**Summary of** **ITB de Labo’s**

**Extended Kalman Filter (EKF) Research Activities in 2022**

* 1. **Outline**

1. **Introduction**
2. Background Problems
3. Fundamentals of Kalman Filter
4. **Mathematical Derivation and Programming Integration**
5. Algorithm Procedures
6. Kinematics Model
7. Prediction Step
8. Update Step (measurement & fusion)
9. Headings Callibration
10. **Research Output**
11. Simulation & Tuning
12. Future Prospects
    1. **Reports**
13. **Introduction**
14. **Background Problems**

The development for MSD700 project needs information of the robot’s position at any time to enable a fully autonomous system. For outdoor use, GPS is the primary solution to obtain the global position, but its current performance still left much to be desired. The problems relating to the GPS’s position acquisition performance can be listed as follow:

* The precision and accuracy of GPS’s position and its displacement are too dependent on the environment, such as the number of satelites detected, surrounding tall bulding, cloudy sky, etc. (highest error up to 6 m)
* The GPS’s compass and IMU data is unavailabe because of module communication issue, which means there is no headings information.
* The measurement frequency is still too low for autonomous control implementation (close to 1Hz or 1 meas./sec.)

Those problems can be alleviated through sensor/data fusion. One of the most common solution is by implemeting Kalman Filter. ITB de Labo creates the kalman filter algorithm and the simulation environment on MATLAB, and convert it to Python script so that it can be used on Raspberry Pi minicomputer. The algorithm is deployed on MSD700’s prototype with the architecture is as shown on **Figure 1**.

Timeline

Description automatically generated**\\**

**Figure 1.** MSD700 prototype’s architecture

1. **Fundamentals of Kalman Filter**

Kalman filter is an algorithm that fuses (combines) a series of measurements and predicted states of a system to produce the best (updated/corrected) state estimates. The estimation (data fusion) is based on system’s mathematical model, previous states, control/driving inputs, noisy values, and their probability distributions overtime. Kalman filter exclusively assumes that the probability distributions is Gaussian, which means the probability is normally distributed (the mean/expected value is in the middle).

For GPS’s problem in obtaining the best position (state) estimates, **Figure 2** illustrated how kalman filter works. From previous position () and its probability distribution (), we can predict **(1)** curent position (). Since the prediction is calculated with a mathematical model which may not describes the system prefectly, the predicted position’s probability distribution () is wider. In other words, it has high probability of deviating from the actual position. The model derivation is further explained on page xxx

Further prediction will only continue to lower the confidence in the predicted position. This is where the measurements **(2)** comes to play. The measured position () may be different than the predicted position () and it has its own probability distribution (). The predicted value and measured value is then fused **(3)** to estimate the actual position () and its new probability distribution (). This estimated value will have a better probability distribution (smaller distribution area, higher confidence in expected/mean value). The mathematical operation is further explained on page xxx

For easier visualization, the fusion (update/correction) step can be thought of as taking the overlapping area in a Venn Diagram as shown in **Figure 3**. In the sense of probability distribution, this sliced area between two circle (prediction and measurement) has smaller area (less deviation, narrower probability distribution) and you are sure that it has the same member as both circles (higher confidence of being correct).

Shape

Description automatically generated

**Figure 2.** Kalman Filter ilustration ([Machine Learning TV](https://www.youtube.com/watch?v=LioOvUZ1MiM))

Diagram, venn diagram

Description automatically generated

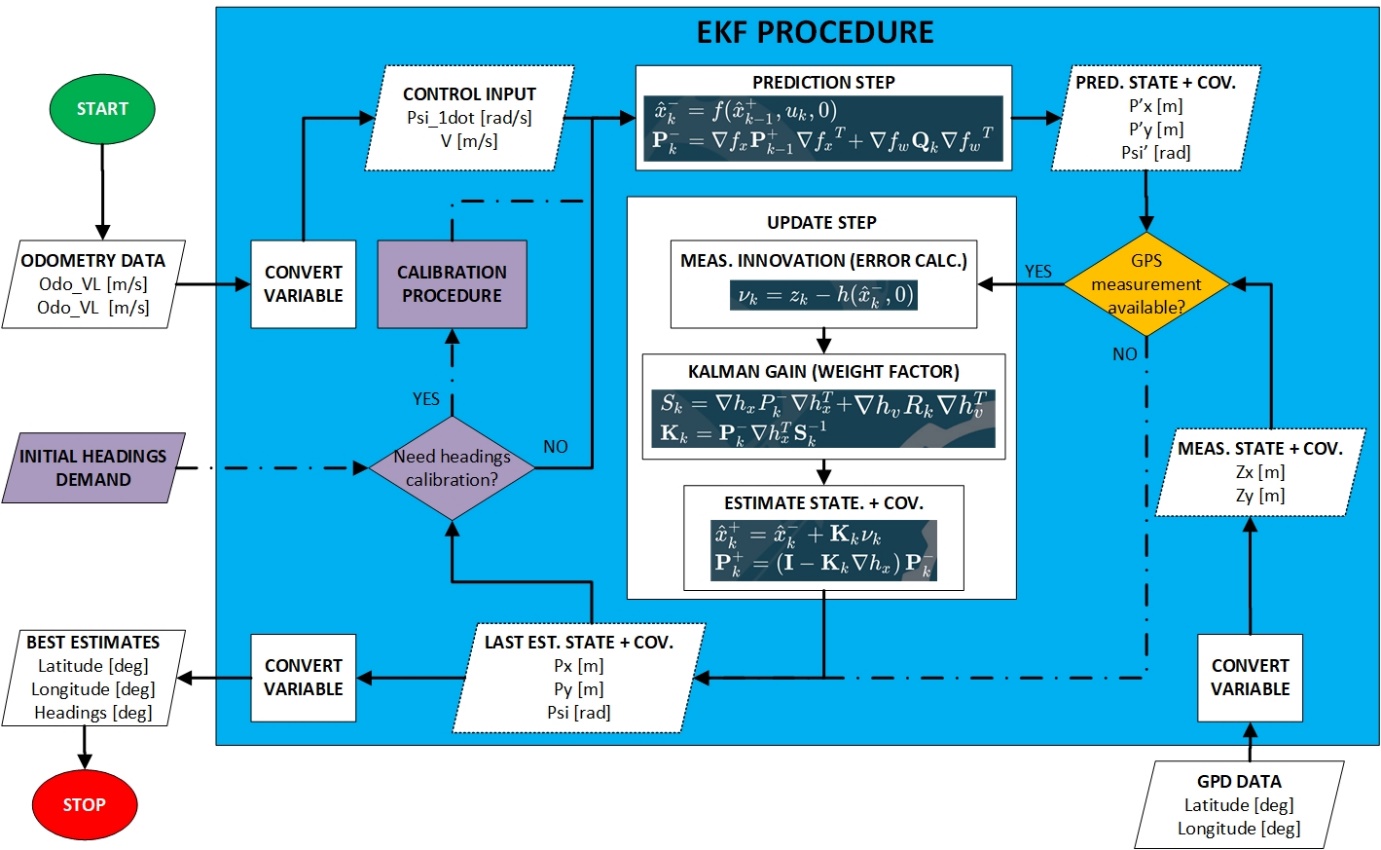
**Figure 3.** Venn Diagram visualization

For our case, those procedures are run for each time step which creates an iterative (looping) process as seen in **Figure 4.** This process is done in a microcontroller and/or minicomputer. The GPS problems described previously are solved as follows:

* GPS’s precision and accuracy are improved by estimating the position (and displacement if necessary by doing simple math outside the EKF process (after – before position) through combining (fuse) both measurement and prediction value.
* Headings can now be calculated through prediction step based on the estimated position, though a suitable callibration is needed for acceptable prediction/estimation (further explanation on page xxx)
* Whenever a new GPS measurement is not yet available, the update step (measurement and fusion) can be bypassed and the position is obtained solely from prediction step. This settings provides more position data at much higher frequency compared to being dependant on GPS only.

