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

<table>
<tr><th>Tasks</th><th>Contributor(s)</th></tr>
<tr><td>Experiment 1: Build and Program PX4</td><td rowspan="17">Cameron Beneteau, Shaqif Ahmed, Tyler Moss, and Noah Betik</td></tr>
<tr><td>Experiment 2: Write Applications on NuttX with PX4</td></tr>
<tr><td>Setup A: Import PX4 to MCUXpresso</td></tr>
<tr><td>Setup B: Hello World Application</td></tr>
<tr><td>Experiment 3: uORB Messaging</td></tr>
<tr><td>Experiment Setup A: Interact with uORB</td></tr>
<tr><td>Experiment Setup B: Subscribe to a Message</td></tr>
<tr><td>Project 2 – Step 0: RC Channels Reading Application</td></tr>
<tr><td>Experiment Setup C: Publish a Message</td></tr>
<tr><td>Project 2 – Part 1 (Revisiting Project 1 using PX4)</td></tr>
<tr><td>Experiment 4: MAVLink</td></tr>
<tr><td>Experiment Setup A: Setting up the FMU side and Reading Messages from FMU</td></tr>
<tr><td>Experiment Setup B: Read Custom Messages from FMU</td></tr>
<tr><td>Experiment Setup C: Send Messages to FMU</td></tr>
<tr><td>Experiment 5: Ultrasonic Sensor</td></tr>
<tr><td>Experiment 6: Camera</td></tr>
<tr><td>Project 2 – Part 2</td></tr>
</table>

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.

```c
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>
```

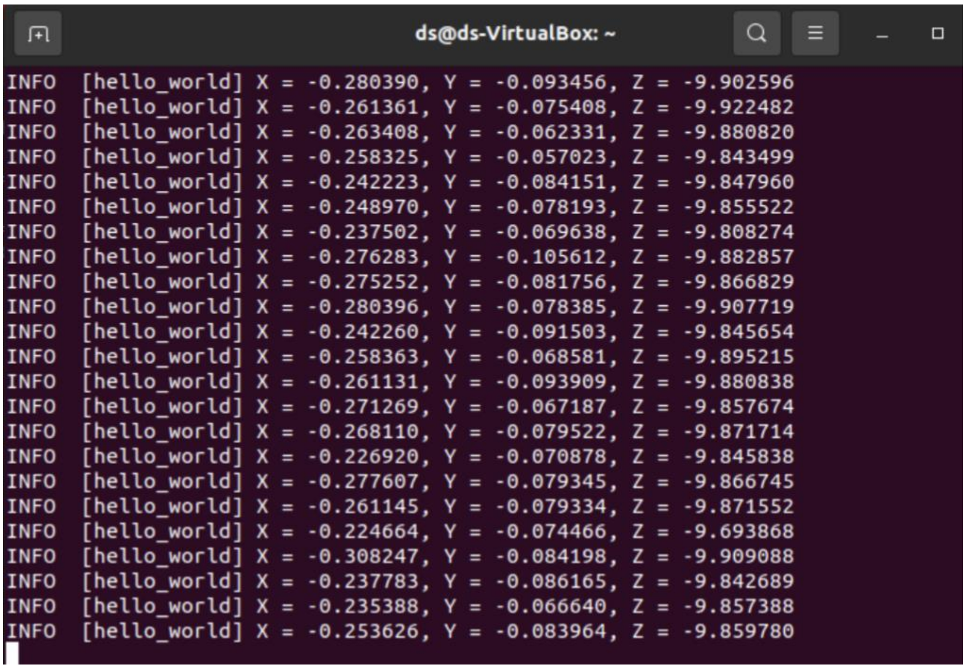
```
nsh> listener vehicle_air_data

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

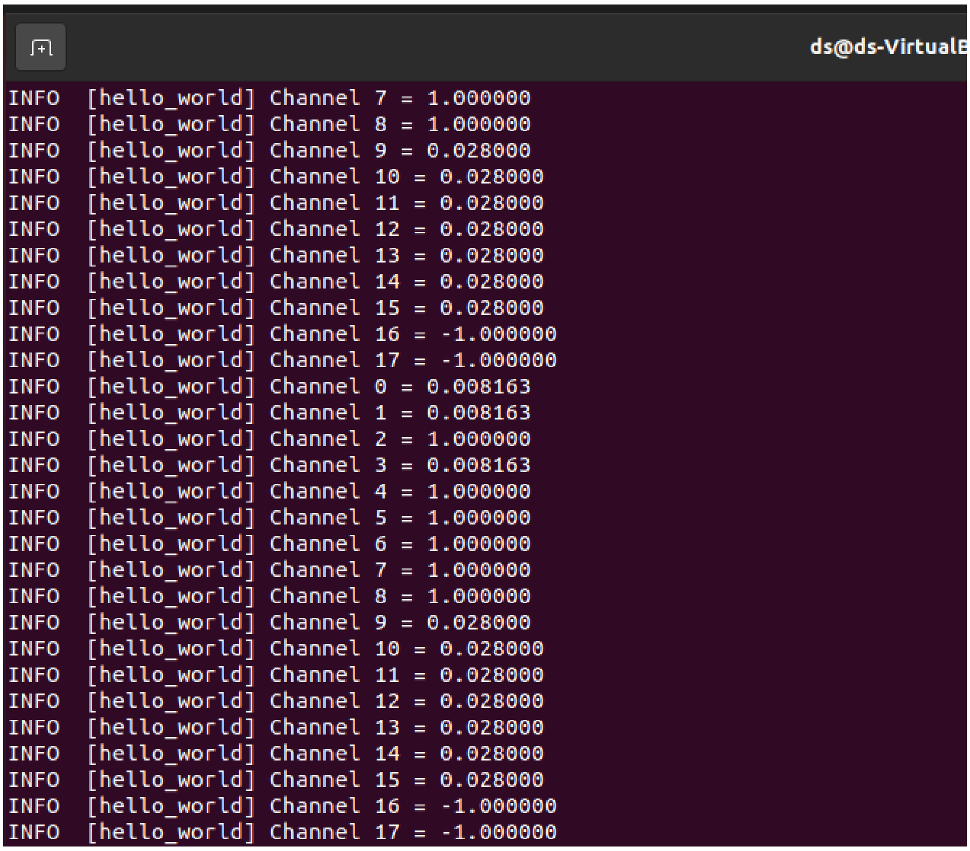
In this setup, an application was created to subscribe to specific uORB messages, showing the process of listening for and reacting to data published by other components within the PX4.

