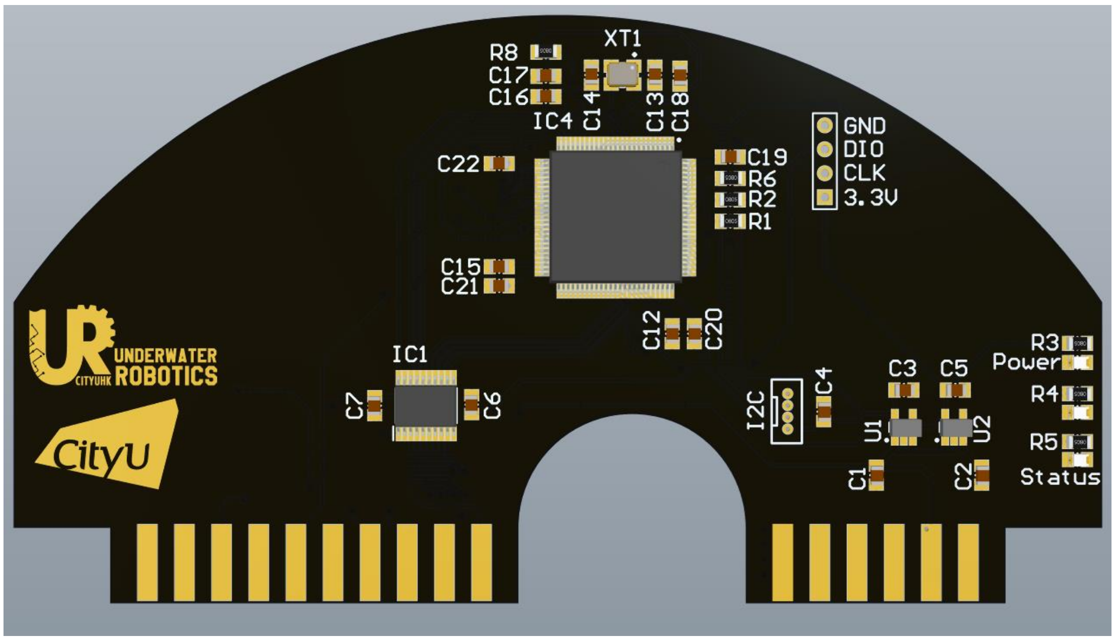
EE3070 Project Final Report

Design and Simulation of Autopilot Control System for underwater vehicle

Ngan Wai Sing SID:55685374 ROV Team



Introduction & Summary

The current problem of the underwater vehicle is that it is easily affected by the environment condition like waves, wind which causing unnecessary drifting or movement, thus the pilot or automatic program are unable to control the vehicle accurately to finish the task. To solve this problem, Autopilot Control System is necessary which uses different sensors to keep tracking the current position of the vehicle, base on the current position and desire position to adjust the vehicle's movement. In this solution, we provided an Autopilot Control System that with different flight mode and feedback of the state of the system and vehicle for the user to monitor. By using this solution, users are able to control their underwater vehicle's movement and orientation easily and accurately.

This project has been divided into serval parts, sensor module, flight controller, and user interface. Sensor modules are responsible to interact with different sensors and send the collected data to the flight controller. Flight controller is responsible to filter the collected data from sensor module, use them to estimate the current position and orientation of the vehicle, and adjust the movement of the vehicle to the desire position given by the user or automatic program. User interface is responsible to provide an interface for users to interact with the system, like controlling the desire position, getting the current position, changing the flight mode, etc.

Objective

- Get data from inertial measurement unit (IMU sensor) and depth sensor.
- Use Extended Kalman Filter (EKF) to filter the sensors' data and estimate the current position.
- Use PID controller to adjust the output PWM for controlling the vehicle.
- Make a PCB board that integrates the system.
- Provide an interface for user to interact and monitor the system.

Function Specification

- Able to estimate the vehicle’s position and orientation accurately.
- Able to adjust the vehicle to the destination accurately.
- Provided different flight modes.
    - Stabilize Mode – Allows user to control the vehicle manually, but self-levels the roll and pitch.
    - Position Mode – The vehicle maintains a consistent heading while allowing the user to control the throttle manually.
    - DepthHold Mode – Maintains a consistent depth while user is allow to control roll, pitch, and yaw.
    - Manual Mode – The controller will have no control over the vehicle, but the user is able to control the vehicle.
- Provided user interface to monitor the system, control the vehicle and tune the control parameter.

Technical Background

Pose(Orientation) estimation

In this project, IMU is used to estimate the vehicle's orientation. IMU provided angular speed, acceleration, and magnetic field, data processing has to been done to transfer the raw data of IMU to the vehicle's orientation. Moreover, due to the noise of the IMU, the calculated orientation will drift as time go by as integration is involved in the calculation, so data filtering is needed. In this project Attitude and Heading Reference System (AHRS) algorithm has been used to do data preprocessing and estimate the orientation of the vehicle. There is serval filtering algorithm for AHRS, like Complementary Filter, Mahony Filter, and Extended Kalman Filter. In this project, Extended Kalman Filter (EKF) has been selected for data preprocessing and pose estimation, as EKF's performance is more stable, even the data is very noisy, and some of the opensource commercial drone autopilot are using EKF for AHRS, so it would be more references for the development. Here will discuss the mathematics model of AHRS.

Assumption of the model:

- Angles are described in radian format.
- Coordinate are described in ENU (East, North, Up) format.
- $-\pi \leq Angular\ position\ \leq \pi$
- $Vehicle's\ Left = +X,\ Vehicle's\ Front = +Y,\ Vehicle's\ Top = +Z$

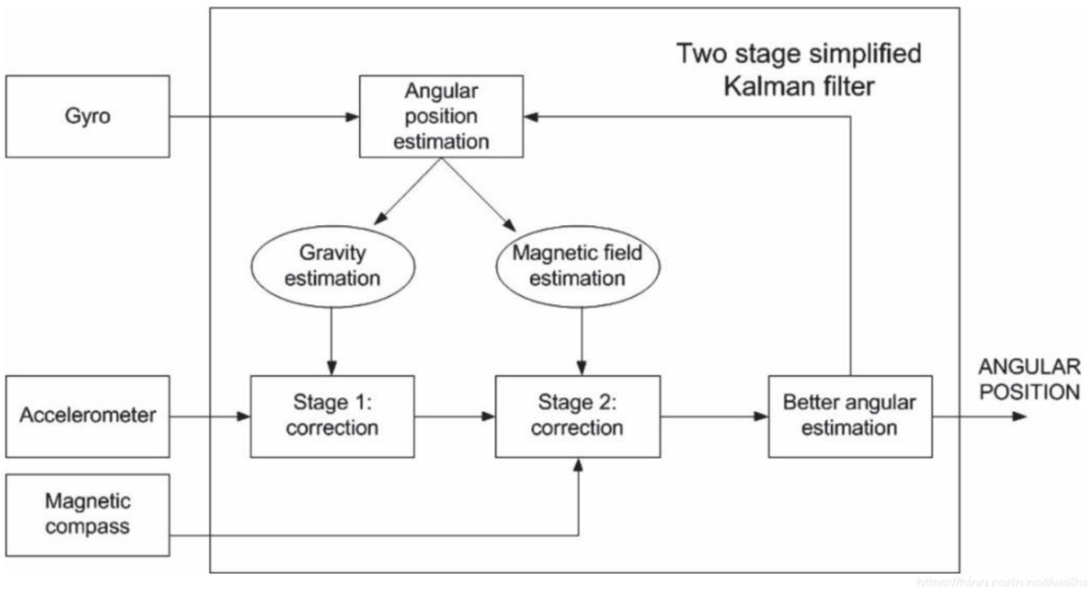

Two stage simplified Kalman filter
Gyro
Angular position estimation
Gravity estimation
Magnetic field estimation
Accelerometer
Stage 1: correction
Stage 2: correction
Better angular estimation
ANGULAR POSITION
Magnetic compass

Fig.1 Block diagram of EKF model[1]

This is the overview of AHRS, when the system got the IMU data, it will use the angular speed from the gyroscope and the last state of the system to predict the new angular position (orientation). Then the predicted angular position will estimate the vehicle experienced gravity, compare with the gravity measured by the accelerometer to do EKF correction (Stage 1). After that, as the accelerometer can only measure the changes of roll and pitch angle, so the magnetic compass is used to correct the predicted yaw angle by comparing the magnetic field estimated by predicted angular position and the measured magnetic field from the magnetic compass.

