

Project 2: Autonomous Vehicle (Putting It All Together) Group 10 & Group 44

COMPENG 4DS4: Embedded Systems

Professor: Mohamed Hassan

Due: Wednesday April 3rd, 2024

Name: Cameron Beneteau

Email: beneteac@mcmaster.ca

Student Number: 400274092

Name: Shaqif Ahmed

Email: ahmes80@mcmaster.ca

Student Number: 400263303

Name: Tyler Moss

Email: mosst2@mcmaster.ca

Student Number: 400274039

Name: Noah Betik

Email: betikn@mcmaster.ca

Student Number: 400246583

Declaration of Contributions

Tasks	Contributor(s)
Experiment 1: Build and Program PX4	Cameron Beneteau, Shaqif Ahmed, Tyler Moss, and Noah Betik
Experiment 2: Write Applications on NuttX with PX4	
Setup A: Import PX4 to MCUXpresso	
Setup B: Hello World Application	
Experiment 3: uORB Messaging	
Experiment Setup A: Interact with uORB	
Experiment Setup B: Subscribe to a Message	
Project 2 – Step 0: RC Channels Reading Application	
Experiment Setup C: Publish a Message	
Project 2 – Part 1 (Revisiting Project 1 using PX4)	
Experiment 4: MAVLink	
Experiment Setup A: Setting up the FMU side and Reading Messages from FMU	
Experiment Setup B: Read Custom Messages from FMU	
Experiment Setup C: Send Messages to FMU	
Experiment 5: Ultrasonic Sensor	
Experiment 6: Camera	
Project 2 – Part 2	

GitHub Repository

<https://github.com/COMPENG-4DS4-Winter2024/project2-group-10>

<https://github.com/COMPENG-4DS4-Winter2024/project2-group-44>

Overview

The focus of this lab was building and programming the PX4 software on the FMUK66. Initial experiments introduced application development within the NuttX operating system, utilizing the PX4 framework for creating and deploying applications. A significant aspect of the lab involved exploring the uORB messaging system for inter-process communication and utilizing the MAVLink protocol for communication with external devices and sensors. Integration of an ultrasonic sensor and a camera with a Raspberry Pi was done for obstacle detection and navigation tasks. Later projects applied these topics towards controlling the vehicle's movements, leveraging sensor data for real-time decision-making and obstacle avoidance.

Experiment 1: Build and Program PX4

This experiment was to compile and install the PX4 flight software onto the FMUK66. Using a virtual machine equipped with Ubuntu, the PX4 was built for the FMUK66-v3 target and programmed onto the FMU via JLink and USB connections. Successful deployment was verified through the FMU's LED responses.

Experiment 2: Write Applications on NuttX with PX4

This experiment was to develop applications on the NuttX operating system using the PX4 framework. The experiment was divided into two setups: importing the PX4 project into the MCUXpresso IDE for development and creating a "Hello World" application as an introduction to application development within the PX4 environment.

Setup A: Import PX4 to MCUXpresso

In this setup, the PX4 project was imported into the MCUXpresso IDE, configuring a development environment optimized for editing, building, and debugging PX4 applications.

Setup B: Hello World Application

This task involved developing a "Hello World" application as an initiation into the PX4 application development process on NuttX. It demonstrated the basics of creating an application, highlighting the workflow of writing, building, and deploying code to the FMU.

```
int hello_world_main(int argc, char *argv[])
{
    for (int i = 0; i < 5; i++)
    {
        PX4_INFO("Hello World %d", i);
        px4_sleep(1);
    }
    return 0;
}
```

```
nsh> hello_world
INFO [hello_world] Hello World 0
INFO [hello_world] Hello World 1
INFO [hello_world] Hello World 2
INFO [hello_world] Hello World 3
INFO [hello_world] Hello World 4
nsh> 
```

Experiment 3: uORB Messaging

Experiment 3 explored the uORB messaging system, essential for inter-process communication across the vehicle's systems. The focus was on understanding how components within PX4 subscribe to and publish messages, facilitating coordinated operations among the vehicle's sensors and actuators.

