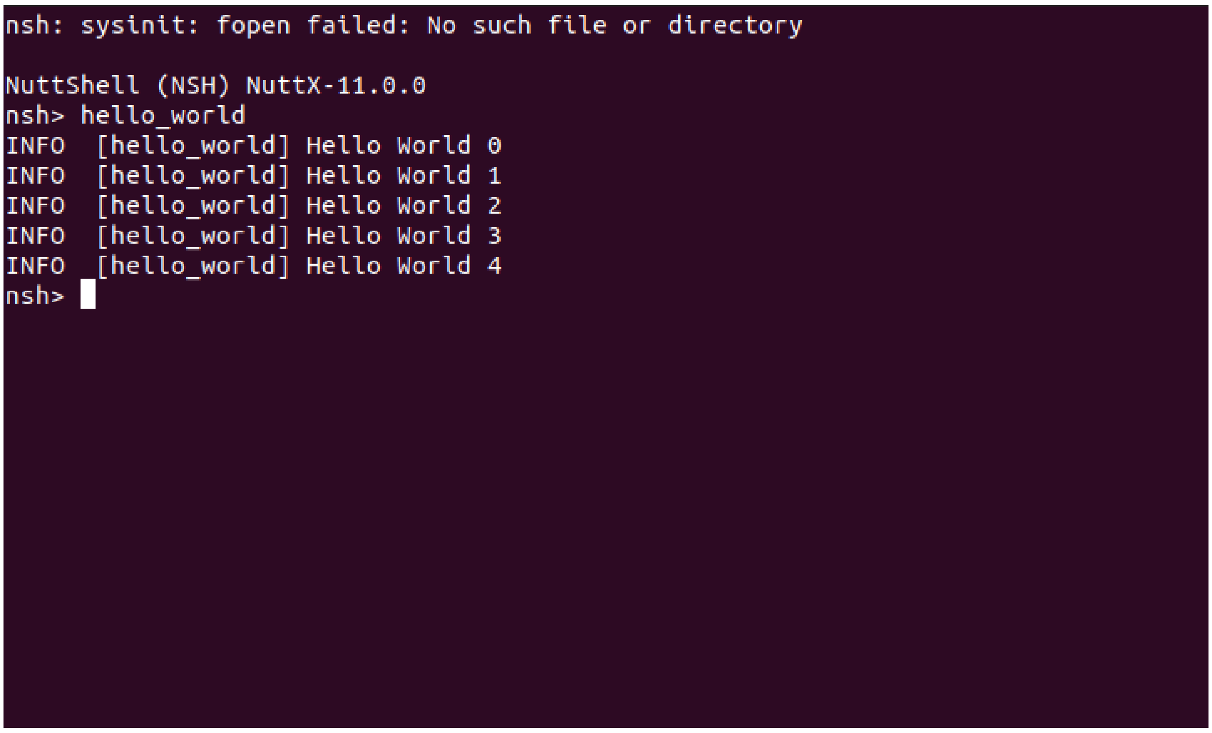
4DS4 Project 2:
Autonomous Vehicle: Putting it All Together
Group 04

Justin Covach covachj 400243271
Sava Jankovic 400292525
Cass Smith smithc85 400255184
Patrick Wang wangp56 400264434

Declaration of Contributions

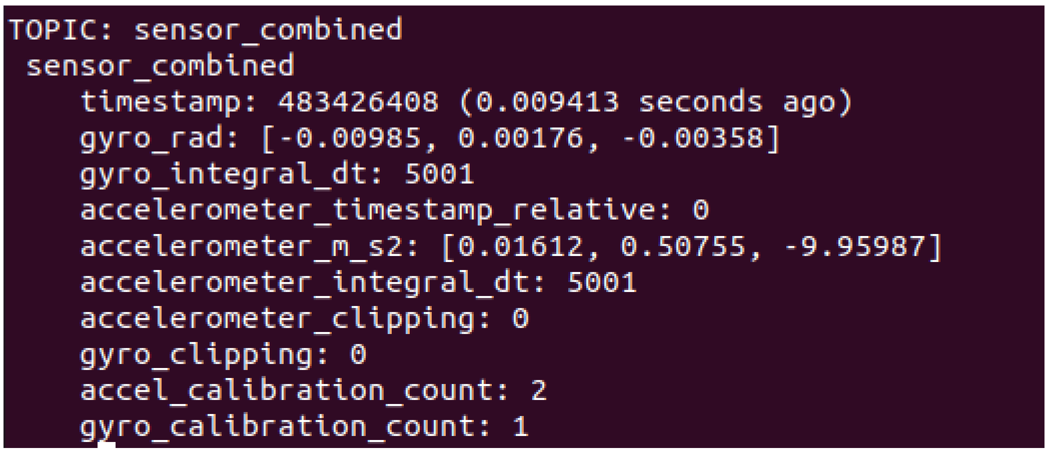
Member	Contribution
Justin Covach	Worked on all parts, and report
Sava Jankovic	Initial set up, partial LED, and final debugging
Cass Smith	Worked on RC and MAVLINK steps
Patrick Wang	Worked on all parts (except raspberry pi development) and report

Experiment 2: Write Applications on NuttX with PX4

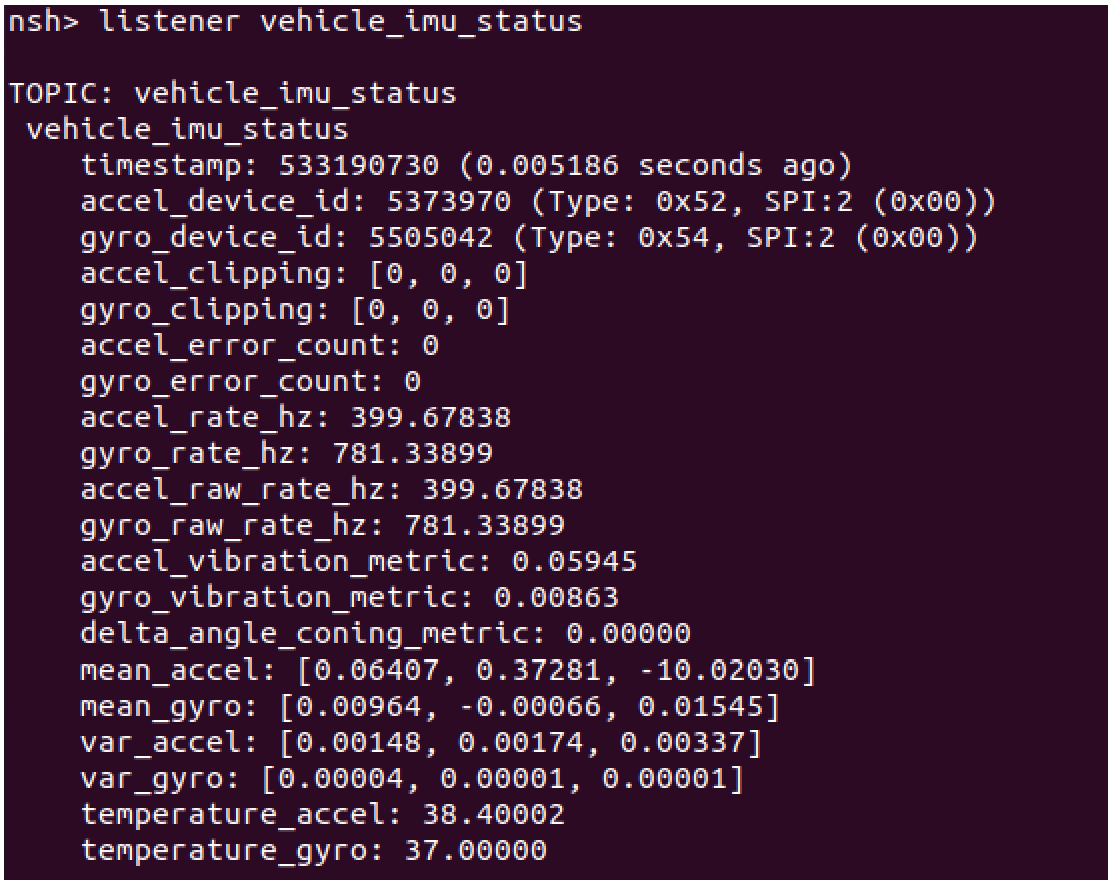


```
nsh: sysinit: fopen failed: No such file or directory