EKF is based on Kalman Filter (KF), which uses the sensors' measurements, with statistical noise and other inaccuracies, and estimates the unknown variable that tends to be more accurate than only based on a single measurement, as the measurement contains noises and different inaccuracies. The assumption of KF is that the noises of the dataset are Gaussian distributed (Linear) which in this case KF is not suitable, as the noise from the sensors is randomly distributed (Nonlinear). While EKF is the extension of KF, which solved the nonlinear problem in our case, it linearizes the model and data put in EKF and processes the filtering using the linearized model bases on KF. Here is the explanation of linearization in EKF. In this example, the input is one dimension.
First order Taylor Series are used to approximate the nonlinear function.

$$f(x_k) = f(x_{k-1}) + f'(x_{k-1})(x_k - x_{k-1})$$

The assumption is that the time between two measurements is small enough that the difference of operating point tends to 0, so the slope of those two linear functions is similar which we can use the last state linear function and the difference of the current and last operating point to approximate the current state linear function.

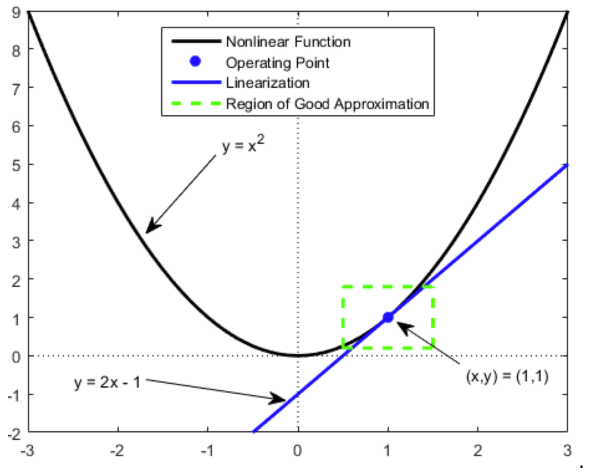


Fig2. Linearization explanation [2]

While in our model we are using four dimensions function, as the quaternion which used to describe the orientation of the vehicle (F(w,x,y,z)) have four variables, that's mean each function have to differentiate with four variables. Jacobian Matrix (JF) is used for the calculation. The equations are as below.

$$F(\mathrm{w,x,y,z}) = \begin{bmatrix} f_1(\mathrm{w,x,y,z}) \\ f_2(\mathrm{w,x,y,z}) \\ f_3(\mathrm{w,x,y,z}) \\ f_4(\mathrm{w,x,y,z}) \end{bmatrix} \quad J_F = \begin{bmatrix} \frac{\partial f_1}{\partial w} & \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial w} & \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \frac{\partial f_3}{\partial w} & \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} \\ \frac{\partial f_4}{\partial w} & \frac{\partial f_4}{\partial x} & \frac{\partial f_4}{\partial y} & \frac{\partial f_4}{\partial z} \end{bmatrix}$$

After linearized the function, the function can be put in the Kalman filter.

Intialize step

When the system is initialized, the accelerometer and magnetometer's reading will be used to calculate the initial orientation of the system($\hat{X}_0$). Here is the equation for initialization.

$$\hat{X}_0(Roll) = \operatorname{asin}\left(\frac{a_y}{\sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2}}\right)$$

$$\hat{X}_0(Pitch) = \operatorname{atan} 2(-a_x, a_z)$$

$$\begin{bmatrix} m_x^a \\ m_y^a \end{bmatrix} = \begin{bmatrix} m_x cos(Pitch) + m_z \sin(Pitch) \\ m_x \sin(Pitch)\sin(Roll) + m_y \cos(Roll) + m_z \cos(Pitch)\sin(Roll) \end{bmatrix}$$

$$\hat{X}_0(Yaw) = \operatorname{atan2}(m_y^a, m_x^a)$$

While the initialize error covariance matrix ($\hat{P}_0$), which identified the covariance between all of the error of predicted value and the true value can be set as a constant. We do not have to make $\hat{P}_0$ to be very accurate, as the system will keep tuning the error covariance matrix ($\hat{P}_k$) to a fine state.

Predict step

In this state, the current state quaternion ($\hat{X}_k^-$) and current state error covariance matrix ($\hat{P}_k^-$) have to be predicted. Here are the equations.

($\omega\ is\ the\ angular\ velocity\ from\ the\ Gyroscope$)

Ω_{nb}^n is the rotational matrix for each orientation, base on the current angular movement. T is the time elapsed between the last and current state.

$$\Omega_{nb}^n = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & -\omega_x \\ \omega_z & \omega_y & -\omega_z & 0 \end{bmatrix}$$

A_k is the discrete-time transform matrix which transforms last state quaternion($\hat{X}_{k-1}$) to current state predict quaternion ($\hat{X}_k^-$).

$$A_k = I + \frac{1}{2}\Omega_{nb}^{n}T$$

$$\hat{X}_k^- = A_k\hat{X}_{k-1}$$

The current state predicts the error covariance matrix $\hat{P}_k^-$, is predicted by A_k, $\hat{P}_{k-1}$ and Q the process noise covariance matrix.

$$\hat{P}_k^- = A_k\hat{P}_{k-1}A_k^T + Q$$

Calculate Kalman Gain

Kalman gain is used to determine the final estimated state of this iteration. As we are comparing the predicted experienced gravity and the measured gravity, so we have to transform the quaternion to gravity which is h. (q is the current state predicted quaternion)

$$h = \begin{bmatrix} 2q_1q_3 - 2q_0q_2 \\ 2q_0q_1 + 2q_2q_3 \\ {q_0}^2 - {q_1}^2 - {q_2}^2 + {q_3}^2 \end{bmatrix}$$

As the relationship between $h\&\hat{X}_k^-$ is non-linear, so we have to linearlize the h with the variable $\hat{X}_k^-$, which Jacobian matrix is used for calculation. The equation is shown below.

$$H_k = \left.\frac{\partial h}{\partial X}\right|_{\hat{X}_k^-} = \begin{bmatrix} -2q_2 & 2q_3 & -2q_0 & 2q_1 \\ 2q_1 & 2q_0 & -2q_2 & -2q_3 \\ -2q_1 & -2q_0 & 2q_3 & 2q_2 \end{bmatrix}$$

Kalman gain equation is shown below, which R is the noise of the observations (accelerometer).

$$K = \frac{\hat{P}_k^- H_k^T}{H_k\hat{P}_k^- H_k^T + R}$$

Update step

The final estimated quaternion ($\hat{X}_k$) is base on the last state and the error between obeservations(z_k) and prediction($h(\hat{X}_k^-, 0)$) times kalman gain. The final estimated error covariance matrix is base on the last state and the kalman gain.

$$\hat{X}_k = \hat{X}_k^- + K(z_k - h(\hat{X}_k^-, 0))$$

$$\hat{P}_k = (1 - KH)\hat{P}_k^-$$

After updating the current estimated state, the algorithm will go back to predict step and the value will be stored as the last state for the next iteration.

PID Controller

A PID controller is a control loop algorithm to do accurate and responsive correction base on the error between reality and setpoint. P controller is controlling the response of the system, the higher the Kp is, the larger the output which the system is more responsive. I controller is controlling the error accumulation as it integrates the error over time, so the system can still able to react when the error is small but the error accumulated is large. D controller is controlling the future trend, as it differentiates the error, if the error tends to be smaller, then the system will be less responsive.

Here is the PID controller model used in our system. First, the error between setpoint and the estimated pose from AHRS will be calculated and put into the PID controller. The output (How many should the system correct) of each controller is summed up and pass through a model which transforms the corrected pose to PWM and output to the thrusters.

