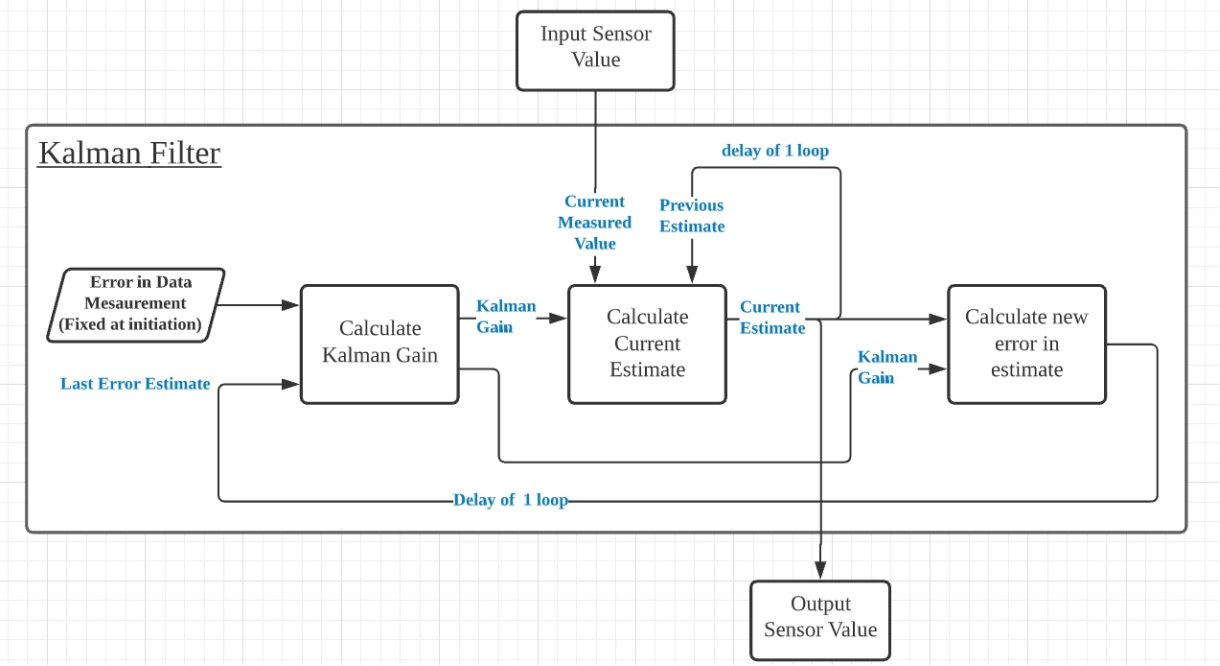


Figure 11. Kalman Filter Flow Diagram

I found that the most basic form of the Kalman filter was very intuitive and understandable. It can be split into three parts that may not seem to be in chronological order, however it is not a problem when implemented in a looping code. Viewing common implementations of Kalman filter in other people's repos also helped me understand this structure. In the end I implemented these mathematical equations in code using the "Simple Kalman Filter" Library

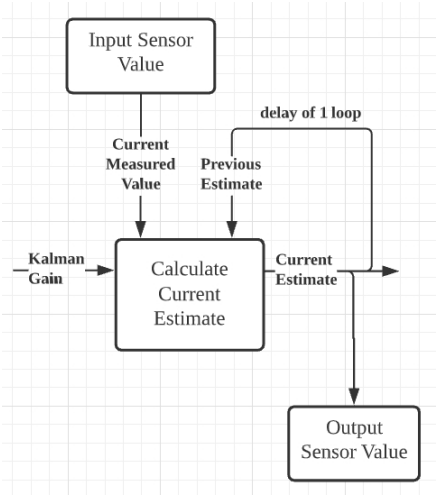


Figure 12. Kalman Filter Equation 1

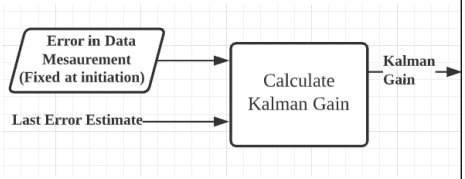


Figure 13. Kalman Filter Equation 2

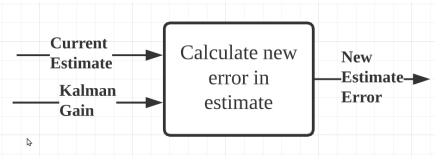


Figure 14. Kalman Filter Equation 3

Current Estimate (Est_t, ie. output) is calculated by essentially taking a ratio between **Current Measured Value** ($Meas$, ie. input) **and Previous Estimation of Value** ($E_{st\,t-1}$)

This ratio is the **Kalman Gain** (K_{gain}).