```
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;
}
```



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.



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.

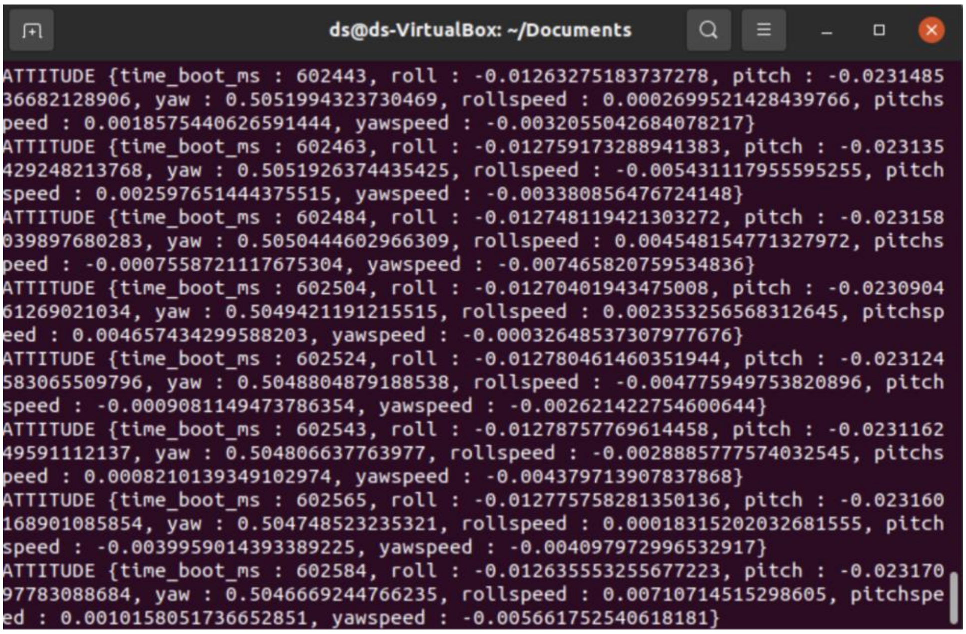
Not completed.

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.

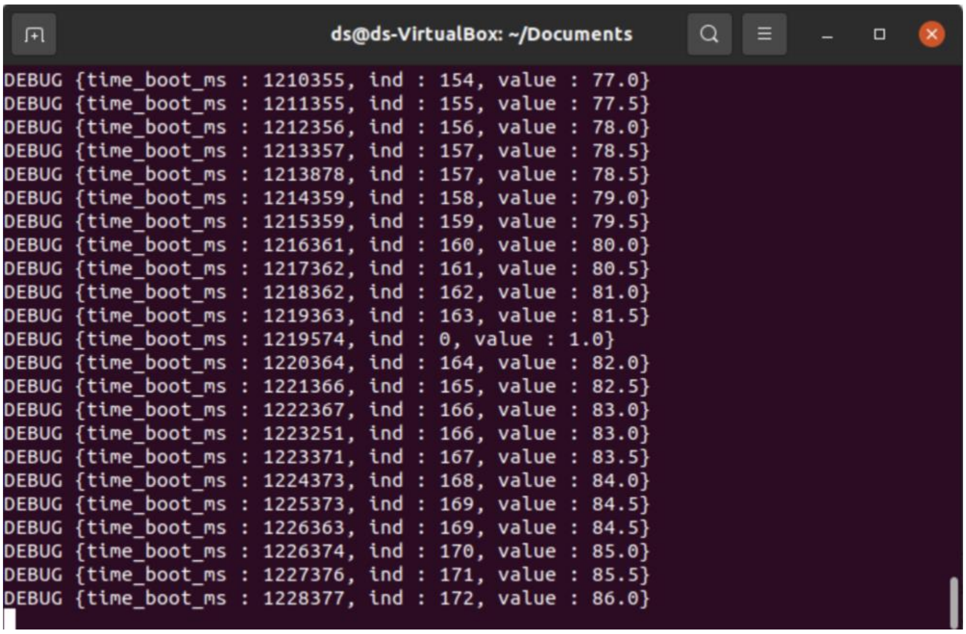
Experiment Setup A: Setting up the FMU side and Reading Messages from FMU

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