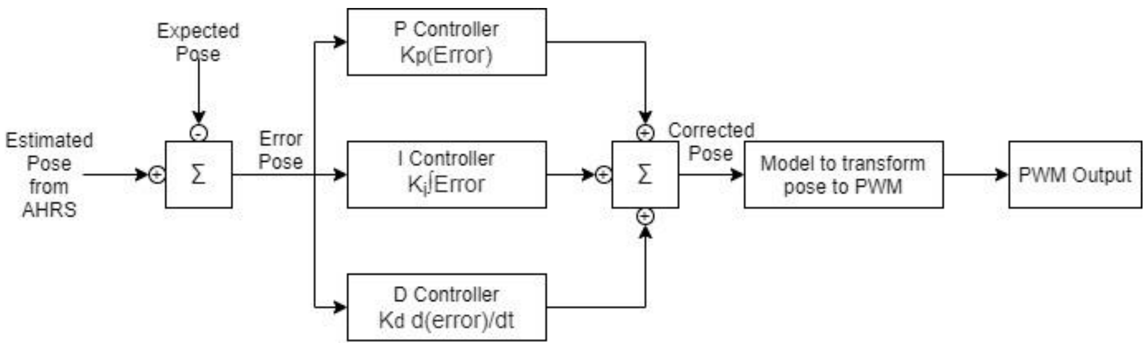


Fig3. Block diagram of PID model

Hardware and Software Architecture

This is the system block diagram, which has two main parts. Sensors module which contains the sensors (IMU, pressure (depth) sensor) needed for the system. Flight controller which responsible to process the sensor's data, communicate with the user, and control the thrusters. The red connection lines indicate that it is a hardware connection. The black connection lines indicate that it is a software or logical connection.

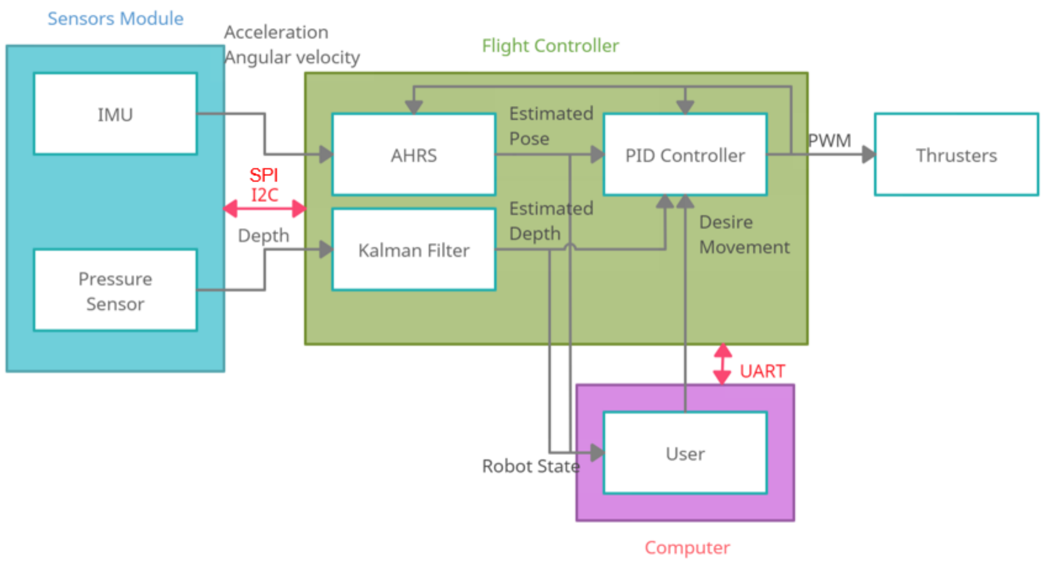


Fig4. System Block diagram

Prototyping period

The objective is to implement and test the EKF on MATLAB. MATLAB is easier to do a large number of matrix operations and less hardware control is needed, so it is easier to prototype the EKF.

Here is the hardware setup during prototyping the AHRS model. Arduino UNO and MPU9250 (9DOF IMU) have been used, with my phone's compass as a reference. Arduino UNO and MPU9250 are communicating through I2C.

Fig5,6. Testing environment

EKF mathematics models have been put in MATLAB. A fast model of orientation estimation which integrate the gyroscope reading to track the orientation has been put in to compare with the EKF model, and some graph has plot out for visualization of the results.

Flight Controller implementation

The objective is to implement the model developed on MATLAB into STM32 with C code and design and choose the hardware of this system. As the final product should work on an embedded system.

STM32H750VBT6(ARM Cortex-M7) has been chosen as the MCU in this project, as the ARM Cortex-M7 series support Floating Point Unit (FPU), which can accelerate the floating-point computation that is important in this project as many calculations with floating-point variables involved. Moreover, STM32H750VBT6 has a high master clock rate in the ARM Cortex-M series, so the system response time is higher which improves the accuracy and stability of the whole system.

SPI has been used to communicate between STM32 and MPU9250, as SPI can communicate at a higher frequency (Maximum 1MHz) comparing to I2C (Maximum 400kHz), which allows MPU9250 to update the flight controller more often, so the flight controller can update the whole system at a higher frequency that improves the accuracy and stability of the AHRS. Here is the hardware setup during testing.

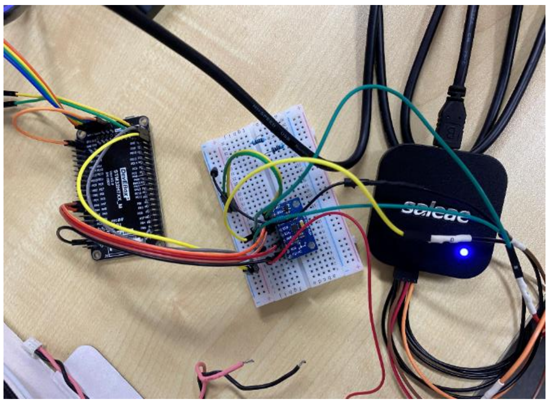


Fig7. Testing environment

During access to the register in MPU9250, the software has to control the CS Pin to low first to let the sensor know that the communication is start and ready to receive data. Then the software has to send the 8-bits register where the last 7-bits denote the address of the register and the first bit denote whether it is a read operation(0) or write operation(1) to read or write the register. Here is the SPI timing diagram.

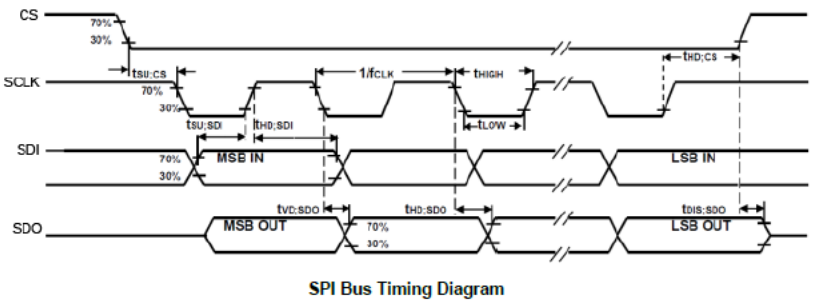


Fig8. SPI Timing diagram[3]Source datasheet

After finish reading the sensors data, the whole system can be implemented into STM32 with C code. Matrix operations are not supported on C standard library, it is hard to implement the AHRS into C directly, while CMSIS which provided a different API for ARM Cortex processors have a DSP library that able to compute complicated mathematics operations included matrix operation, and the library supports the FPU instructions, which is beneficial in this case, so we used this library for developing AHRS. Moreover, there are different tasks the MCU has to handle simultaneously (Main Control, Interact with the user, Auxiliary Control), Free RTOS v2.0 has been used to help to schedule different tasks.

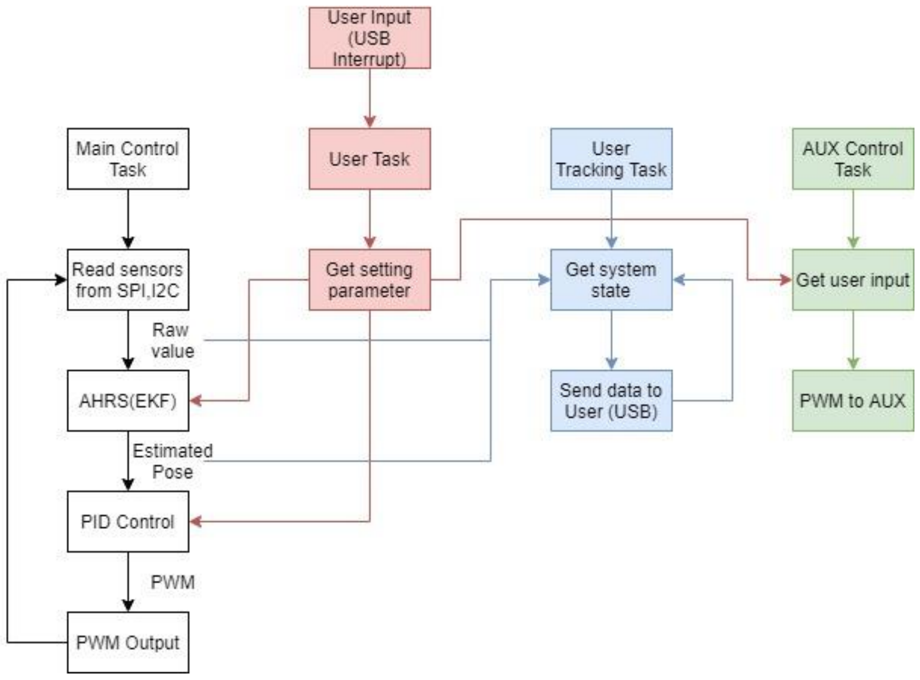


Fig9. Flowchart of RTOS tasks

Hardware (PCB) design for the flight controller

Power

The system can be power up either from the backplane system of the ROV through the gold fingers on PCB or the USB connection to the user. After having the 5V, Low-dropout (LDO) linear regulator has been used for regulating the operating voltage for the system which the output voltage much stable even for the line and load variations, changes in temperature.