Experiment Setup A: Interact with uORB

The setup provided experience with the uORB messaging system, showing how to interact with, publish, and subscribe to uORB messages within PX4. Below are some of the commands that were tested in the terminal.

```
nsh> listener commander_state

TOPIC: commander_state
commander_state
  timestamp: 486846701 (0.438555 seconds ago)
  main_state_changes: 0
  main_state: 0

nsh> █
```

```
nsh> listener vehicle_air_data

TOPIC: vehicle_air_data
vehicle_air_data
  timestamp: 571084200 (0.059045 seconds ago)
  timestamp_sample: 571083132 (1068 us before timestamp)
  baro_device_id: 5333009 (Type: 0x51, I2C:2 (0x60))
  baro_alt_meter: 99.1927
  baro_temp_celcius: 37.8750
  baro_pressure_pa: 100139.2500
  rho: 1.1898
  calibration_count: 0

nsh> █
```

```

nsh> listener battery_status

TOPIC: battery_status
battery_status
timestamp: 517091475 (0.009299 seconds ago)
voltage_v: 7.3144
voltage_filtered_v: 7.3787
current_a: 0.0000
current_filtered_a: 0.0000
current_average_a: 15.0000
discharged_mah: 0.0000
remaining: 1.0000
scale: 1.0000
time_remaining_s: nan
temperature: nan
voltage_cell_v: [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000]
max_cell_voltage_delta: 0.0000
custom_faults: 0
average_power: 0.0000
available_energy: 0.0000
full_charge_capacity_wh: 0.0000
remaining_capacity_wh: 0.0000
design_capacity: 0.0000
nominal_voltage: 0.0000
capacity: 0
cycle_count: 0
average_time_to_empty: 0
serial_number: 0
manufacture_date: 0
state_of_health: 0
max_error: 0
interface_error: 0
faults: 0
average_time_to_full: 0
over_discharge_count: 0
connected: True
cell_count: 0
source: 0
priority: 0
id: 1
is_powering_off: False
is_required: False
warning: 0
mode: 0

nsh>

```

Experiment Setup B: Subscribe to a Message

In this setup, an application was created to subscribe to specific uORB messages, showing the process of listening for and reacting to data published by other components within the PX4.

```

int hello_world_main(int argc, char *argv[])
{
    int sensor_combined_handle;
    sensor_combined_s sensor_data;
    sensor_combined_handle = orb_subscribe(ORB_ID(sensor_combined));
    orb_set_interval(sensor_combined_handle, 200);

    while (1)
    {
        orb_copy(ORB_ID(sensor_combined), sensor_combined_handle, &sensor_data);
        PX4_INFO("X = %f, Y = %f, Z = %f", (double)sensor_data.accelerometer_m_s2[0],
                (double)sensor_data.accelerometer_m_s2[1],
                (double)sensor_data.accelerometer_m_s2[2]);
        px4_usleep(200000);
    }
    return 0;
}

```

```
ds@ds-VirtualBox: ~  
INFO [hello_world] X = -0.280390, Y = -0.093456, Z = -9.902596  
INFO [hello_world] X = -0.261361, Y = -0.075408, Z = -9.922482  
INFO [hello_world] X = -0.263408, Y = -0.062331, Z = -9.880820  
INFO [hello_world] X = -0.258325, Y = -0.057023, Z = -9.843499  
INFO [hello_world] X = -0.242223, Y = -0.084151, Z = -9.847960  
INFO [hello_world] X = -0.248970, Y = -0.078193, Z = -9.855522  
INFO [hello_world] X = -0.237502, Y = -0.069638, Z = -9.808274  
INFO [hello_world] X = -0.276283, Y = -0.105612, Z = -9.882857  
INFO [hello_world] X = -0.275252, Y = -0.081756, Z = -9.866829  
INFO [hello_world] X = -0.280396, Y = -0.078385, Z = -9.907719  
INFO [hello_world] X = -0.242260, Y = -0.091503, Z = -9.845654  
INFO [hello_world] X = -0.258363, Y = -0.068581, Z = -9.895215  
INFO [hello_world] X = -0.261131, Y = -0.093909, Z = -9.880838  
INFO [hello_world] X = -0.271269, Y = -0.067187, Z = -9.857674  
INFO [hello_world] X = -0.268110, Y = -0.079522, Z = -9.871714  
INFO [hello_world] X = -0.226920, Y = -0.070878, Z = -9.845838  
INFO [hello_world] X = -0.277607, Y = -0.079345, Z = -9.866745  
INFO [hello_world] X = -0.261145, Y = -0.079334, Z = -9.871552  
INFO [hello_world] X = -0.224664, Y = -0.074466, Z = -9.693868  
INFO [hello_world] X = -0.308247, Y = -0.084198, Z = -9.909088  
INFO [hello_world] X = -0.237783, Y = -0.086165, Z = -9.842689  
INFO [hello_world] X = -0.235388, Y = -0.066640, Z = -9.857388  
INFO [hello_world] X = -0.253626, Y = -0.083964, Z = -9.859780
```