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.

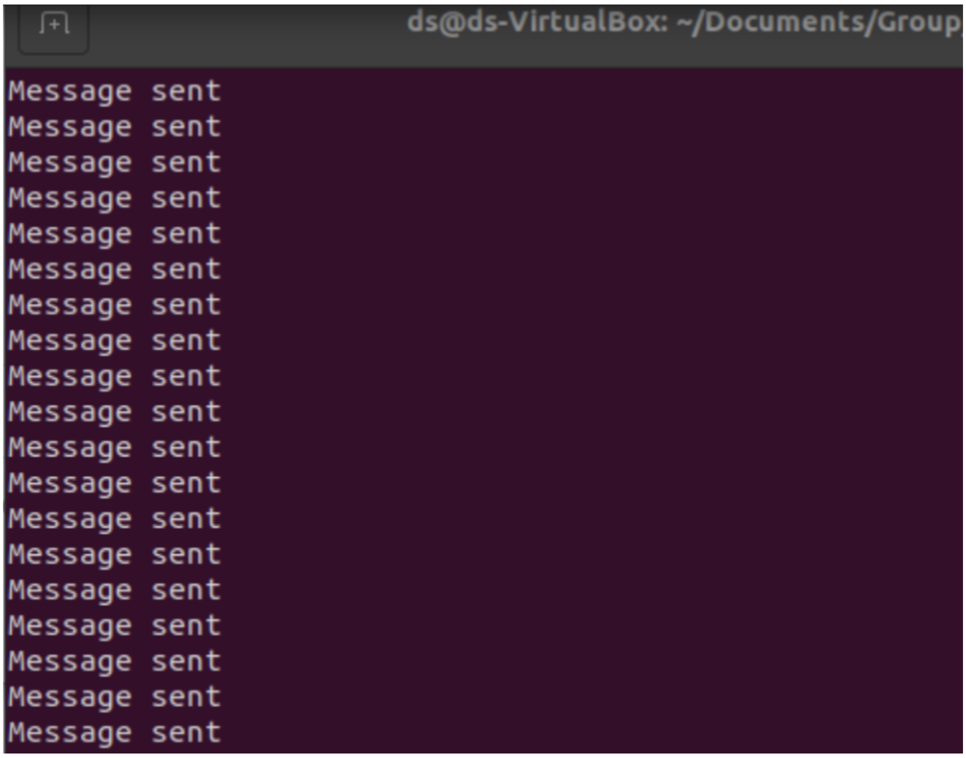


Experiment Setup C: Send Messages to FMU

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

```cpp
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;
}
```

```python
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")
```



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.

```python
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);
```