IC

STM32H750VBT6 is used, as mentioned previously. Two IMU (Bosch-BMI055(6DOF) & InvenSense-MPU9250(9DOF)) have been used to prevent hardware fault of any one of the IMU and reduce noise by taking the mean of those IMU readings. Also, a CAN transceiver IC has been used to drive and detect data to and from the CAN bus.

I/O

Micro-USB or CAN-Bus have been used for the connection between user and flight controller. There is a logic shifter connected between the PWM outputs of the MCU and the PWM output to the thruster, which acts as a buffer and sharpens the PWM output to the thrusters. The output PWM is connected to the gold fingers on PCB, when the system is plugged into the backplane system of the ROV, the PWM signal will be transferred to the thrusters. Also, there are status LEDs to indicate the state of the system.

Reserved connectivity

I2C plug is reserved for the extra sensors to plug into the system like depth sensors. AUX plug is reserved for the extra auxiliary device which controller by PWM.Here is the schematic diagram and the PCB of the system.

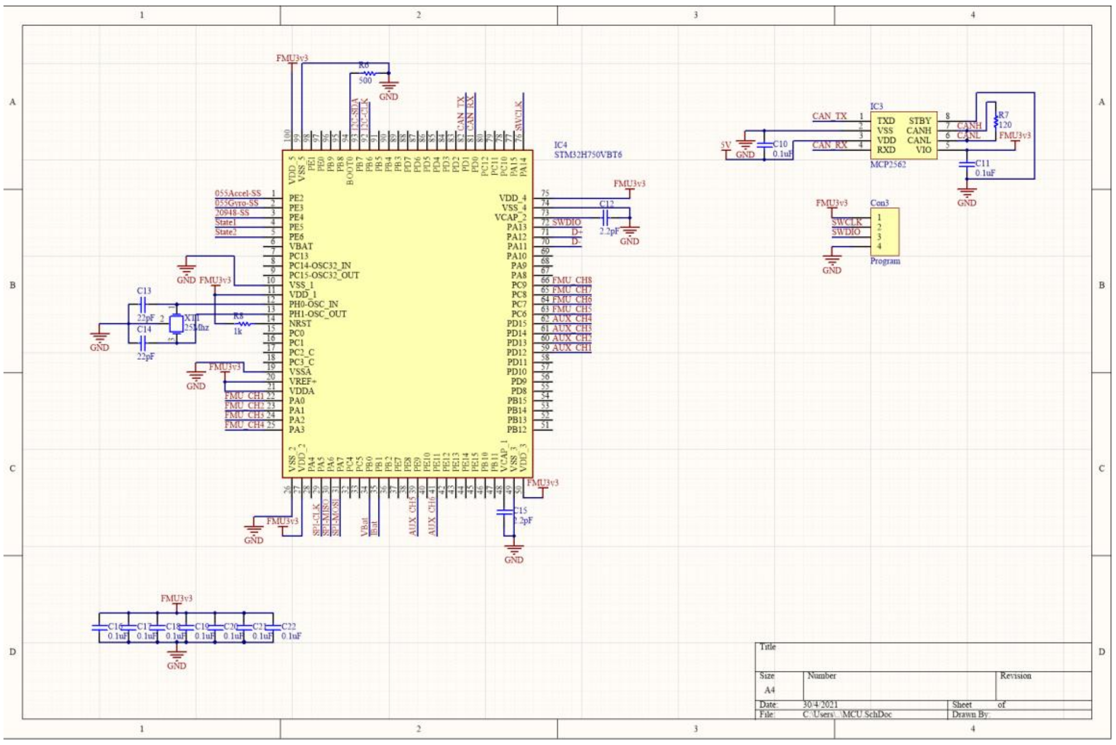


Fig10. Schematic diagram of the MCU and communcation IC

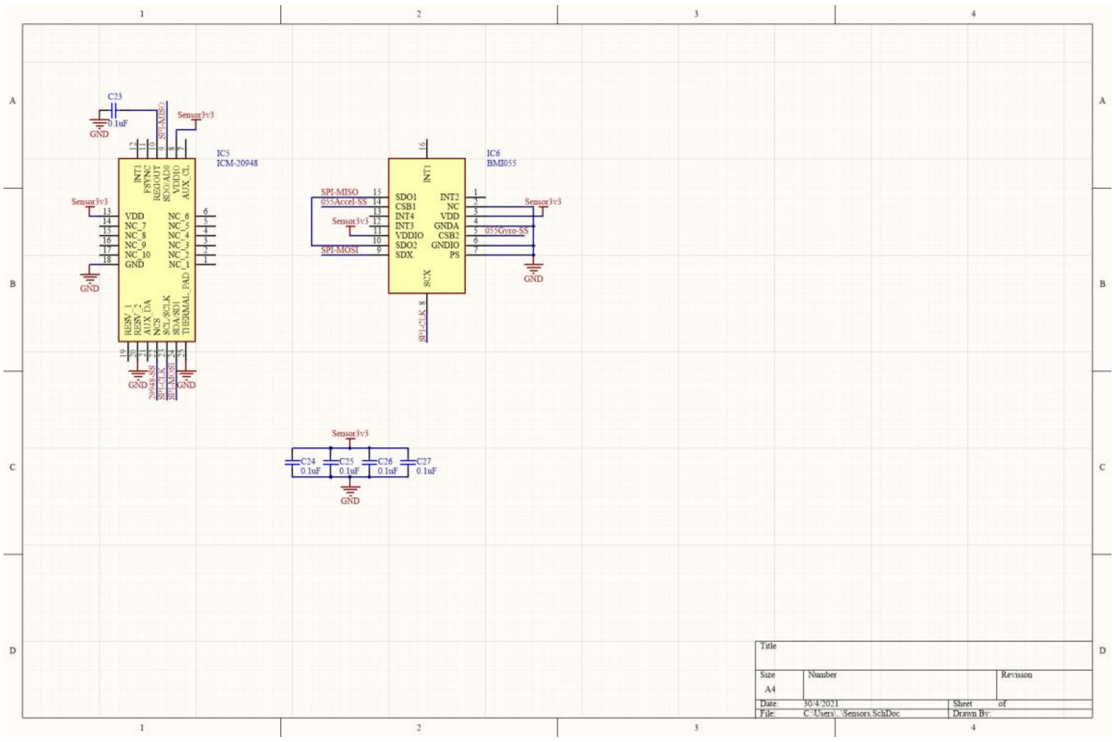


Fig11. Schematic diagram of the sensor modules

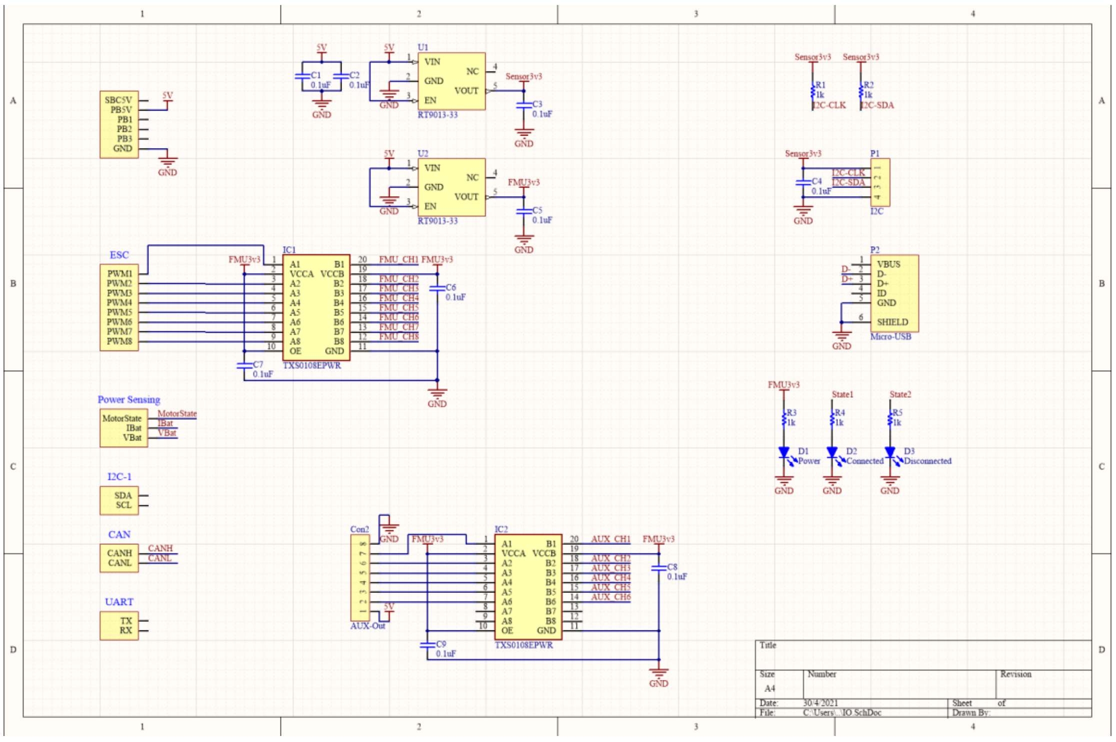


Fig12. Schematic diagram of the I/O, Power and status LED

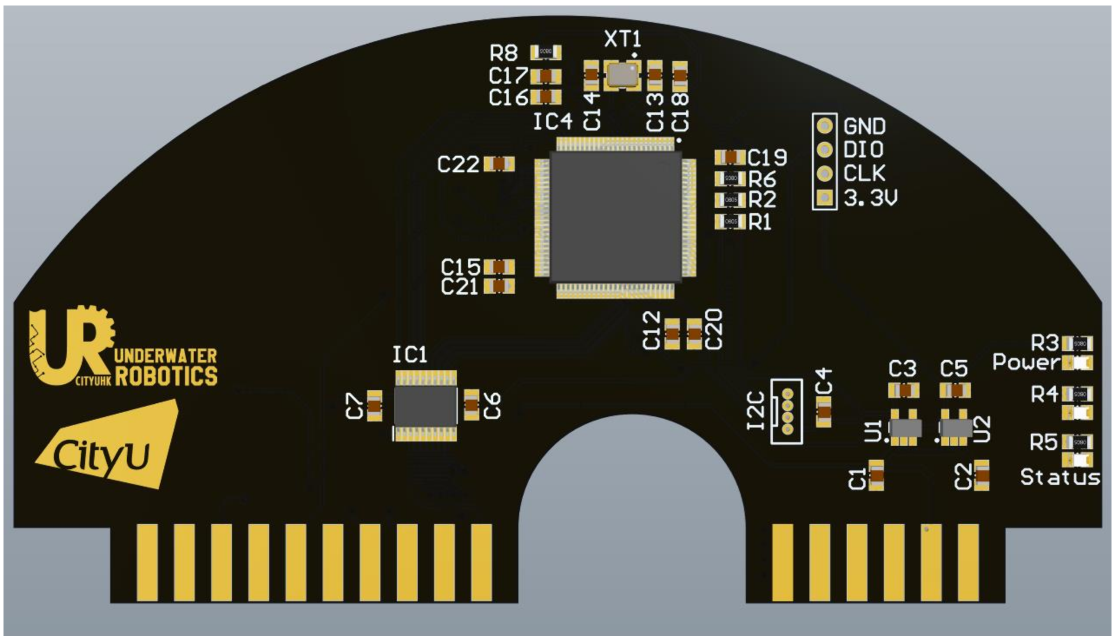


Fig13. 3D model of the front of the PCB

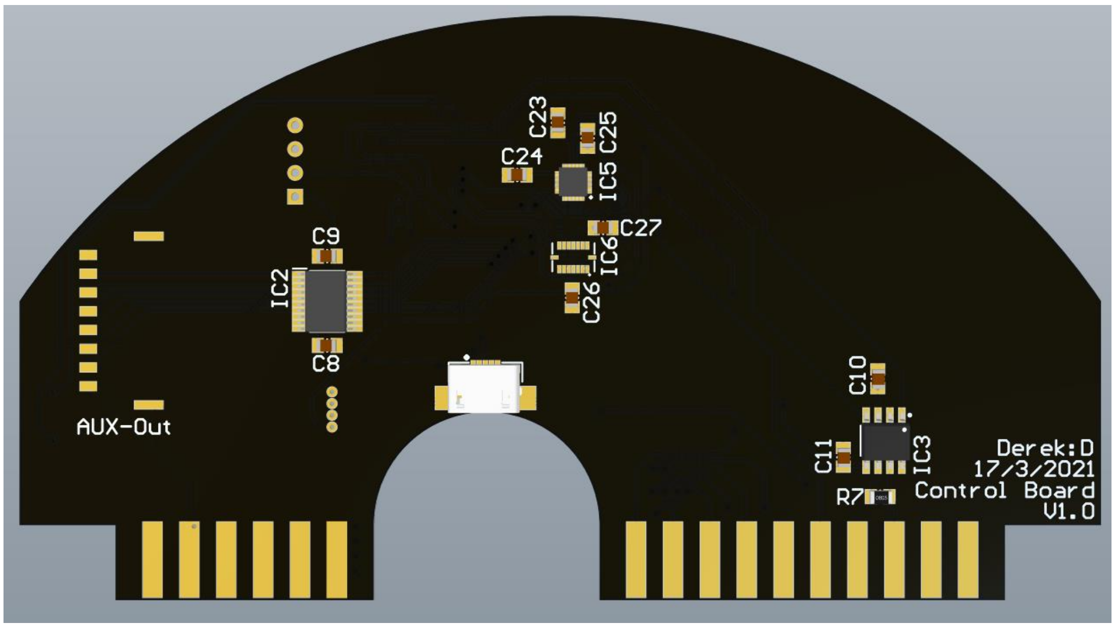


Fig14. 3D model of the back of the PCB

Simulation

Due to the current situation, it is hard to find a pool to test the hardware, so instead of testing the system with an actual ROV, simulations are used. Robot Operating System (ROS) and Gazebo have been used for simulation. Virtual RobotX (VRX) competition's Gazebo environment have been used as the simulation environment, It is a simulation of WAM-V (Unmanned Surface Vehicle) included different sensors like 6DOF IMU, GPS, Compass, Sonars, etc.