Project 2 – Step 0: RC Channels Reading Application

The project involved creating an application to print the values of RC channels every 0.2 seconds data from remote control, utilizing uORB subscriptions.

```
ds@ds-VirtualB  
INFO [hello_world] Channel 7 = 1.000000  
INFO [hello_world] Channel 8 = 1.000000  
INFO [hello_world] Channel 9 = 0.028000  
INFO [hello_world] Channel 10 = 0.028000  
INFO [hello_world] Channel 11 = 0.028000  
INFO [hello_world] Channel 12 = 0.028000  
INFO [hello_world] Channel 13 = 0.028000  
INFO [hello_world] Channel 14 = 0.028000  
INFO [hello_world] Channel 15 = 0.028000  
INFO [hello_world] Channel 16 = -1.000000  
INFO [hello_world] Channel 17 = -1.000000  
INFO [hello_world] Channel 0 = 0.008163  
INFO [hello_world] Channel 1 = 0.008163  
INFO [hello_world] Channel 2 = 1.000000  
INFO [hello_world] Channel 3 = 0.008163  
INFO [hello_world] Channel 4 = 1.000000  
INFO [hello_world] Channel 5 = 1.000000  
INFO [hello_world] Channel 6 = 1.000000  
INFO [hello_world] Channel 7 = 1.000000  
INFO [hello_world] Channel 8 = 1.000000  
INFO [hello_world] Channel 9 = 0.028000  
INFO [hello_world] Channel 10 = 0.028000  
INFO [hello_world] Channel 11 = 0.028000  
INFO [hello_world] Channel 12 = 0.028000  
INFO [hello_world] Channel 13 = 0.028000  
INFO [hello_world] Channel 14 = 0.028000  
INFO [hello_world] Channel 15 = 0.028000  
INFO [hello_world] Channel 16 = -1.000000  
INFO [hello_world] Channel 17 = -1.000000
```

Experiment Setup C: Publish a Message

This task highlighted the opposite end of uORB communication by developing an application that publishes messages. It showed how to send data to other components within the PX4.

```
ds@ds-VirtualBox: ~  
NuttShell (NSH) NuttX-10.2.0  
nsh>  
nsh> hello_world  
INFO [hello_world] Enter speed value (0 to 1). If you enter a value outside the  
range,the motor will be stopped and the appli  
INFO [hello_world] Motor speed is 0.500000  
INFO [hello_world] Angle speed is 0.000000  
INFO [hello_world] Enter angle value (0 to 1). If you enter a value outside the  
range,the motor will be stopped and the appli  
INFO [hello_world] Enter speed value (0 to 1). If you enter a value outside the  
range,the motor will be stopped and the appli  
INFO [hello_world] Motor speed is 0.530000  
INFO [hello_world] Angle speed is 0.500000  
INFO [hello_world] Enter angle value (0 to 1). If you enter a value outside the  
range,the motor will be stopped and the appli  
INFO [hello_world] Enter speed value (0 to 1). If you enter a value outside the  
range,the motor will be stopped and the appli  
INFO [hello_world] Motor speed is 0.500000  
INFO [hello_world] Angle speed is 0.300000  
INFO [hello_world] Enter angle value (0 to 1). If you enter a value outside the  
range,the motor will be stopped and the appli
```

```
test_motor_s test_motor;  
double motor_value = 0; // a number between 0 to 1  
double angle_value = 0; // a number between 0 to 1  
uORB::Publication<test_motor_s> test_motor_pub(ORB_ID(test_motor));  
  
int rc_channels_handle;  
  
rc_channels_s sensor_data;  
rc_channels_handle = orb_subscribe(ORB_ID(rc_channels));  
orb_set_interval(rc_channels_handle, 200);  
  
PX4_INFO("Enter speed value (0 to 1). If you enter a value outside the range, the motor will be stopped  
scanf("%lf", &motor_value);  
  
PX4_INFO("Enter angle value (0 to 1). If you enter a value outside the range, the motor will be stopped  
scanf("%lf", &angle_value);
```



```

while(1)
{
    PX4_INFO("Enter speed value (0 to 1). If you enter a value outside the range,the motor will be stopped");
    scanf("%lf", &motor_value);
    if(motor_value > 1.0 || motor_value < 0)
        break;

    PX4_INFO("Motor speed is %f", motor_value);

    test_motor.timestamp = hrt_absolute_time();
    test_motor.motor_number = DC_MOTOR;
    test_motor.value = (float)motor_value;
    test_motor.action = test_motor_s::ACTION_RUN;
    test_motor.driver_instance = 0;
    test_motor.timeout_ms = 0;

    test_motor_pub.publish(test_motor);

    PX4_INFO("Enter angle value (0 to 1). If you enter a value outside the range,the motor will be stopped");
    scanf("%lf", &angle_value);
    PX4_INFO("Angle speed is %f", angle_value);
    if(angle_value > 1.0 || angle_value < 0)
        break;

    test_motor.timestamp = hrt_absolute_time();
    test_motor.motor_number = Angle_MOTOR;
    test_motor.value = (float)angle_value;
    test_motor.action = test_motor_s::ACTION_RUN;
    test_motor.driver_instance = 0;
    test_motor.timeout_ms = 0;

    test_motor_pub.publish(test_motor);
}