Note:
Est_t and Est_{t-1} are Estimation of Value for t and t-1 respectively

$E_{est\,t}$ and $E_{est\,t-1}$ are Estimate Error (or more accurately, Error in Estimation of Value) for t and t-1 respectively

Equation:

$$Est_t = E_{est\,t-1} + K_{gain} * (Meas - Est_{t-1})$$

Kalman Gain (K_{gain}) is the mathematical representation of the "trust-worthiness" or "weight" of the current measured value compared to the previous estimated value.

In our implementation it is calculated as a ratio based on the **Error in Data Measurement** (E_{mea} , tuned constant) and **Last Estimate's Error** (E_{est})

Equation:

$$K_{gain} = \frac{E_{est}}{E_{est} + E_{mea}}$$

New Estimate Error ($E_{est\,t}$) is the amount of uncertainty in the measurements for this loop (uncertainty in this estimate). This is then passed into a delay and used in other parts of the equation.

This can be calculated based on the **Previous estimate** ($E_{est\,t-1}$) and the **Kalman Gain** (K_{gain})

Equation:

$$E_{est\,t} = (1 - K_{gain})(E_{est\,t-1})$$

Combine IMU Data: Complementary Filter

Less of a filter and more of a combination method, so that the resulting summed signal still stays at the same scale as the input signals (since it adds up to 1, its complementary). Alpha is the confidence of one sensor's reading over another. The alpha used in the program is 0.75 but it is subject to tuning.

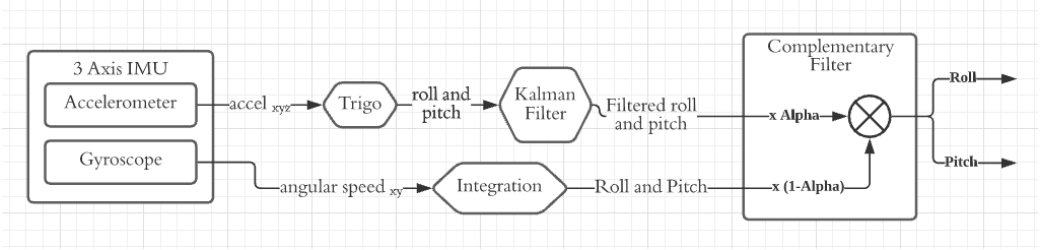


Figure 15. Full Flow Diagram for IMU

Servo Motor Control

Control Flow Diagram of each P controller, one for each servo

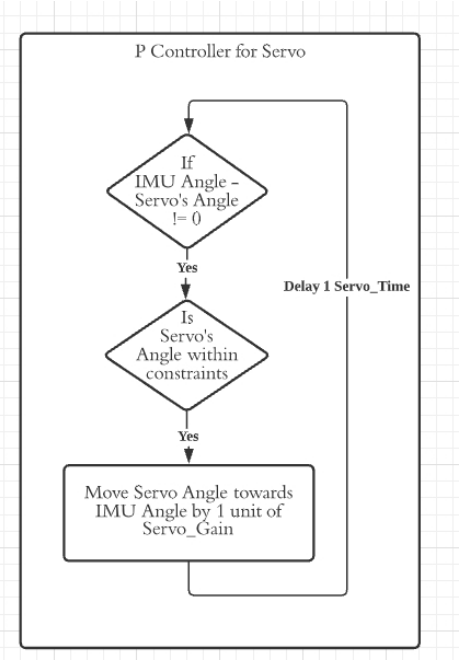


Figure 16. Flow Diagram for Servo Control

Sometimes the servo motors would draw too much power and cause the brownout detector to trigger on our MCU. I believe this is because of several reasons:

1. Bad power delivery between main PCB and Stab PCB, currently it draws power through one of the ports of the PCA9685 bus, which must have very thin tracks, suited to only control 1 motor but we're using it to power 2 motors and 1 MCU.
2. Bad software design as initially I didn't implement any protection or speed limits and gave the servo's internal PID controller to draw as much power as necessary.