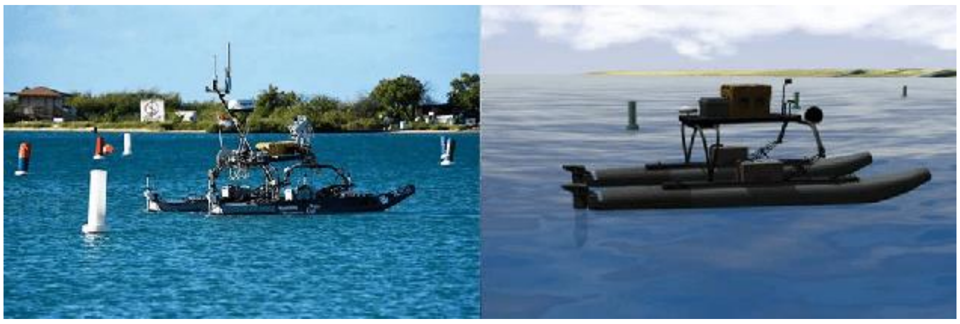


Fig15. Comparsions between actual and simulated WAM-V

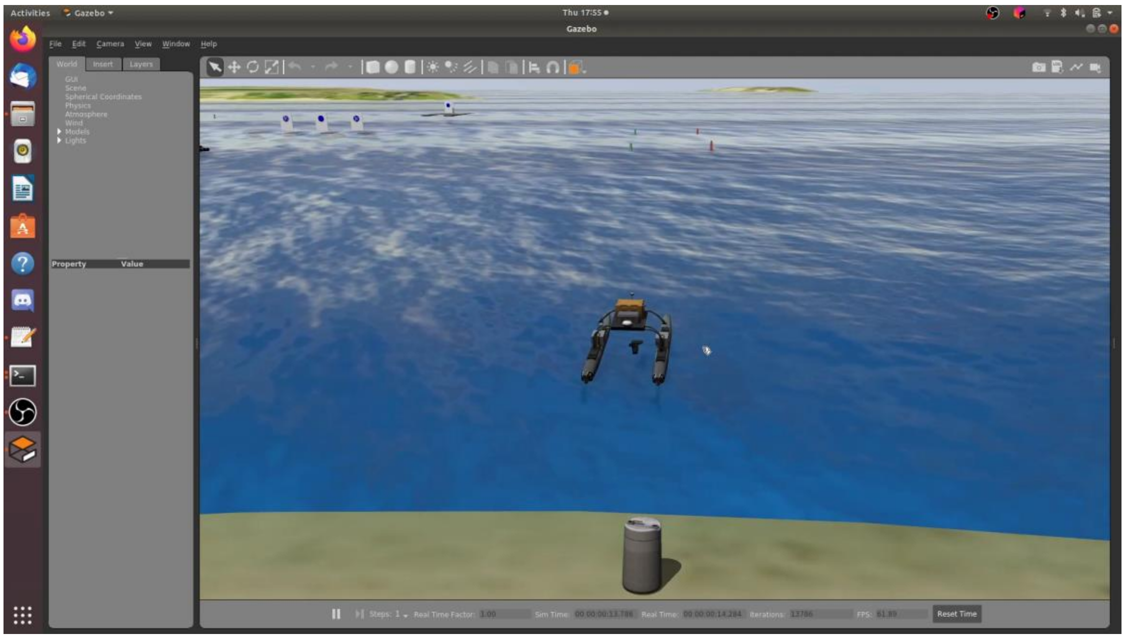


Fig16. Gazebo environment

The purpose of the simulation is to test the AHRS and the control algorithm, so only the “Main Control” task on RTOS [see “Flowchart of RTOS tasks”] are implemented in the simulation.

ROS have been used to communicate with the program and the simulated WAM-V. There are two nodes (Main Control, User input) in the system. User input node is to provide user desire movement to Main Control node through “Control Command” topic on ROS. Main control is the AHRS and PID to control the WAM-V, it gets the IMU, GPS, and Compass sensors data from the “WAMV/sensors” topic on ROS. Then process the data on AHRS and compare the result with the desired movement from the user on PID and control the thruster on the “WAMV/thruster” topic on ROS.

Project Schedule

Task	Duration	Week1	Week2	Week3	Week4	Week5	Week6	Week7	Week8	Week9	Week10	Week11	Week12	Week13
Get background informataion	1 Week	🟦												
Prototype AHRS in MATLAB	4 Week		▨	▨	▨	▨								
Reading IMU by STM32	2 Week	🟦	🟦											
Implement AHRS system into STM32	3 Week					▨	▨	▨						
Implement PID system into STM32	2 Week								🟦	🟦				
Generate PWM signal by STM32	1 Week								▨					
Design user's communication protocol	2 Week										🟦	🟦		
Design different flight mode	2 Week								▨	▨				
Gather the system with RTOS	2 Week											🟦	🟦	
Final Testing	2 Week												▨	▨
Design PCB board	2 Week					🟦	🟦							
Solder and Debug PCB	2 Week							▨	▨					

Fig17. Gantt chart of the project schedule

Testing, Results and Discussion

Testing result on MATLAB. [Prototyping Period]

Drifting test result of Roll and Pitch angle on EKF (Orange) compare with Fast model (Blue). The hardware is put on the desk without any movement.

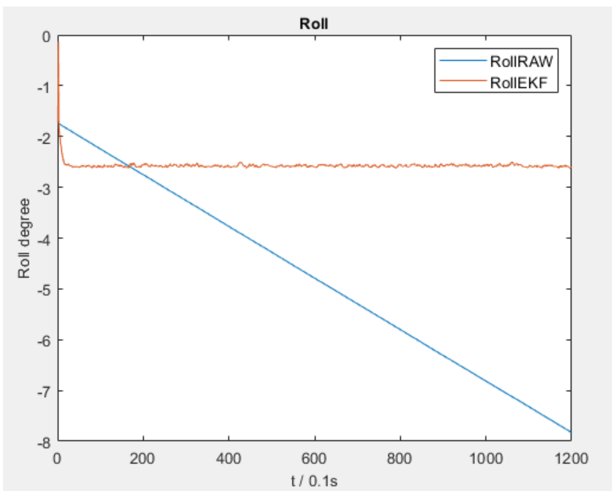


Fig18. Roll Drift Test result on MATLAB

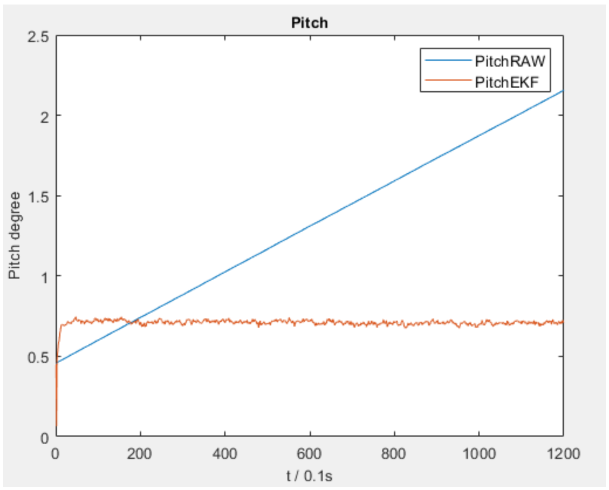


Fig19. Pitch Drift Test result on MATLAB

Figure 18,19 shows the Roll and Pitch angle calculate from the fast model keep drifting after the system is started, while in the EKF model those angles keep stabled at a point.

Drifting test result of Yaw angle on EKF (Orange (under Yellow)) compare with Fast model (Blue). The hardware is put on the desk without any movement.

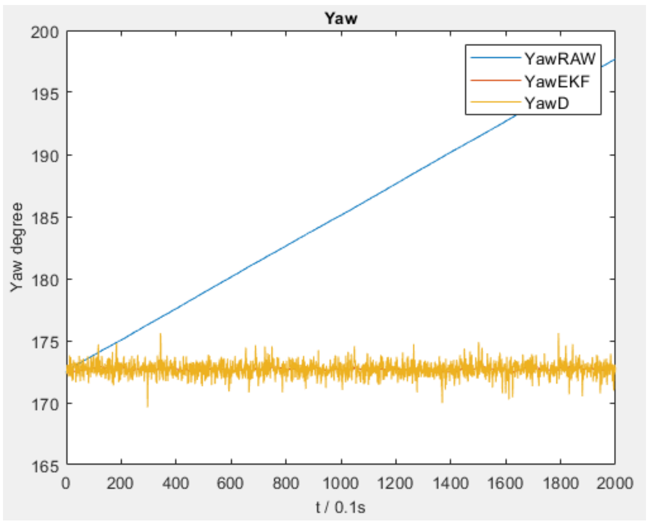


Fig20. Yaw Drift Test result on MATLAB

Figure 20 shows the Yaw angle calculate from the fast model keeps drifting after the system is started, while in the EKF model those angles keep stable at a point. Although the result is a little bit fluctuate (+- 5 degree), it is caused by the noise from the magnetometer, as the magnetometer is easily affected by noise, even the data are filtered, it still remains noisy.