The next part will explain the mathematical and programming aspect of Extended Kalman Filter (EKF), one of the most versatile kind of kalman filter. The notation which is used on further explanations may be different from the ones introduced previusly, but the whole concept is the same. In addition, the integration of EKF on MSD700’s prototype will also be described for better context. Readers are expected to have basics on mathematics, programming to understand this documents, but the basics of statistical importance will be brieflly explained in page xxx

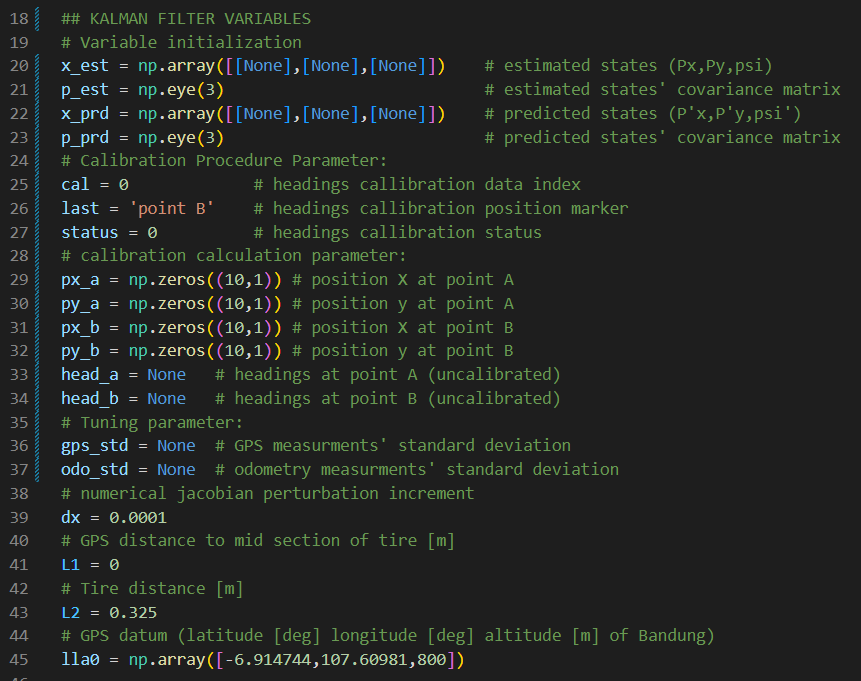
1. **Mathematical Derivation and Programming Integration**

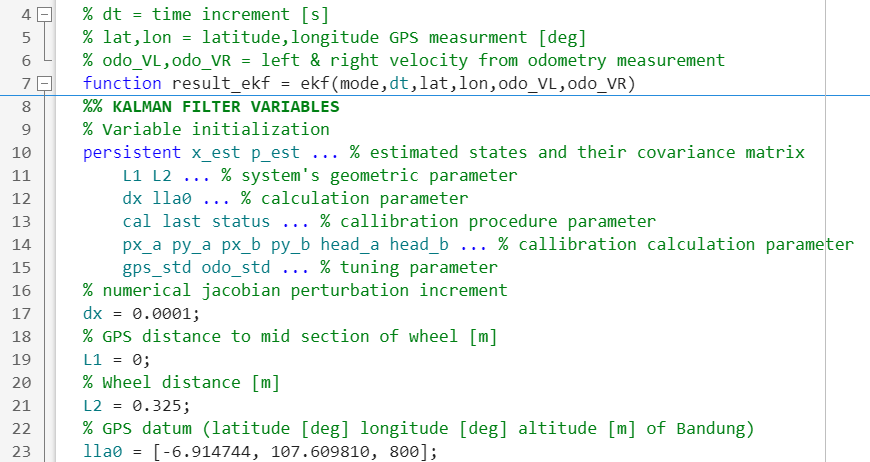
**Figure 4.** EKF flowchart

1. **Algorithm Procedures**

**MAIN PROCEDURE**

Based on the flowchart on **Figure 4**, the EKF algorithm main procedure is denoted by solid arrow. This procedure is the same as the one explained on introduction in which odometry data is used for prediction step while GPS data and the predicted state are combined through update step to obtain the state estimate. All the required variables can be seen on **Figure 5** as they were initialized on the program (code). The algorithm’s input are the left and right wheel’s velocities (in m/s) from odometry and GPS’s latitude and longitude (in degree), while the output is the estimated/updated latitude, longitude, and headings (all in degree). The purpose of each variable will be explained later on in this document.

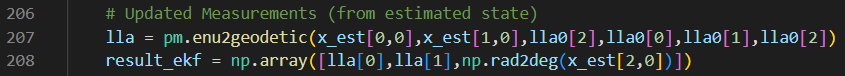
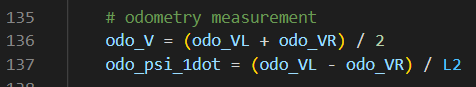
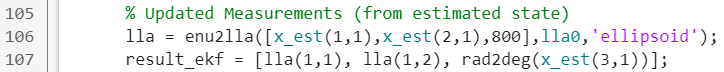
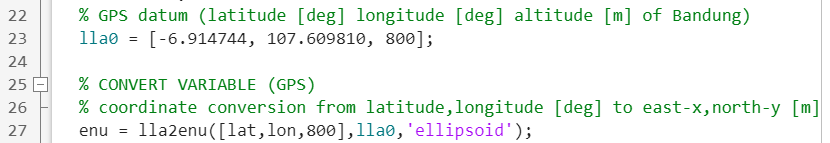
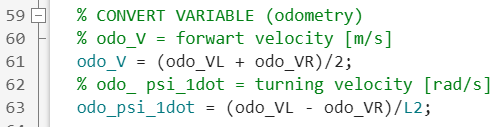




**Figure 5.** Variable Initialization (black = Python, white = MATLAB)

**VARIABLE/UNIT CONVERSION**

To simplify the mathematical operation, positions are calculated as x and y coordinate in meter (m) and headings in radians (rad) with the control input of the system being forward velocity (m/s) and turning velocity (m/s). A global position datum is used for position x = 0 and y = 0, and the position is converted to x-east and y-north in reference to the datum through a readily available function both in MATLAB and Python. The calculated heading is also converted from radian to degree with a readily available function as well. The left and right wheels’ velocities are converted to the desired form using the kinematics model of the systems that will be explained in detail on page xxx. This procedure is denoted by the “convert variable” block on **Figure 4** and written in code as shown in **Figure 6**.

**Figure 6.** Variable/unit conversion (black = Python, white = MATLAB)

**AUXILARY PROCEDURE: BYPASSING UPDATE STEP**

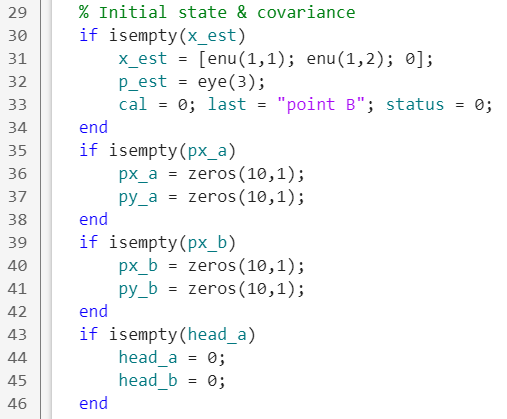
Whenever the GPS’s measurements are unavailable, the update step is bypassed and the predicted states become the output. This procedure is denoted by the orange block and dotted line on **Figure 4**. This is necessary because GPS has lower data acquisition rate than odometry and is susceptible to signal-loss which render the update state ineffective at times.