NuttShell (NSH) NuttX-11.0.0
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> 
```

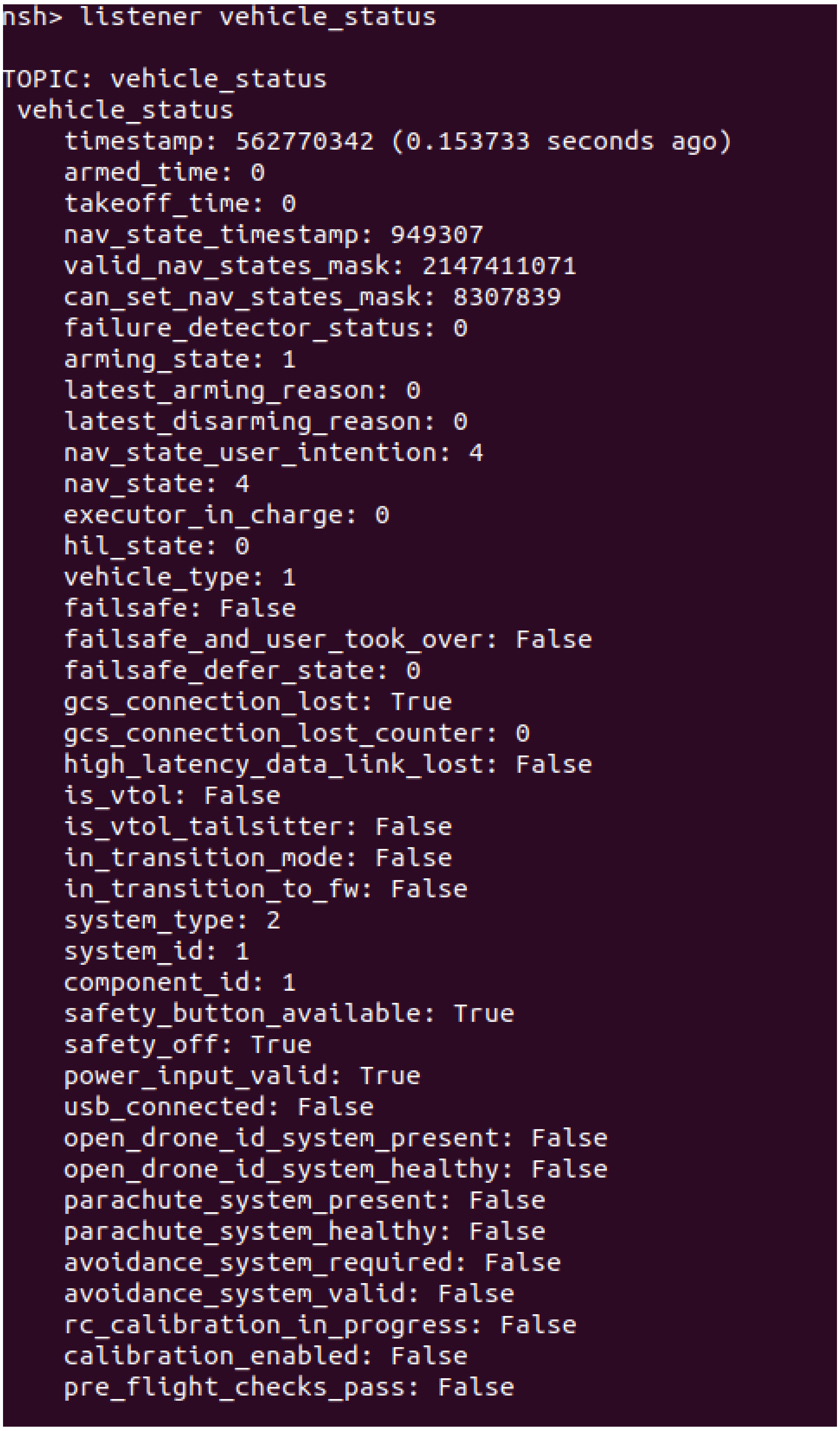
Experiment 3: uORB Messaging



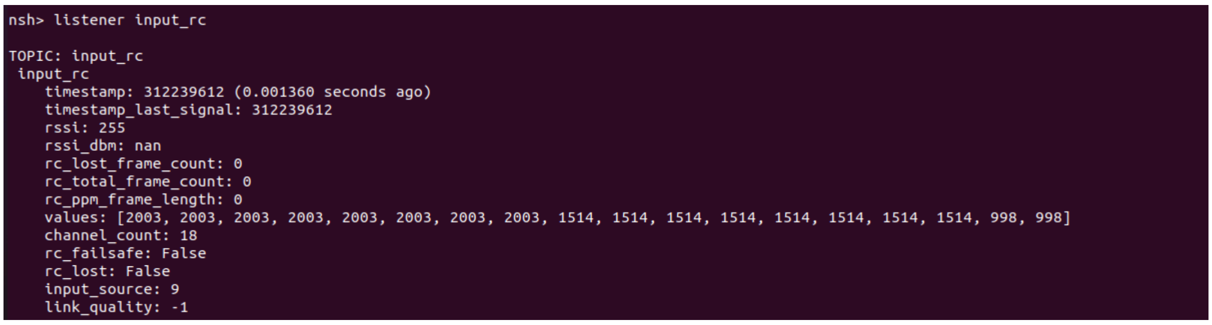
```
TOPIC: sensor_combined
 sensor_combined
    timestamp: 483426408 (0.009413 seconds ago)
    gyro_rad: [-0.00985, 0.00176, -0.00358]
    gyro_integral_dt: 5001
    accelerometer_timestamp_relative: 0
    accelerometer_m_s2: [0.01612, 0.50755, -9.95987]
    accelerometer_integral_dt: 5001
    accelerometer_clipping: 0
    gyro_clipping: 0
    accel_calibration_count: 2
    gyro_calibration_count: 1
```



```
nsh> listener vehicle_imu_status

TOPIC: vehicle_imu_status
 vehicle_imu_status