```

Project 2 – Part 1 (Revisiting Project 1 using PX4)

Revisiting concepts from Project 1, this project integrated controlling mechanisms and sensor data handling within the PX4 framework. It focused on enhancing vehicle control through the development of more sophisticated applications that use various PX4 capabilities.

(a) Modifying code from Experiment 3C to control both the dc motor and the servo.

This task required modifying existing code for control over both the DC motor and the servo.

(b) Writing code that reads RC channel data and controls motors accordingly.

This task focused on developing code capable of interpreting real-time RC channel inputs and translating these into actionable commands for the vehicle's motors.

```

test_motor_s test_motor;
double motor_value = 0; // a number between 0 to 1
double servo_value = 0; // a number between 0 to 1
uORB::Publication<test_motor_s> test_motor_pub(ORB_ID(test_motor));

int rc_channels_handle;

rc_channels_s sensor_data;
rc_channels_handle = orb_subscribe(ORB_ID(rc_channels));
orb_set_interval(rc_channels_handle, 200);

PX4_INFO("Enter speed value (0 to 1). If you enter a value outside the range, the motor will be stopped");
scanf("%lf", &motor_value);

PX4_INFO("Enter angle value (0 to 1). If you enter a value outside the range, the motor will be stopped");
scanf("%lf", &servo_value);

```



```

while (1)
{
    orb_copy(ORB_ID(rc_channels), rc_channels_handle, &sensor_data);

    motor_value = ((double)sensor_data.channels[2] + 1.0) / 2.0;
    if (motor_value > 1.0 || motor_value < 0)
        break;

    servo_value = ((double)sensor_data.channels[0] * (-1.0) + 1.0) / 2.0;
    if (servo_value > 1.0 || servo_value < 0)
        break;

    PX4_INFO("Motor speed is %f", motor_value);
    test_motor.timestamp = hrt_absolute_time();
    test_motor.motor_number = DC_MOTOR;
    test_motor.value = (float)motor_value;
    test_motor.action = test_motor_s::ACTION_RUN;
    test_motor.driver_instance = 0;
    test_motor.timeout_ms = 0;
    test_motor_pub.publish(test_motor);

    PX4_INFO("Servo angle is %f", servo_value);
    test_motor.timestamp = hrt_absolute_time();
    test_motor.motor_number = SERVO_MOTOR;
    test_motor.value = (float)servo_value;
    test_motor.action = test_motor_s::ACTION_RUN;
    test_motor.driver_instance = 0;
    test_motor.timeout_ms = 0;
    test_motor_pub.publish(test_motor);
}

```

(c) BONUS – Write code that publishes a control message to change the LED colour.

Not completed.

(d) BONUS – Modifying the code to mimic Project 1 functionalities, integrating accelerometer readings, RC channel reading with gear/speed modes, motor controls, and LED colour controls.

Not completed.

Experiment 4: MAVLink

Experiment 4 introduced MAVLink, a communication protocol used for messaging between the FMU and external devices. The objective was to establish and understand the setup and operation of MAVLink communication.