**AUXILARY PROCEDURE: HEADING CALIBRATION**

In addition, headings calibration is required to ensure a correct estimations of heading. This is necessary because we do not have the means to measure the heading directly (do not have other value for comparison/fusion). Hence, the initial heading need to be as close to the truth as possible. The calibration procedure is conducted only for initial depolyment of the architecture. This additional step is shown by the purple block and dotted line on **Figure 4** and explained in detail on page xxx

**INITIAL STATE & COVARIANCE**

Beside the heading, EKF also need to have initial position and its probability distribution (the value in reference to the previous introduction) at the start of its deployment. This initial value should not be arbitrary to prevent a divergen estimation result. GPS measurement is used for this, which means the algorithm can only be deployed after the first GPS measurement. The initial probability distribution (in the form of covariance matrix) though can be arbitrary, as shown in **Figure 7**. The statistical meaning of probability distribution and covariance for EKF algorithm will be further explained briefly on page xxx



**Figure 7.** Initial State and Covariance Matrix (black = Python, white = MATLAB)

1. **Fundamental Statistics**

The development for MSD700 project needs information of the robot’s position at any time to enable a fully autonomous system. For outdoor use, GPS is the primary solution to obtain Filter.

1. **Kinematics Model**

The development for MSD700 project needs information of the robot’s position at any time to enable a fully autonomous system. For outdoor use, GPS is the primary solution to obtain Filter.

1. **Prediction Step**

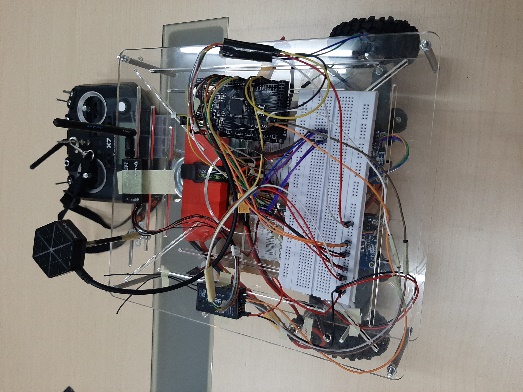
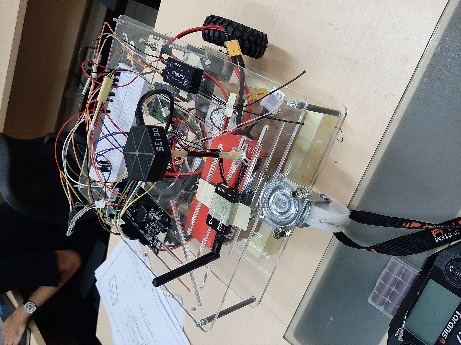
The development for MSD700 project needs information of the robot’s position at any time to enable a fully autonomous system. For outdoor use, GPS is the primary solution to obtain Filter.

1. **Update Step (measurement & fusion)**

The development for MSD700 project needs information of the robot’s position at any time to enable a fully autonomous system. For outdoor use, GPS is the primary solution to obtain Filter.

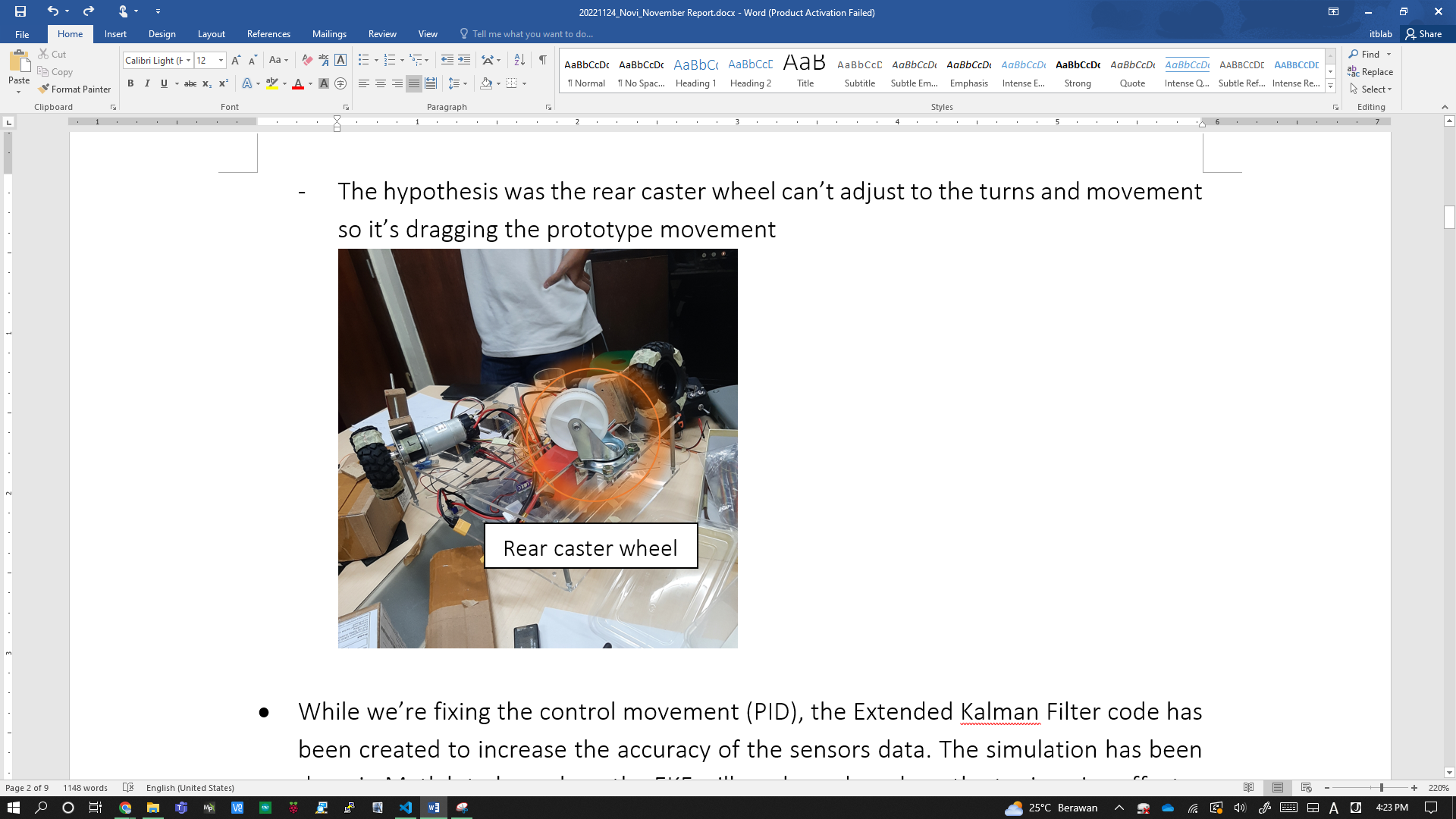
1. **Headings Callibration**

The development for MSD700 project needs information of the robot’s position at any time to enable a fully autonomous system. For outdoor use, GPS is the primary solution to obtain Filter.

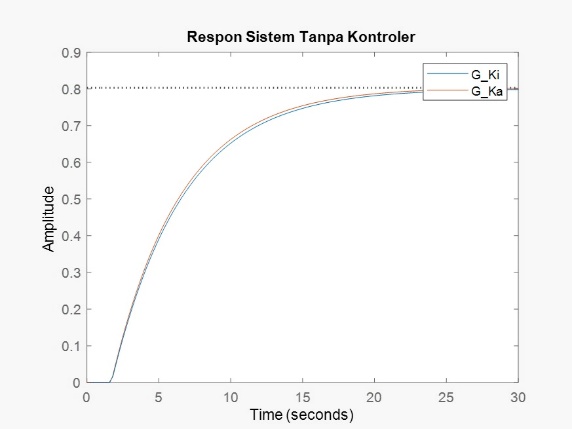
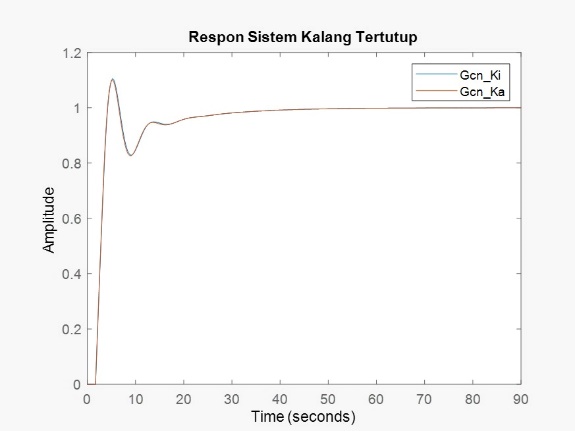