    timestamp: 533190730 (0.005186 seconds ago)
    accel_device_id: 5373970 (Type: 0x52, SPI:2 (0x00))
    gyro_device_id: 5505042 (Type: 0x54, SPI:2 (0x00))
    accel_clipping: [0, 0, 0]
    gyro_clipping: [0, 0, 0]
    accel_error_count: 0
    gyro_error_count: 0
    accel_rate_hz: 399.67838
    gyro_rate_hz: 781.33899
    accel_raw_rate_hz: 399.67838
    gyro_raw_rate_hz: 781.33899
    accel_vibration_metric: 0.05945
    gyro_vibration_metric: 0.00863
    delta_angle_coning_metric: 0.00000
    mean_accel: [0.06407, 0.37281, -10.02030]
    mean_gyro: [0.00964, -0.00066, 0.01545]
    var_accel: [0.00148, 0.00174, 0.00337]
    var_gyro: [0.00004, 0.00001, 0.00001]
    temperature_accel: 38.40002
    temperature_gyro: 37.00000
```



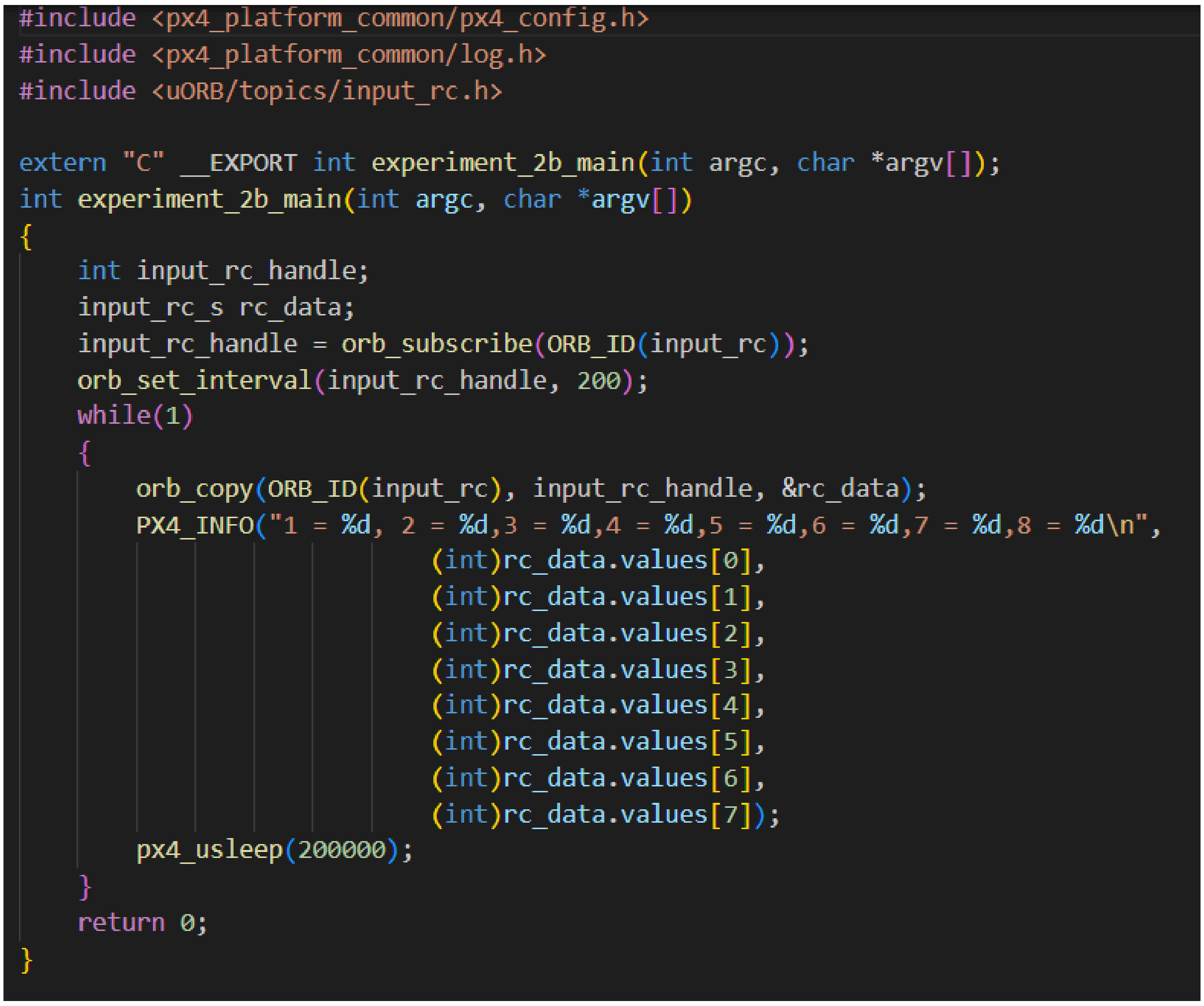
```
nsh> listener vehicle_status

TOPIC: vehicle_status
 vehicle_status
    timestamp: 562770342 (0.153733 seconds ago)
    armed_time: 0
    takeoff_time: 0
    nav_state_timestamp: 949307
    valid_nav_states_mask: 2147411071
    can_set_nav_states_mask: 8307839
    failure_detector_status: 0
    arming_state: 1
    latest_arming_reason: 0
    latest_disarming_reason: 0
    nav_state_user_intention: 4
    nav_state: 4
    executor_in_charge: 0
    hil_state: 0
    vehicle_type: 1
    failsafe: False
    failsafe_and_user_took_over: False
    failsafe_defer_state: 0
    gcs_connection_lost: True
    gcs_connection_lost_counter: 0