In order to control the servo motors without causing overcurrent issues, I wrote this pseudo P controller in order to limit current draw at the expense of response time. After tuning, I arrived at

Servo_time (delay) = 10ms
Servo_gain = 2 degree

Additionally I also implemented angle limitations of 30 degrees pitch so the stabilizer wouldn't hit anything, and a limit of10 degrees roll for robustness since we do not require much roll for our tasks.

4. Data Visualization and Tuning

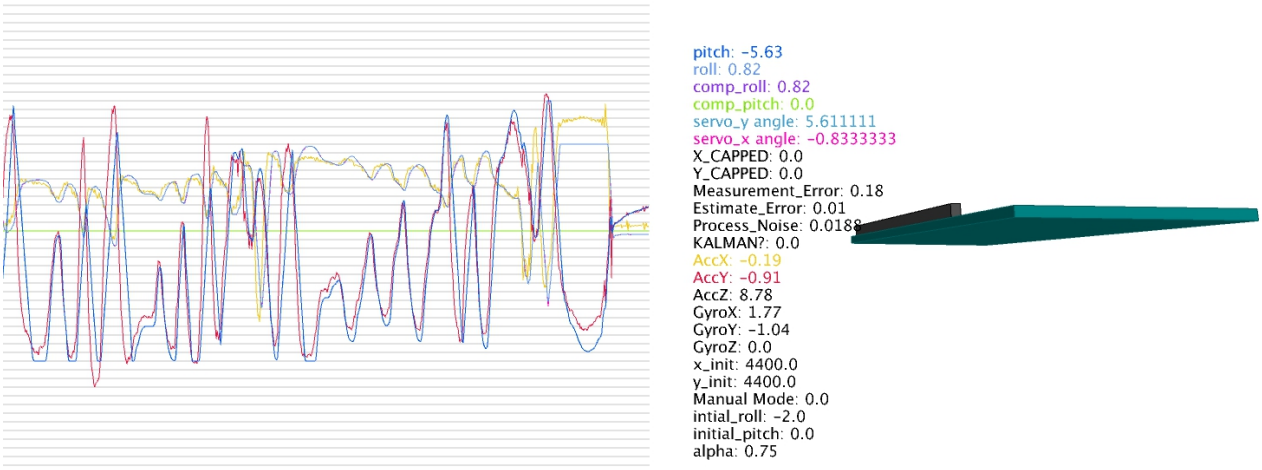


Figure 17. Visualized Data on Processing 3 (Java) Program

Overview

One thing that I found consistently helping throughout the software design and debugging process was the ability to visualize each stage of the data, as it is passed from filter to filter and eventually used for actuation.

Therefore I modified and combined TJK Electronic's data visualization code that was written on Processing 3 in order to both graph data, display the 3D model of the stabilizer, as well as provide numerical values for reference.

Graphing

I found the graph to be especially useful for seeing the delays between real time movement, as well as delays between the raw IMU readings and our filter. During tuning, we can clearly see the trade-off of between the filter delay and data accuracy.

I also use it to see when certain constraints are triggered, for example rotation speed and pitch and roll limits, and the response time trade off when it happens.

Numerical Values

I found the numerical values useful when figuring out where to set certain thresholds. It is also very useful during calibration as it tells me the exact position of the servos at every moment in time. I also use it as the legend of the graph to show the color of each variable and plot line.

3D Model

The 3D model was useful to understand response time, as it makes it very clear when a movement starts and stops. The amplitude and frequency of jittering and overshooting is also very clear on the 3D model.

Input For Tuning

I implemented some code to allow me to change the variable on the ESP32-CAM through a computer in real time. I used it for the following:

1. Tune Kalman Filter Values
2. Tune Complementary Filter Values
3. Activate / Deactivate certain filters to see the difference in response
4. Switch into manual mode and use keyboard to move the servo to flat position (calibration)

5. Discontinued Work

Image Recognition and Tensorflow Lite

Image recognition with ESP32-CAM was something I dedicated a lot of time to learning and developing, but it never made itself into the final version.