Current Prototype Looks

* 1. The remote control and receiver has been set to control the prototype. There’s 3 modes in the system: failsafe, manual, and auto mode. The prototype can move well without PID in manual mode before the PID tuning process but we’ll need PID for autonomous feature. After added the PID, the movement was quite abrupt.
     1. The simulation has been done in Matlab to find out the characteristic of the motor to tuning the PID. The tuning was done after simulation with the additional LPF added to the code
     2. The simulation for remote control has been done using the Octave application to find out the remote signal
     3. There’re 2 PID methods: using omega (angular velocity) and pulse (PWM). The pulse method gives a better result. Even though the signal from the motors in serial monitor showing the same response but the movement in reality wasn’t reliable yet. The movement between the left and right motor weren’t the same. This cause the prototype can’t go straight
     4. The hypothesis was the rear caster wheel can’t adjust to the turns and movement so it’s dragging the prototype movement

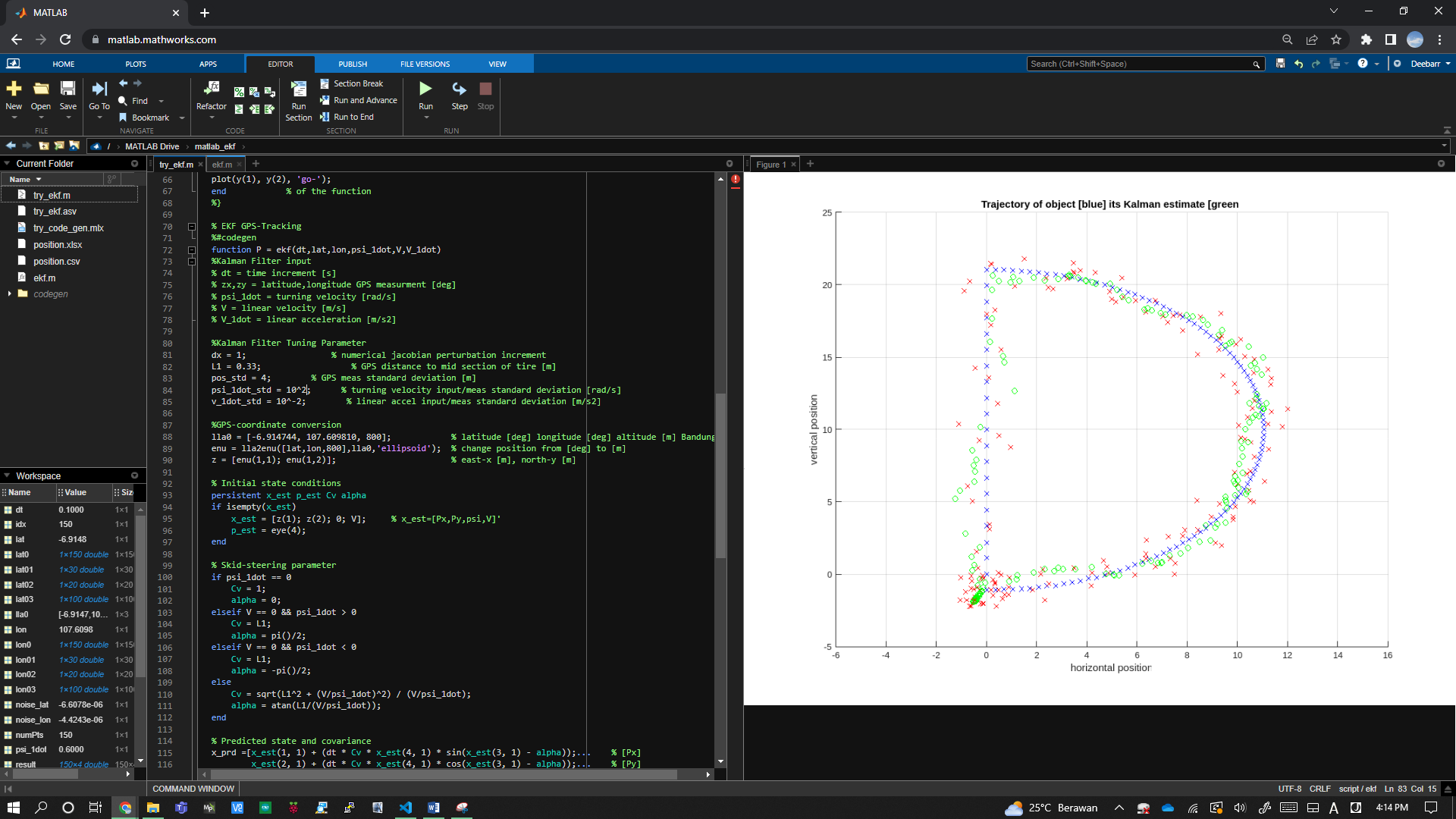


One of the PID problem causes

Motor characteristic response (without PID & with PID)

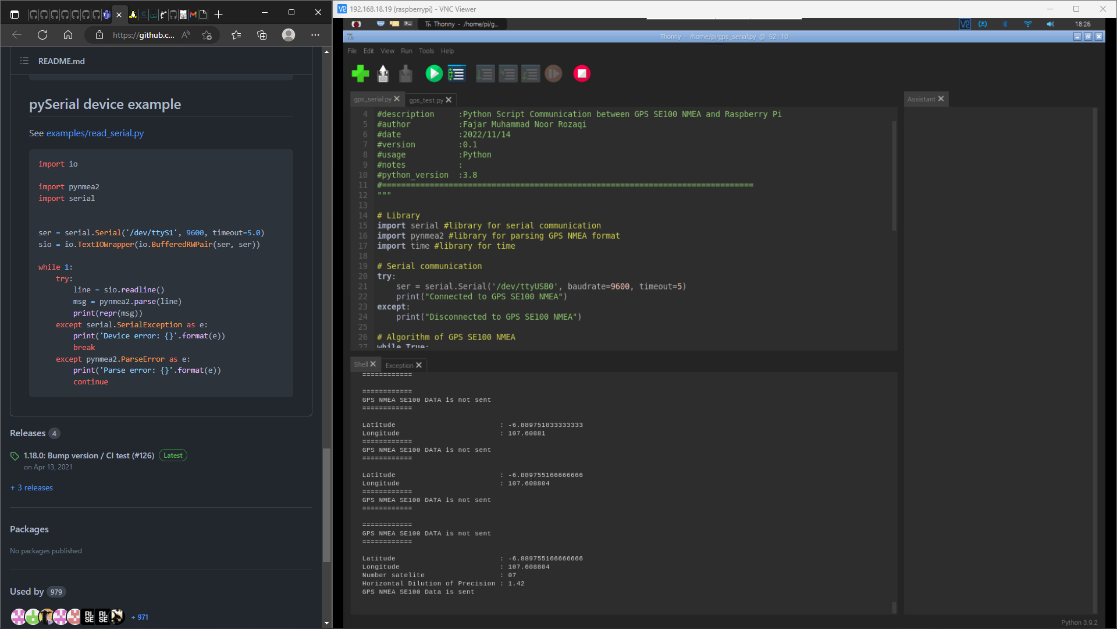
* 1. While we’re fixing the control movement (PID), the Extended Kalman Filter code has been created to increase the accuracy of the sensors data. The simulation has been done in Matlab to know how the EKF will works and see how the tuning give effects.
     1. The prediction step and update step from the EKF have been separated from the main EKF program. This step was done to enable the EKF to functioning even without the GPS data