Experiment Setup A: Setting up the FMU side and Reading Messages from FMU

This setup involved configuring the FMU to send and receive MAVLink messages. It included creating a Python application to interact with the FMU.

```
ds@ds-VirtualBox: ~/Documents
ATTITUDE {time_boot_ms : 602443, roll : -0.01263275183737278, pitch : -0.0231485
36682128906, yaw : 0.5051994323730469, rollspeed : 0.0002699521428439766, pitchs
peed : 0.0018575440626591444, yawspeed : -0.0032055042684078217}
ATTITUDE {time_boot_ms : 602463, roll : -0.012759173288941383, pitch : -0.023135
429248213768, yaw : 0.5051926374435425, rollspeed : -0.005431117955595255, pitch
speed : 0.002597651444375515, yawspeed : -0.003380856476724148}
ATTITUDE {time_boot_ms : 602484, roll : -0.012748119421303272, pitch : -0.023158
939897680283, yaw : 0.5050444602966309, rollspeed : 0.004548154771327972, pitchs
peed : -0.0007558721117675304, yawspeed : -0.007465820759534836}
ATTITUDE {time_boot_ms : 602504, roll : -0.01270401943475008, pitch : -0.0230904
61269021034, yaw : 0.5049421191215515, rollspeed : 0.002353256568312645, pitchsp
eed : 0.004657434299588203, yawspeed : -0.00032648537307977676}
ATTITUDE {time_boot_ms : 602524, roll : -0.012780461460351944, pitch : -0.023124
583065509796, yaw : 0.5048804879188538, rollspeed : -0.004775949753820896, pitch
speed : -0.0009081149473786354, yawspeed : -0.002621422754600644}
ATTITUDE {time_boot_ms : 602543, roll : -0.01278757769614458, pitch : -0.0231162
49591112137, yaw : 0.504806637763977, rollspeed : -0.0028885777574032545, pitchs
peed : 0.0008210139349102974, yawspeed : -0.004379713907837868}
ATTITUDE {time_boot_ms : 602565, roll : -0.012775758281350136, pitch : -0.023160
168901085854, yaw : 0.504748523235321, rollspeed : 0.00018315202032681555, pitch
speed : -0.0039959014393389225, yawspeed : -0.004097972996532917}
ATTITUDE {time_boot_ms : 602584, roll : -0.01263553255677223, pitch : -0.023170
97783088684, yaw : 0.5046669244766235, rollspeed : 0.00710714515298605, pitchspe
ed : 0.0010158051736652851, yawspeed : -0.005661752540618181}
```

Experiment Setup B: Read Custom Messages from FMU

This experiment showed how to customize MAVLink communication by developing applications that send and receive specific messages.

```
ds@ds-VirtualBox: ~/Documents
DEBUG {time_boot_ms : 1210355, ind : 154, value : 77.0}
DEBUG {time_boot_ms : 1211355, ind : 155, value : 77.5}
DEBUG {time_boot_ms : 1212356, ind : 156, value : 78.0}
DEBUG {time_boot_ms : 1213357, ind : 157, value : 78.5}
DEBUG {time_boot_ms : 1213878, ind : 157, value : 78.5}
DEBUG {time_boot_ms : 1214359, ind : 158, value : 79.0}
DEBUG {time_boot_ms : 1215359, ind : 159, value : 79.5}
DEBUG {time_boot_ms : 1216361, ind : 160, value : 80.0}
DEBUG {time_boot_ms : 1217362, ind : 161, value : 80.5}
DEBUG {time_boot_ms : 1218362, ind : 162, value : 81.0}
DEBUG {time_boot_ms : 1219363, ind : 163, value : 81.5}
DEBUG {time_boot_ms : 1219574, ind : 0, value : 1.0}
DEBUG {time_boot_ms : 1220364, ind : 164, value : 82.0}
DEBUG {time_boot_ms : 1221366, ind : 165, value : 82.5}
DEBUG {time_boot_ms : 1222367, ind : 166, value : 83.0}
DEBUG {time_boot_ms : 1223251, ind : 166, value : 83.0}
DEBUG {time_boot_ms : 1223371, ind : 167, value : 83.5}
DEBUG {time_boot_ms : 1224373, ind : 168, value : 84.0}
DEBUG {time_boot_ms : 1225373, ind : 169, value : 84.5}
DEBUG {time_boot_ms : 1226363, ind : 169, value : 84.5}
DEBUG {time_boot_ms : 1226374, ind : 170, value : 85.0}
DEBUG {time_boot_ms : 1227376, ind : 171, value : 85.5}
DEBUG {time_boot_ms : 1228377, ind : 172, value : 86.0}
```