    high_latency_data_link_lost: False
    is_vtol: False
    is_vtol_tailsitter: False
    in_transition_mode: False
    in_transition_to_fw: False
    system_type: 2
    system_id: 1
    component_id: 1
    safety_button_available: True
    safety_off: True
    power_input_valid: True
    usb_connected: False
    open_drone_id_system_present: False
    open_drone_id_system_healthy: False
    parachute_system_present: False
    parachute_system_healthy: False
    avoidance_system_required: False
    avoidance_system_valid: False
    rc_calibration_in_progress: False
    calibration_enabled: False
    pre_flight_checks_pass: False
```



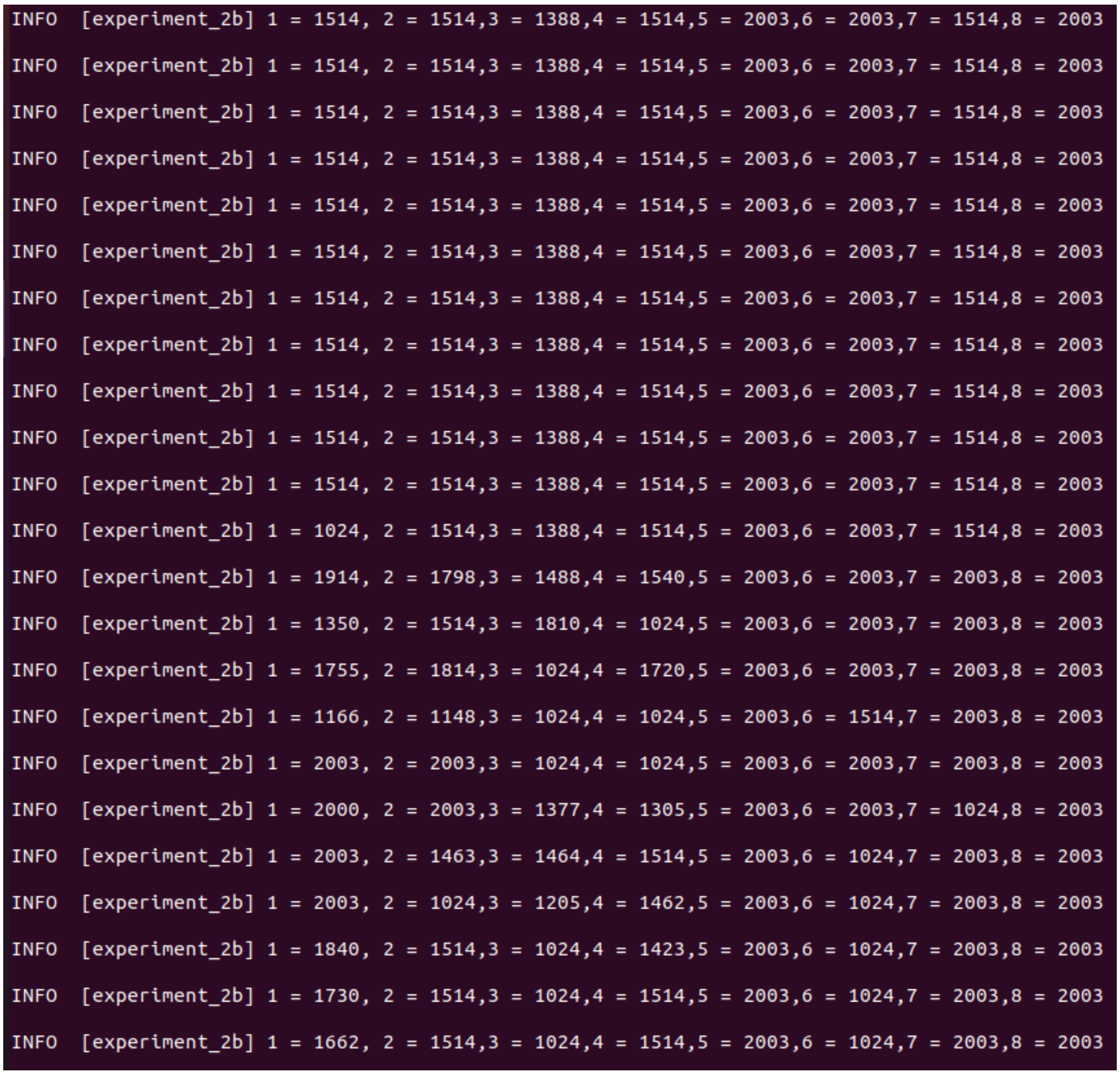
```
nsh> listener input_rc

TOPIC: input_rc
 input_rc
    timestamp: 312239612 (0.001360 seconds ago)
    timestamp_last_signal: 312239612
    rssi: 255
    rssi_dbm: nan
    rc_lost_frame_count: 0
    rc_total_frame_count: 0
    rc_ppm_frame_length: 0
    values: [2003, 2003, 2003, 2003, 2003, 2003, 2003, 2003, 1514, 1514, 1514, 1514, 1514, 1514, 1514, 1514, 998, 998]
    channel_count: 18
    rc_failsafe: False
    rc_lost: False
    input_source: 9
    link_quality: -1
```

Project 2 Step 0

In this part, we subscribed to the uORB topic, input_rc_handle. Data is then copied over and printed every 200ms. The data printed were from channels 1 to 8, which corresponds to the 2 joysticks, 2 of the 2 state switches and 2 of the 3 state switches.