*Extended Kalman Filter simulation with dummy data*

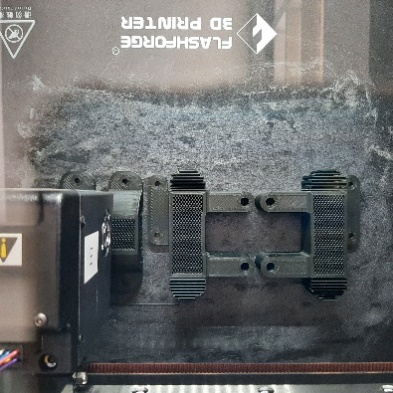
* 1. The GPS data transmission can be done wireless. The data can be sent through telemetry to the Mission Planner. There’s a plan to transmit the encoder data using the telemetry, but when testing using the dummy data, the data that being transmitted is only half of it
  2. The GPS data also can be accessed directly from the Raspberry Pi using the USB to serial TTL converter. The Ubuntu Fossa Focal 20.04 doesn’t support the usage of GPIO pins so the only way is using USB port. The latitude, longitude, number of satellites, and HDOP data from GPS – Raspberry Pi connection can be accessed from the local database and the output in code environment
     1. The data can be read well after parsing the NMEA data format
     2. The NMEA data format has 19 types data but the one that contains the data we needed is the GNGGA type. The code has been made specifically to extract this type of data





*USB to serial TTL converter & GPS parsing data code-testing*

* 1. The prototype has some changes to strengthen the structure:
     1. The motor support has been added to straighten the motor position
     2. The support was made by using 3D print
     3. The chassis support has been changed to the sturdier ones

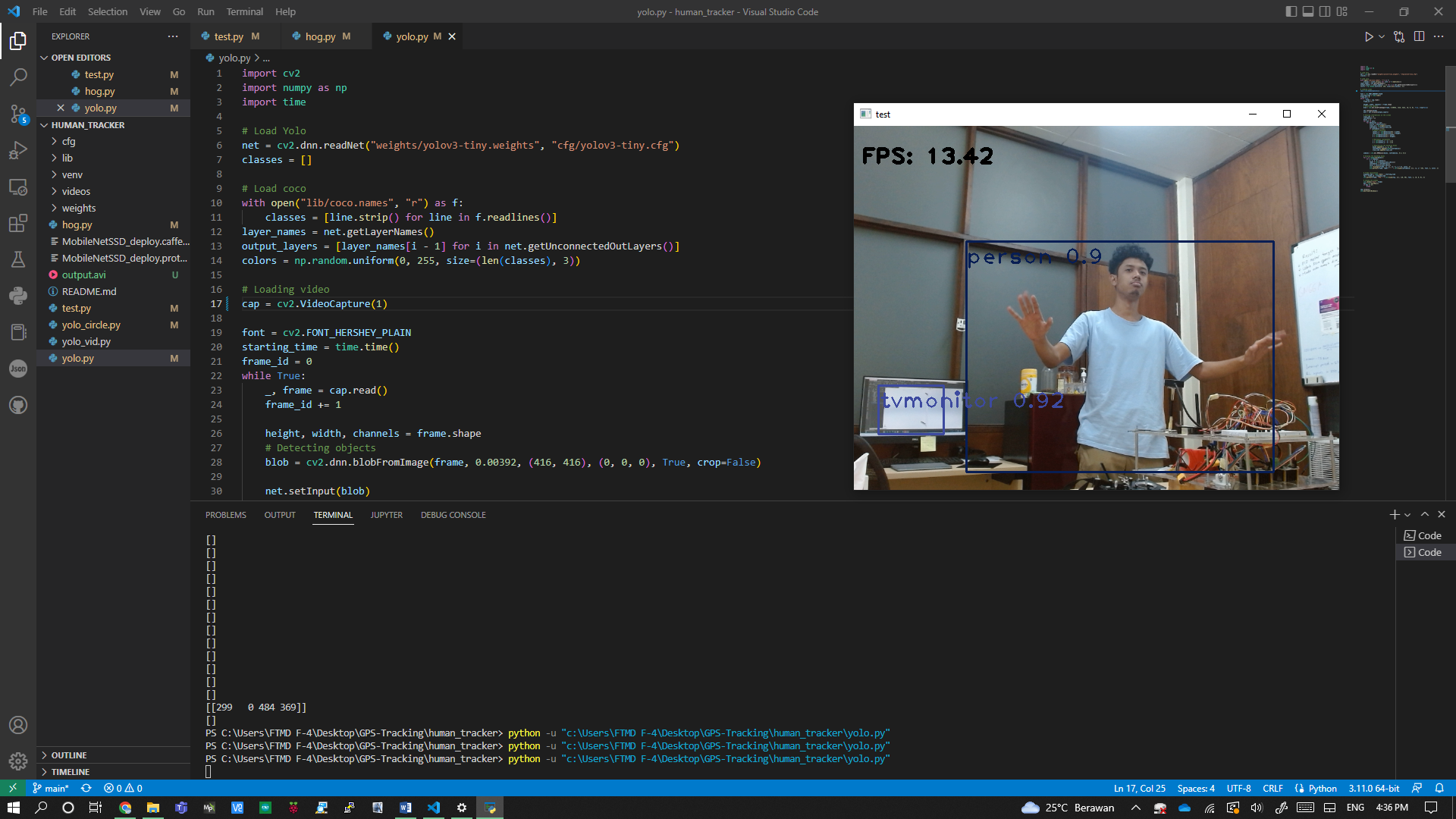


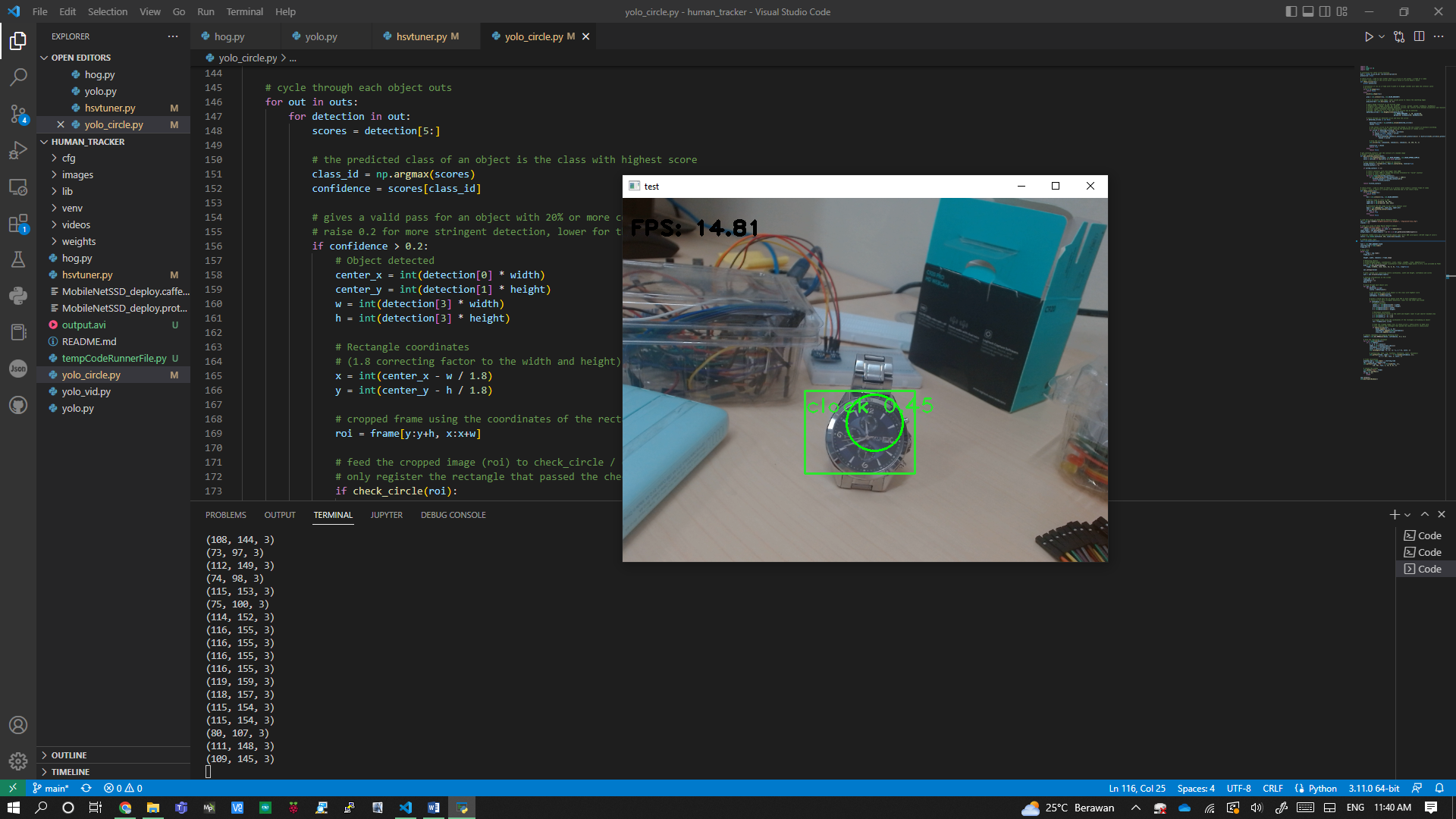
*3D print process and assembly to the prototype (motor support and chassis)*