Experiment Setup C: Send Messages to FMU

This step focused on sending customized messages to the FMU via MAVLink, showing how external inputs can influence the vehicle's operations.

```

int hello_world_main(int argc, char *argv[])
{
    px4_sleep(2);
    debug_value_s debug_data;
    int debug_handle = orb_subscribe(ORB_ID(debug_value));
    orb_set_interval(debug_handle, 500);
    led_control_s led_control;
    uORB::Publication<led_control_s> led_control_pub(ORB_ID(led_control));
    while (1)
    {
        orb_copy(ORB_ID(debug_value), debug_handle, &debug_data);
        led_control.timestamp = hrt_absolute_time();
        led_control.color = led_control_s::COLOR_GREEN;
        led_control.priority = led_control_s::MAX_PRIORITY;
        led_control.led_mask = 0xff;
        if (debug_data.ind == 1)
            led_control.mode = led_control_s::MODE_ON;
        else
            led_control.mode = led_control_s::MODE_OFF;
        led_control_pub.publish(led_control);
        px4_usleep(500000);
    }
    return 0;
}

```

```

from pymavlink import mavutil
import time

# Start a connection
the_connection = mavutil.mavlink_connection("/dev/ttyACM0")

# Wait for the first heartbeat
# This sets the system and component ID of remote system for the link
the_connection.wait_heartbeat()
print(
    "Heartbeat from system (system %u component %u)"
    % (the_connection.target_system, the_connection.target_component)
)

# Once connected, use 'the_connection' to get and send messages
value = 0
while 1:
    message = mavutil.mavlink.MAVLink_debug_message(0, value, 0.0)
    the_connection.mav.send(message)
    time.sleep(1)
    value = (value + 1) % 2
    print("Message sent")

```

```
ds@ds-VirtualBox: ~/Documents/Group
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
Message sent
```

Experiment 5: Ultrasonic Sensor

The experiment incorporated an ultrasonic sensor with the Raspberry Pi to measure distances to obstacles, showing sensor integration into the autonomous system for obstacle detection and avoidance. The provided Python code can be found in the project folder.

Experiment 6: Camera

This stage used a camera module with the Raspberry Pi for visual navigation and obstacle avoidance, showcasing image processing and decision-making based on visual inputs. The provided Python code can be found in the project folder.

Project 2 – Part 2

This segment of the project involved implementing an advanced obstacle detection and response system within the autonomous vehicle. The developed system utilized sensor inputs to dynamically adjust the vehicle's speed based on proximity to obstacles. Specifically, it was programmed to initiate deceleration when obstacles were detected within 50 cm and to bring the vehicle to a complete stop if obstacles approached within 15 cm. The first image is the Python code running of the RaspberryPi and the remaining images show the C++ code running on the FMU to control both the motor and servo.

```

while 1:
    dist = distance()
    print("Measured Distance = %.1f cm" % dist)

    ret = process_camera_frame()
    if ret == 0:
        print("Left direction is preferred")
    elif ret == 1:
        print("Forward direction is preferred")
    elif ret == 2:
        print("Right direction is preferred")

    if dist <= 15:
        speed = 0
    elif 15 < dist < 50:
        speed = 1
    else:
        speed = 2

    message = mavutil.mavlink.MAVLink_debug_message(0, int(ret), int(speed))
    the_connection.mav.send(message)
    print(f"Sent message with speed = {speed} and direction = {ret}")

    received_msg = the_connection.recv_match(type="DEBUG", blocking=True)
    print(
        f"Received from FMU: speed = {received_msg.value}, direction = {received_msg.ind}"
    )

```

```
// Set speed based on pi distance
if (speed == 1)
{
    test_motor.value = 0.6;
}
else if (speed == 2)
{
    test_motor.value = 0.9;
}
else
{ // speed = 0
    test_motor.value = 0.5;
}

// Set direction based on pi camera
if (direction == 0)
{
    test_servo.value = 0.1;
}
else if (direction == 2)
{
    test_servo.value = 0.9;
}
else
{ // direction = 1
    test_servo.value = 0.5;
}
```

```
debug_data.timestamp = hrt_absolute_time();  
debug_value_pub.publish(debug_data);  
  
test_motor_pub.publish(test_motor);  
test_servo_pub.publish(test_servo);  
px4_usleep(50000);
```