Learning TFLite

During the semester, I completed a book about deploying TensorFlow Lite on Microcontrollers: TinyML - Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. The book gives a good conceptual overview of different machine learning methods, functions, and goes deep into how to design your own TFLite implementations. In terms of practical exercises, it goes through multiple worked examples, in which they explain each line of code and the role they play. I also went ahead and implemented the first few examples on the ESP32-CAM, including sine wave approximation and wake word detection through TFLite.

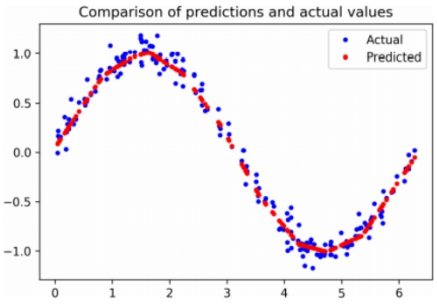


Figure 18. Sine Wave Reconstruction, picture screenshot from Google Colab because I never wrote the data visualization code on ESP32-CAM, but the RMS Values proved to be within the acceptable accuracy.

I also tried converting several models from .h5 and .pb files into .tflite files, then into a .h FlatBuffer that could be uploaded onto microcontrollers. I also tried several levels of quantization in the previous examples and saw how little it affects the final performance. Although I never had the time (and the need) to do it, the next step of implementing TFLite models onto an MCU would have been:

1. Instantiate an Interpreter object.
2. Call some methods that allocate memory for the model.
3. Write the input to the input tensor.
4. Invoke the model.
5. Read the output from the output tensor

Trying out YOLOv4

After all the talk from my teammate Kaleb on how good YoloV4 is, I tried installing it, building a model on my computer and running inference on my PC as well. The result was surprisingly good and despite the tiny model size, I don't think I could've ever ported it onto the ESP32-CAM.

6. Revision History

Rev 1.1 - Clarified Kalman Filter explanation and misplaced formula.

Rev 1 - Original release

7. Appendix

Github Link

The repo contains all my code, 3D printing files, PCB design files
https://github.com/Yeok-c/IC382_ESP32CAM

Design References

ESP32-CAM General Usage

ESP32-CAM-AI-Thinker-schematic-diagram.png (1222×810) (amazonaws.com)

ESP32-CAM Camera Usage

ESP32-CAM Take Photo and Save to MicroSD Card | Random Nerd Tutorials

Change ESP32-CAM OV2640 Camera Settings: Brightness, Resolution, Quality, Contrast, and More | Random Nerd Tutorials

esp-who/examples/single_chip/camera_web_server at master · espressif/esp-who (github.com)

Easier, faster pure video ESP32 cam motion detection (eloquentarduino.github.io)

Adding features to Examples->ESP32->Camera->CameraWebServer - ESP32 Forum

esp-camera returning half pictures when high quality or higher resolution than SVGA - ESP32 Forum

Writing read_jpeg and decode_jpeg functions for TensorFlow Lite C++ - Stack Overflow

TensorFlow Lite inference

Transfer pictures from ESP32-CAM to ESP32 via serial - Using Arduino / Programming Questions - Arduino Forum

ESP32 CAM real time image processing. - ESP32 Forum

Filtering

Open source IMU and AHRS algorithms – x-io Technologies (x-io.co.uk)

TKJElectronics/Example-Sketch-for-IMU-including-Kalman-filter: Software for "Guide to gyro and accelerometer with Arduino including Kalman filtering" (github.com)

MPU-6050 | Mbed

Complementary filter - My IMU estimation experience (google.com)

Guide to gyro and accelerometer with Arduino including Kalman filtering - Using Arduino / Sensors - Arduino Forum

Lecture 10 Inertial Measurement Units II.pdf (stanford.edu)

Stab Servo Control

Improving the Beginner's PID – Introduction « Project Blog (brettbeauregard.com)

Stab Designs

Self-Balancing Platform HOW TO COMPENSATE FOR IMBALANCE WITH FEEDBACK FROM AN IMU