* 1. The follower feature will be developed by using two ultrasonic sensors (maintaining distance) and camera for OpenCV (recognizing target)
     1. The ultrasonic distancing code reference has been searched from the internet but some modification will be needed

*Ultrasonic sensor US-015 & Camera Logitech C920*

* 1. There’re 2 methods for OpenCV: Histogram of Oriented Gradients (HOG) and You Only Look Once (YOLO).
     1. The additional camera has been setup to be used in VSCode
     2. The code for both method has been created and tested using human as the object. The YOLO method gives a better result with higher fps
     3. By using the YOLO method, the color and shape recognition has been created but the accuracy is still too low. It’s supposed to detect human and then red color or circle shape, but when the codes were being tested, many things were being detected as human, red color, and circle shape. This means the filter wasn’t working well
     4. The tuning has been done to increase the accuracy. The aspects that have been tuned were brightness and shape. The HSV can be tuned to recognize the color easier



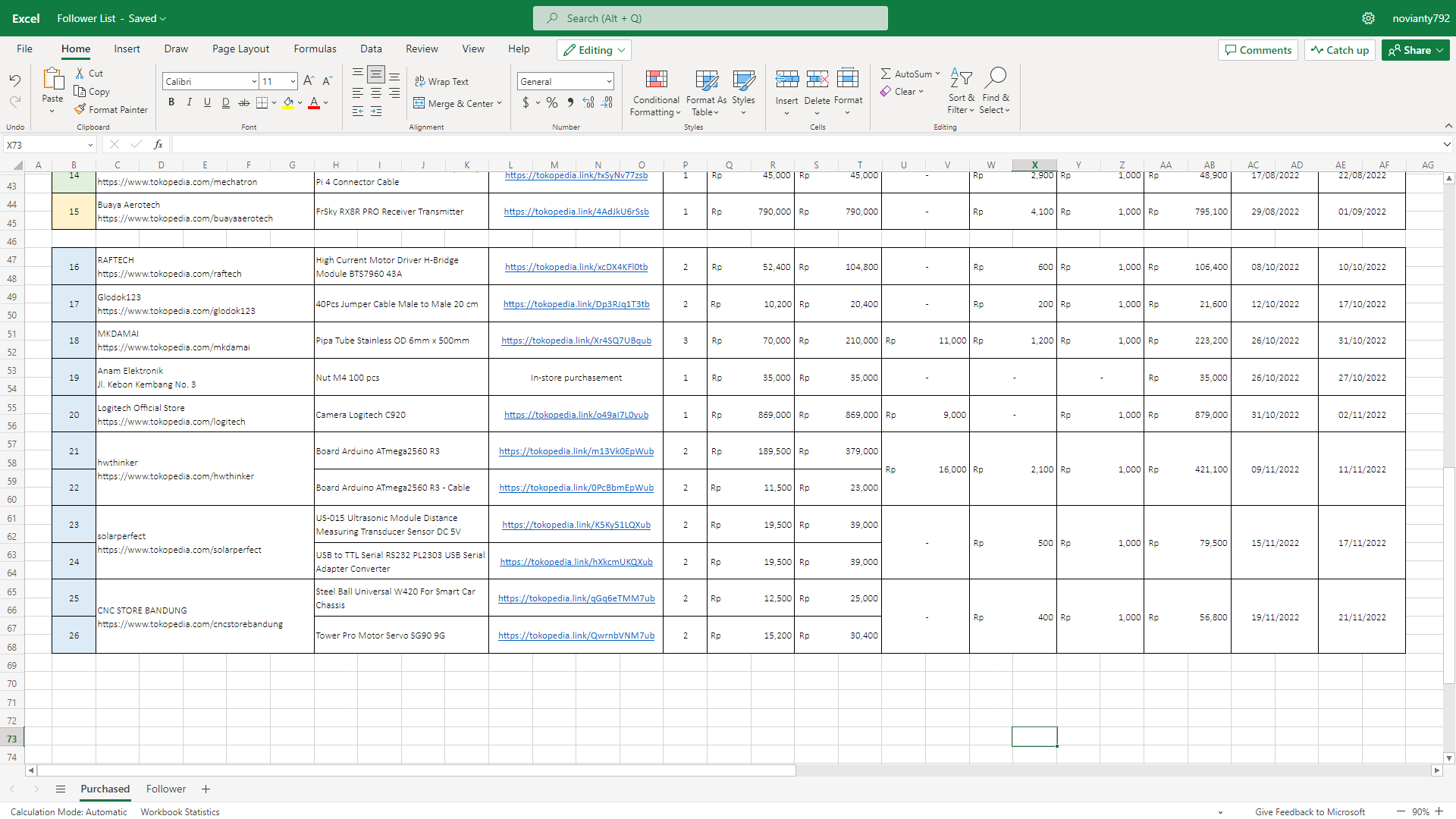


*HOG method result, YOLO method result, and YOLO detecting circle shape*

* 1. The terms of reference for the follower feature has been created. Some components have been purchased:
     1. Stainless tube with OD 6 mm x 500 mm (for chassis)
     2. C920 Pro HD Webcam (for OpenCV)
     3. Arduino ATmega2560 (replace the broken one)
     4. USB to serial TTL converter (to access GPS data directly from the Raspberry Pi)
     5. Ultrasonic sensors (to maintain the distance for follower feature)