The accuracy test result of Yaw angle on EKF with magnetometer correction (Orange) compares with raw EKF results without magnetometer correction(Yellow). The hardware is moving, and the change in angle a compared with the phone's compass. [Fig5,6]

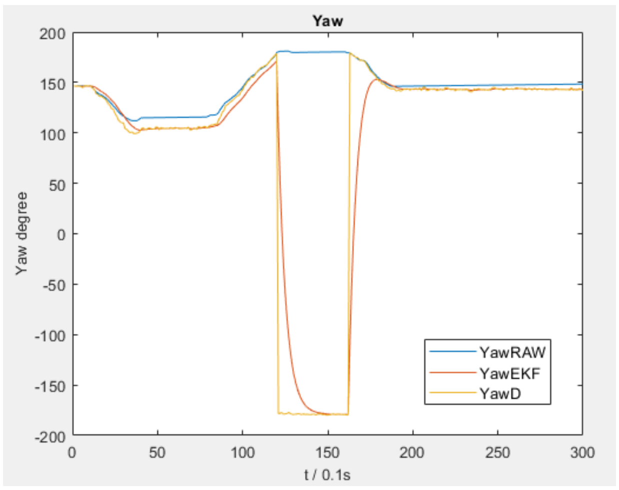


Fig21. Yaw Accracy test result 1

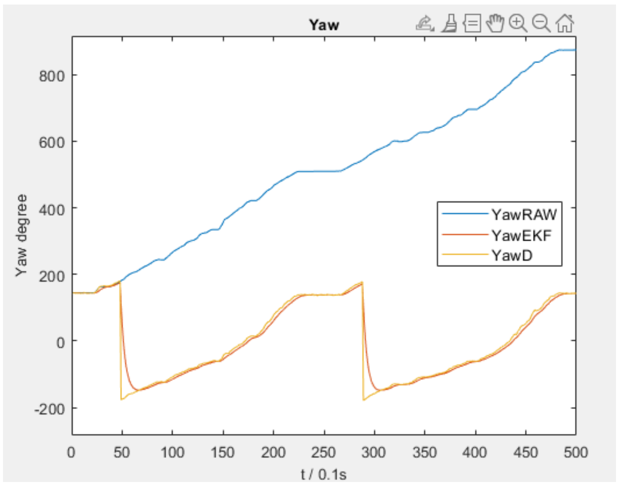


Fig22. Yaw Accracy test result 2

Figure 21 shows the result when the system is changing the Yaw angle, when there are sudden changes on the system the EKF with magnetometer correction is able to smooth the changes compare to the raw EKF results.

Figure 22 shows the result when the system is rotating is over the limitation of (π to -π). The results show that the system is able to limit the angle in the limitation.

This result proved that the EKF model is able to encounter the noise from the measurement and estimate the orientation accurately.

Testing result on Gazebo [Simulation]

This testing test the model's stabilize ability, for comparisons, a simulation without any control has been set up.
The Gazebo environment parameter can be changed, which can simulate the different environments in the real situation.

For this testing:
Wave parameter:

- gain:=0.1 (wave gain)
- period:=5 (wave period)
- direction_x:=5.0 (strength of x direction)
- direction_y:=3.0 (strength of y direction)
- angle:=0.4" (angle of the wave)

Wind parameter:

- direction:=270 (wind direction)
- mean_vel:=3 (mean of wind velocity)
- var_gain:=2 (variation gain)
- var_time:=1 (variation time)

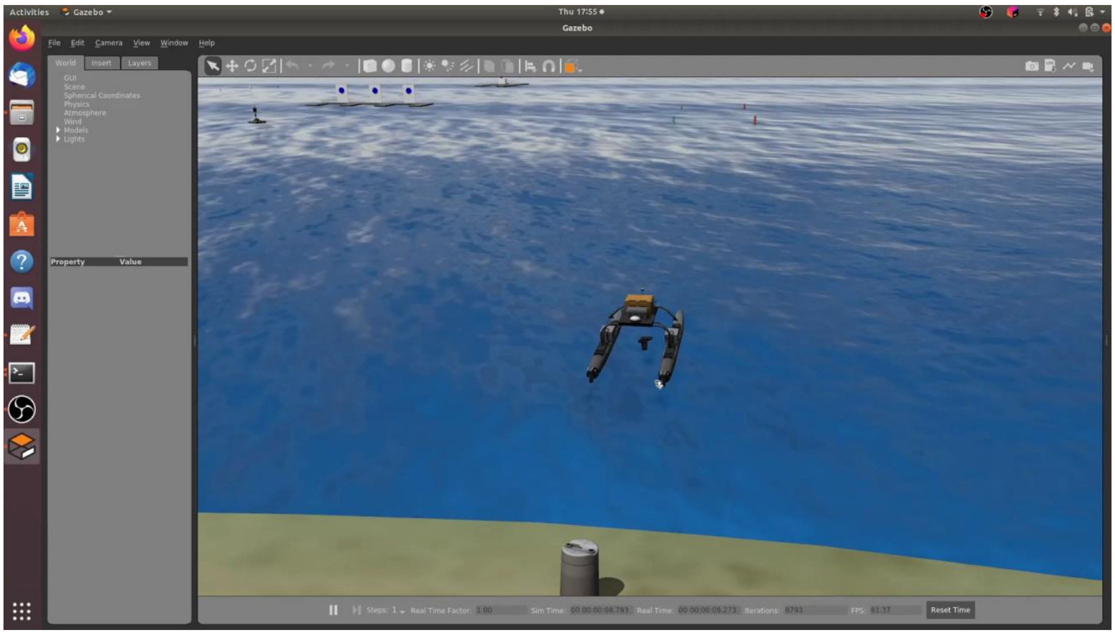


Fig23. Simulation started (Without control)

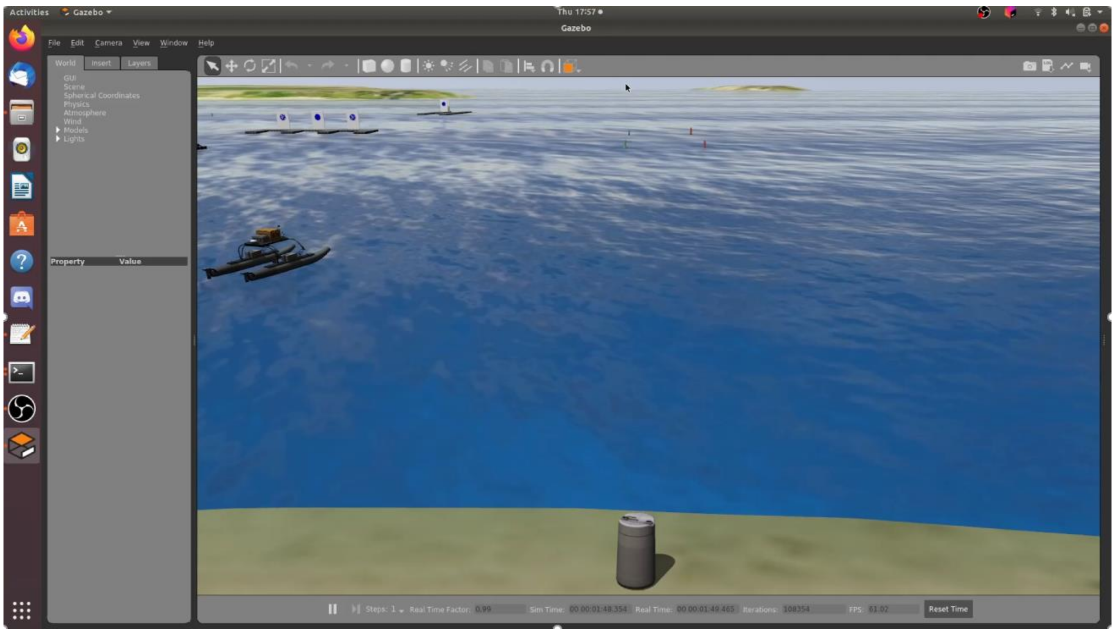
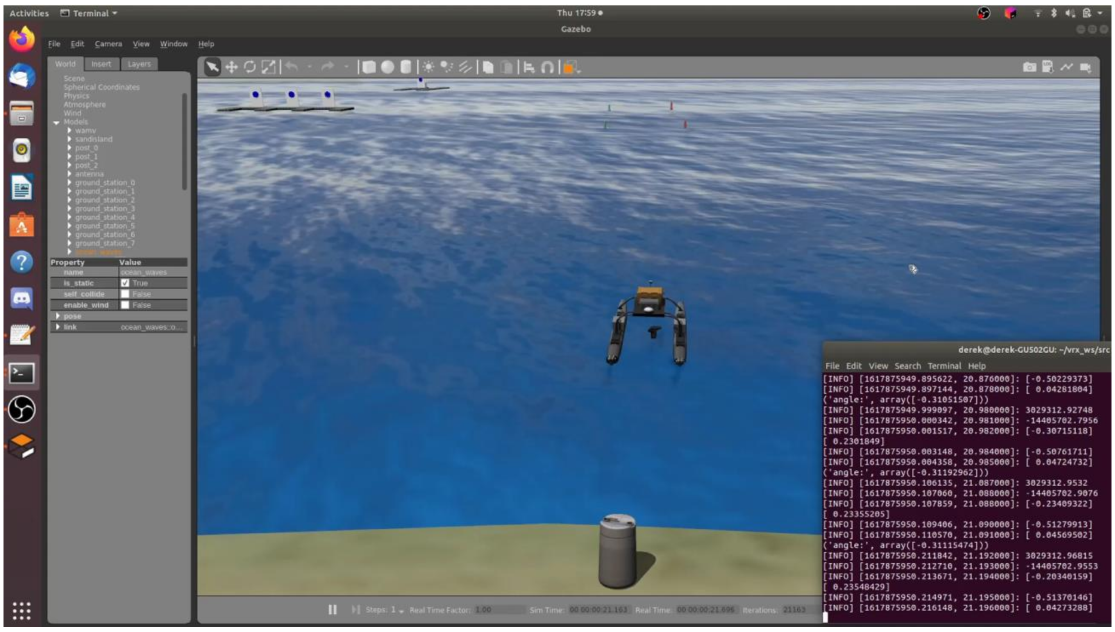