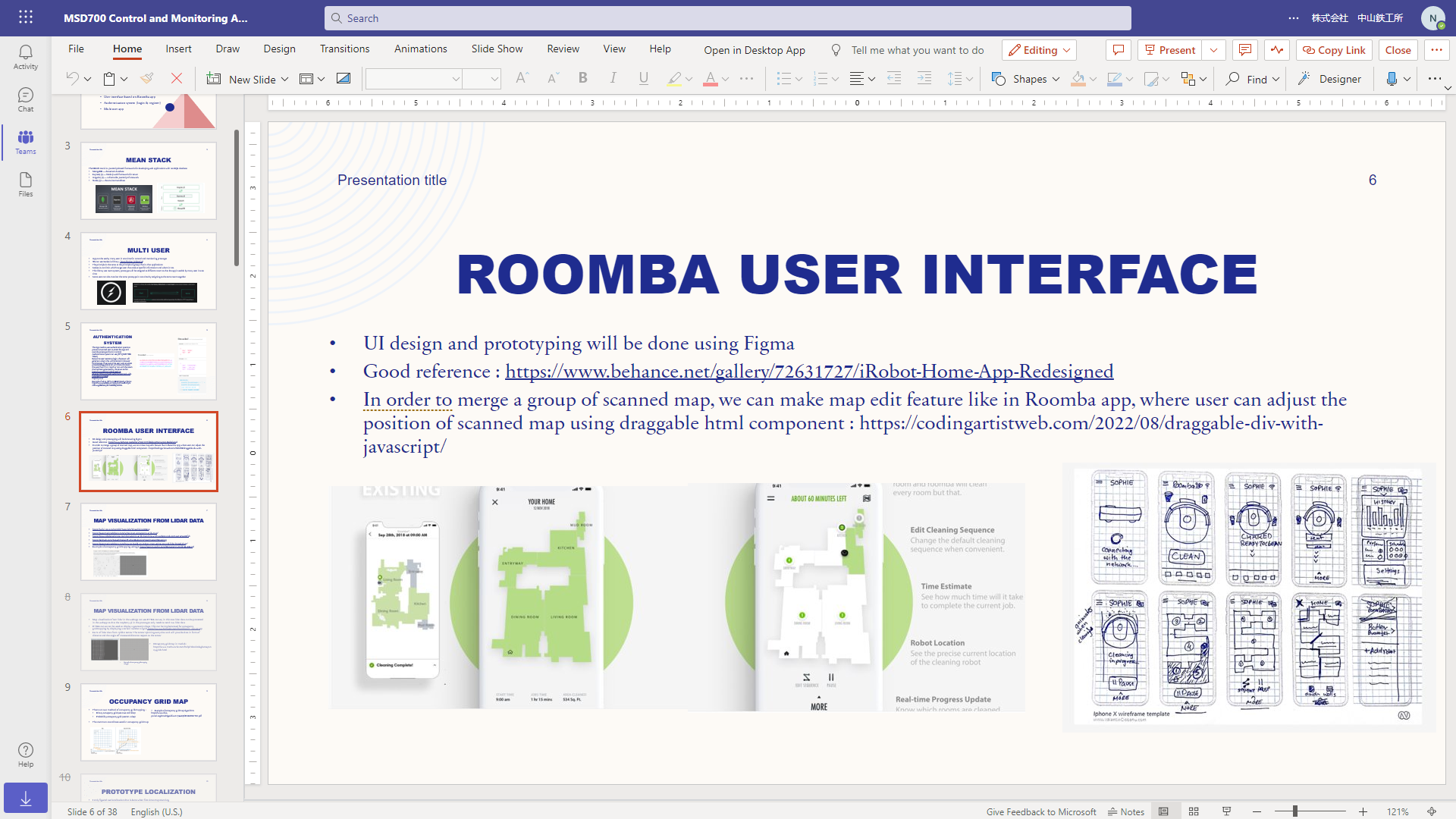
The purchased components have been listed in the procurement report. The details can be accessed here:

<https://ncjpn01.sharepoint.com/:x:/r/sites/ITBdeLabo/_layouts/15/Doc.aspx?action=edit&sourcedoc=%7B92f19a80-8ac0-4ae2-bbaf-0f83bbeaab8f%7D&wdOrigin=TEAMS-WEB.teamsSdk.openFilePreview&wdExp=TEAMS-CONTROL&web=1&cid=4d5b73c5-af34-456a-a7da-34e4b5b35ea1>

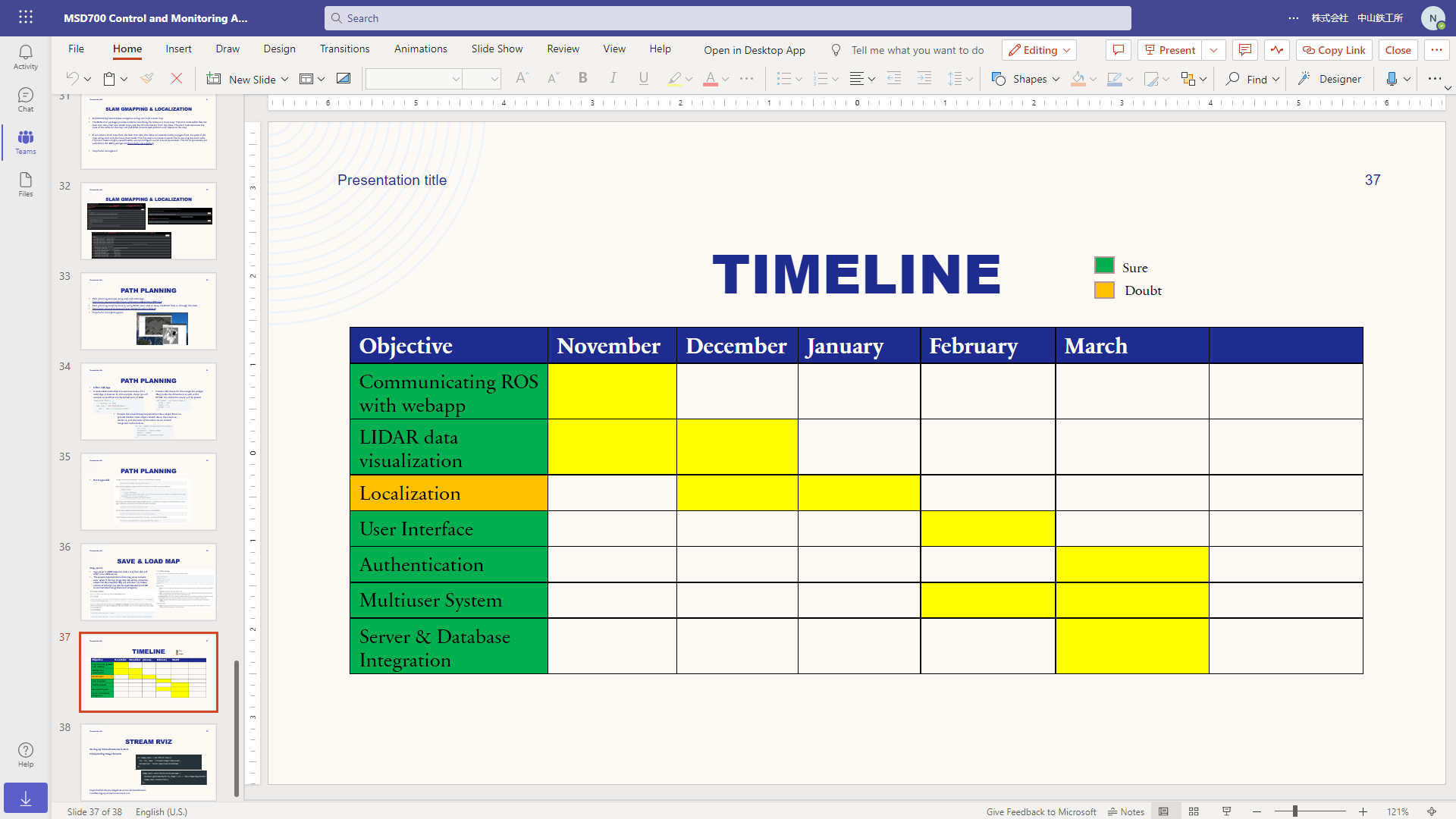


*Procurement list*

* 1. There’s a request from Ms. Veni to make the user interface for MSD700. The plan is to make the duplicate of the RViz and Roomba interface. The UI research has been submitted and the plan has been discussed. The planned features to be executed were:
     1. communicating ROS with the web-app (subscribe, service, publish, get/set param for ROS)
     2. LIDAR data visualization
     3. localization
     4. user interface
     5. multiuser system
     6. server and database integration
  2. The features above haven’t been created, but the research on the ROS has been started



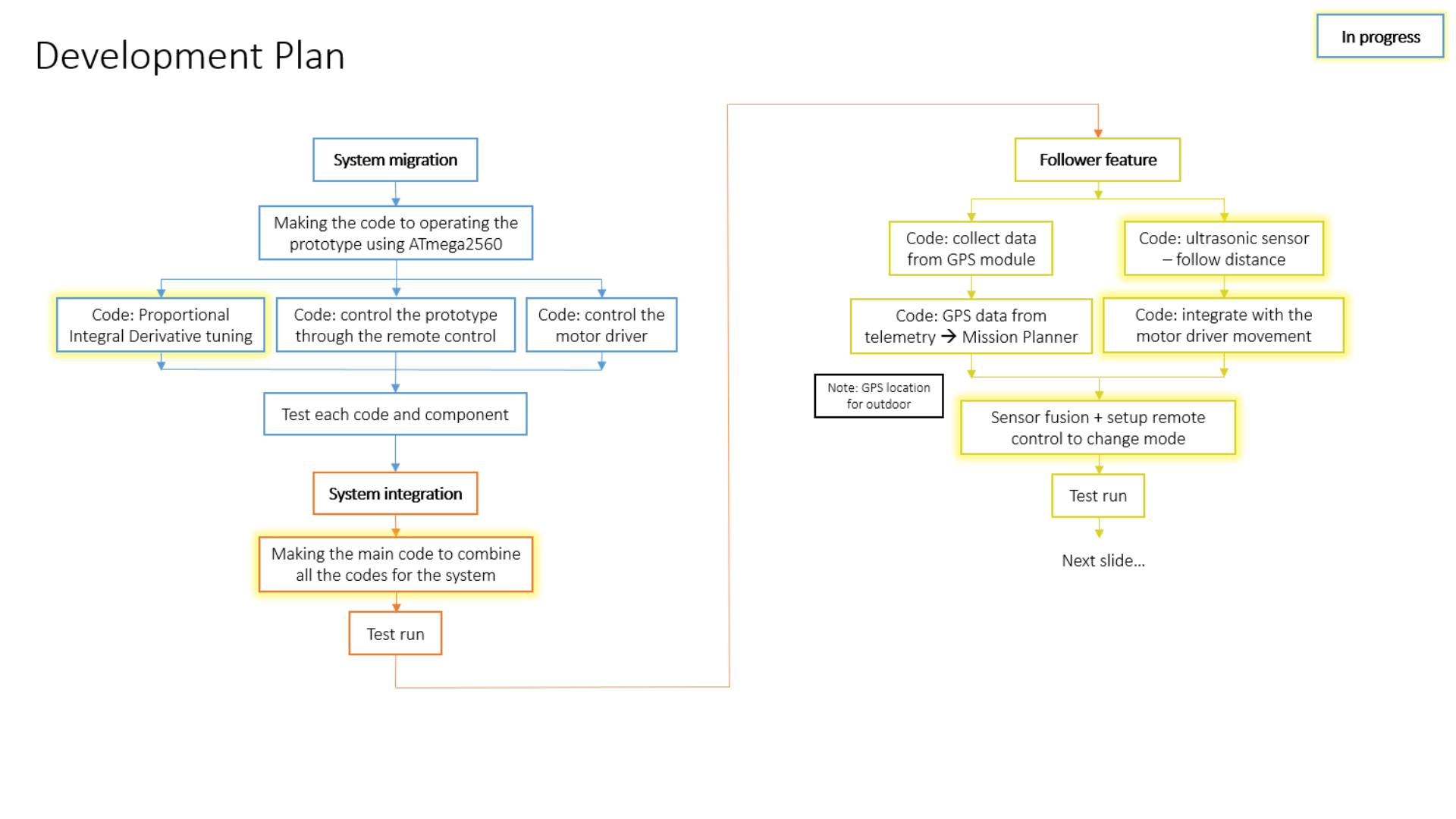
*User Interface reference*



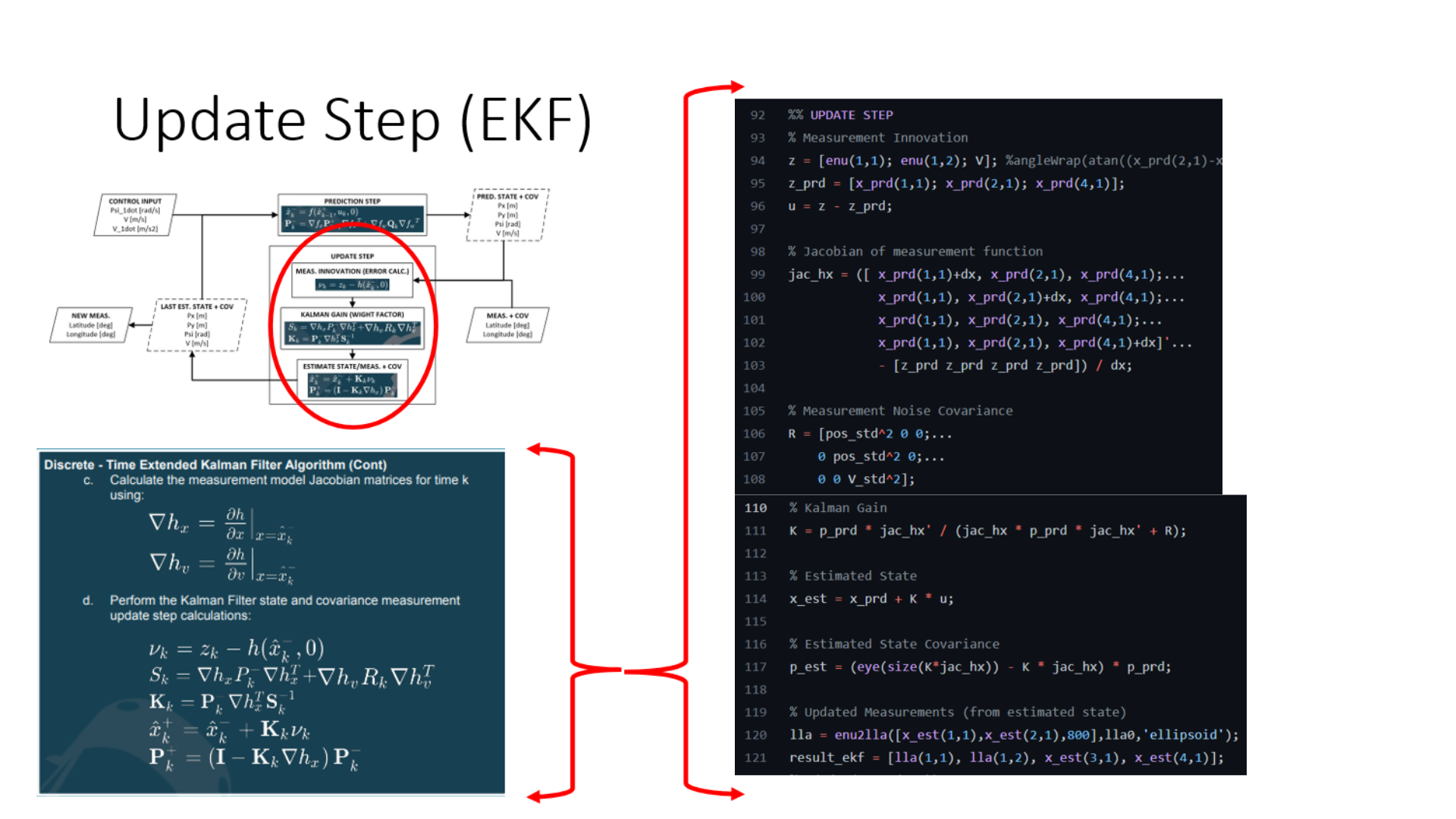
*Timeline for the UI development*

**Learning Section**

* 1. This project development introduction and sharing session meeting has been carried out to the new part-timers. The contents were the project progress and sharing knowledge



*Development progress and planning*



*Sharing knowledge session presentation (EKF)*

* 1. The new part-timer has started to learn how to code the programs and assembly the components (wiring). The things that have been learned:
     1. GPS (collecting GPS data and verify the location)
     2. Receiver – remote control (channel and connection to control the motor)
     3. Motor and PID tuning, etc.
     4. All of the code can be seen here:

<https://github.com/itbdelaboprogramming/GPS-Tracking>