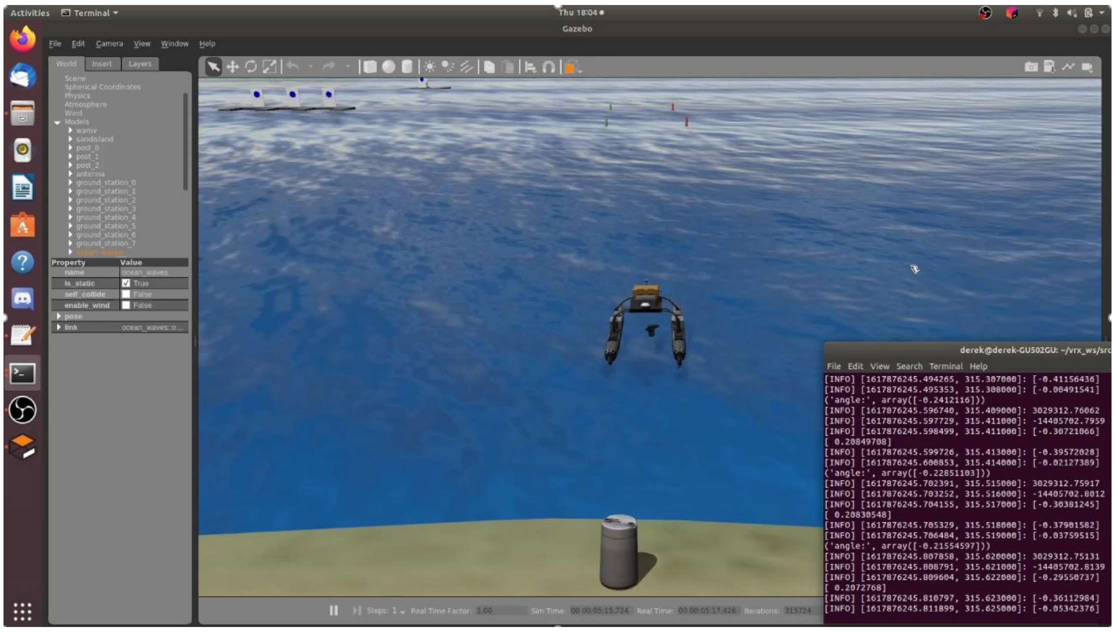
Fig24. Simulation started after 1:30 mins (Without control)

This result shows that the vehicle will lost it's original position, as the environment is strong wind and wave if there are no control on the vehicle.



```
derek@derek-GU502GU: ~/vrx_ws/src
[INFO] [1617875949.895622, 20.876000]: [-0.50229373]
[INFO] [1617875949.897144, 20.878000]: [ 0.04281804]
('angle:', array([-0.31051507]))
[INFO] [1617875949.999097, 20.980000]: 3029312.92748
[INFO] [1617875950.000342, 20.981000]: -14405702.7956
[INFO] [1617875950.001517, 20.982000]: [-0.30715118]
[ 0.2301849]
[INFO] [1617875950.003148, 20.984000]: [-0.50761711]
[INFO] [1617875950.004358, 20.985000]: [ 0.04724732]
('angle:', array([-0.31192962]))
[INFO] [1617875950.106135, 21.087000]: 3029312.9532
[INFO] [1617875950.107060, 21.088000]: -14405702.9076
[INFO] [1617875950.107859, 21.088000]: [-0.23409322]
[ 0.23355205]
[INFO] [1617875950.109406, 21.090000]: [-0.51279913]
[INFO] [1617875950.110570, 21.091000]: [ 0.04569502]
('angle:', array([-0.31115474]))
[INFO] [1617875950.211842, 21.192000]: 3029312.96815
[INFO] [1617875950.212710, 21.193000]: -14405702.9553
[INFO] [1617875950.213671, 21.194000]: [-0.20340159]
[ 0.23548429]
[INFO] [1617875950.214971, 21.195000]: [-0.51370146]
[INFO] [1617875950.216148, 21.196000]: [ 0.04273288]
```

Fig25. Simulation started (With control)



```
derek@derek-GU502GU: ~/vrx_ws/src
[INFO] [1617876245.494265, 315.307000]: [-0.41156436]
[INFO] [1617876245.495353, 315.308000]: [-0.00491541]
('angle:', array([-0.2412116]))
[INFO] [1617876245.596740, 315.409000]: 3029312.76062
[INFO] [1617876245.597729, 315.411000]: -14405702.7959
[INFO] [1617876245.598499, 315.411000]: [-0.30721066]
[ 0.20849708]
[INFO] [1617876245.599726, 315.413000]: [-0.39572028]
[INFO] [1617876245.600853, 315.414000]: [-0.02127389]
('angle:', array([-0.22851103]))
[INFO] [1617876245.702391, 315.515000]: 3029312.75917
[INFO] [1617876245.703252, 315.516000]: -14405702.8012
[INFO] [1617876245.704155, 315.517000]: [-0.30381245]
[ 0.20830548]
[INFO] [1617876245.705329, 315.518000]: [-0.37901582]
[INFO] [1617876245.706484, 315.519000]: [-0.03759515]
('angle:', array([-0.21554597]))
[INFO] [1617876245.807858, 315.620000]: 3029312.75131
[INFO] [1617876245.808791, 315.621000]: -14405702.8139
[INFO] [1617876245.809604, 315.622000]: [-0.29550737]
[ 0.2072768]
[INFO] [1617876245.810797, 315.623000]: [-0.36112984]
[INFO] [1617876245.811899, 315.625000]: [-0.05342376]
```

Fig26. Simulation started after 4:30mins (With control)

This result shows that the model are able to stabilize the vehicle even in a strong wind and wave situation.

Conclusion and Suggested Improvement

The final result is that the from the simulator shows that the flight controller algorithm are able to estimate the orientation of the vehicle correctly and control the vehicle to stabilize which meted the target that controlling the vehicle the user desire position in a noisy environment.

There are few problems faced on the testing.

First, the magnetometer's noise is large that even the EKF model cannot filter the noise as expected, as the magnetometer will be easily affected by the environment like the magnetic field generated when the electronics operating and the magnetic field generated from thrusters. It is hard to reduce the noise generated from thrusters, so the target is to reduce the noise generated from the electronics. As the IMU model (MPU9250) used in this project integrated the accelerometer, gyroscope, and magnetometer in one package, which may cause the magnetometer to operate in a noisy environment, so the results may be affected by the design of the IC package. To solve this problem, a separate magnetometer (compass) IC can be used for the next-generation design.

Second, the simulation environment can only test the performance and the problem faced on the software model while the hardware cannot really be tested in Gazebo. To solve this problem, a hardware test should be carried out. The hardware test's objective could be testing the reliability of the hardware system and the driver program, and the software simulation can make sure the model is logically correct.

References

S. Sabatelli, M. Galgani, L. Fanucci and A. Rocchi, "A Double-Stage Kalman Filter for Orientation Tracking With an Integrated Processor in 9-D IMU," in *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 3, pp. 590-598, March 2013, doi: 10.1109/TIM.2012.2218692. [1]

"Linearize Nonlinear Model," Linearize Nonlinear Models - MATLAB &amp; Simulink - MathWorks. [Online]. Available: https://ww2.mathworks.cn/help/slcontrol/ug/linearizing-nonlinear-models.html. [Accessed: 30-Apr-2021]. [2]

InvenSense "MPU-9250 Product Specification Revision 1.1" MPU-9250 June 2016

[3]