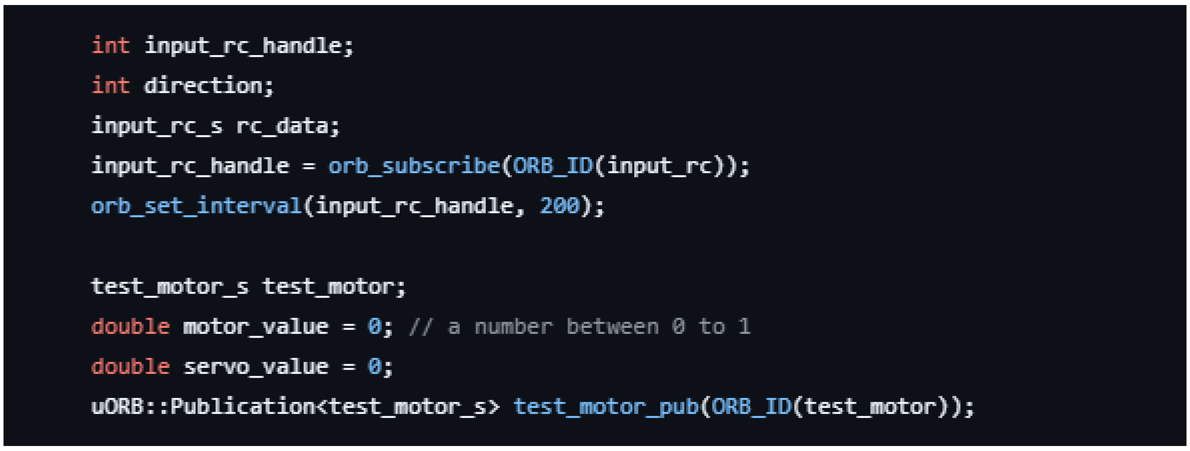
```
#include <px4_platform_common/px4_config.h>
#include <px4_platform_common/log.h>
#include <uORB/topics/input_rc.h>

extern "C" __EXPORT int experiment_2b_main(int argc, char *argv[]);
int experiment_2b_main(int argc, char *argv[])
{
    int input_rc_handle;
    input_rc_s rc_data;
    input_rc_handle = orb_subscribe(ORB_ID(input_rc));
    orb_set_interval(input_rc_handle, 200);
    while(1)
    {
        orb_copy(ORB_ID(input_rc), input_rc_handle, &rc_data);
        PX4_INFO("1 = %d, 2 = %d,3 = %d,4 = %d,5 = %d,6 = %d,7 = %d,8 = %d\n",
                     (int)rc_data.values[0],
                     (int)rc_data.values[1],
                     (int)rc_data.values[2],
                     (int)rc_data.values[3],
                     (int)rc_data.values[4],
                     (int)rc_data.values[5],
                     (int)rc_data.values[6],
                     (int)rc_data.values[7]);
        px4_usleep(200000);
    }
    return 0;
}
```



```
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1514, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1024, 2 = 1514,3 = 1388,4 = 1514,5 = 2003,6 = 2003,7 = 1514,8 = 2003
INFO  [experiment_2b] 1 = 1914, 2 = 1798,3 = 1488,4 = 1540,5 = 2003,6 = 2003,7 = 2003,8 = 2003
INFO  [experiment_2b] 1 = 1350, 2 = 1514,3 = 1810,4 = 1024,5 = 2003,6 = 2003,7 = 2003,8 = 2003
INFO  [experiment_2b] 1 = 1755, 2 = 1814,3 = 1024,4 = 1720,5 = 2003,6 = 2003,7 = 2003,8 = 2003
INFO  [experiment_2b] 1 = 1166, 2 = 1148,3 = 1024,4 = 1024,5 = 2003,6 = 1514,7 = 2003,8 = 2003
INFO  [experiment_2b] 1 = 2003, 2 = 2003,3 = 1024,4 = 1024,5 = 2003,6 = 2003,7 = 2003,8 = 2003
INFO  [experiment_2b] 1 = 2000, 2 = 2003,3 = 1377,4 = 1305,5 = 2003,6 = 2003,7 = 1024,8 = 2003
INFO  [experiment_2b] 1 = 2003, 2 = 1463,3 = 1464,4 = 1514,5 = 2003,6 = 1024,7 = 2003,8 = 2003
INFO  [experiment_2b] 1 = 2003, 2 = 1024,3 = 1205,4 = 1462,5 = 2003,6 = 1024,7 = 2003,8 = 2003
INFO  [experiment_2b] 1 = 1840, 2 = 1514,3 = 1024,4 = 1423,5 = 2003,6 = 1024,7 = 2003,8 = 2003
INFO  [experiment_2b] 1 = 1730, 2 = 1514,3 = 1024,4 = 1514,5 = 2003,6 = 1024,7 = 2003,8 = 2003
INFO  [experiment_2b] 1 = 1662, 2 = 1514,3 = 1024,4 = 1514,5 = 2003,6 = 1024,7 = 2003,8 = 2003
```

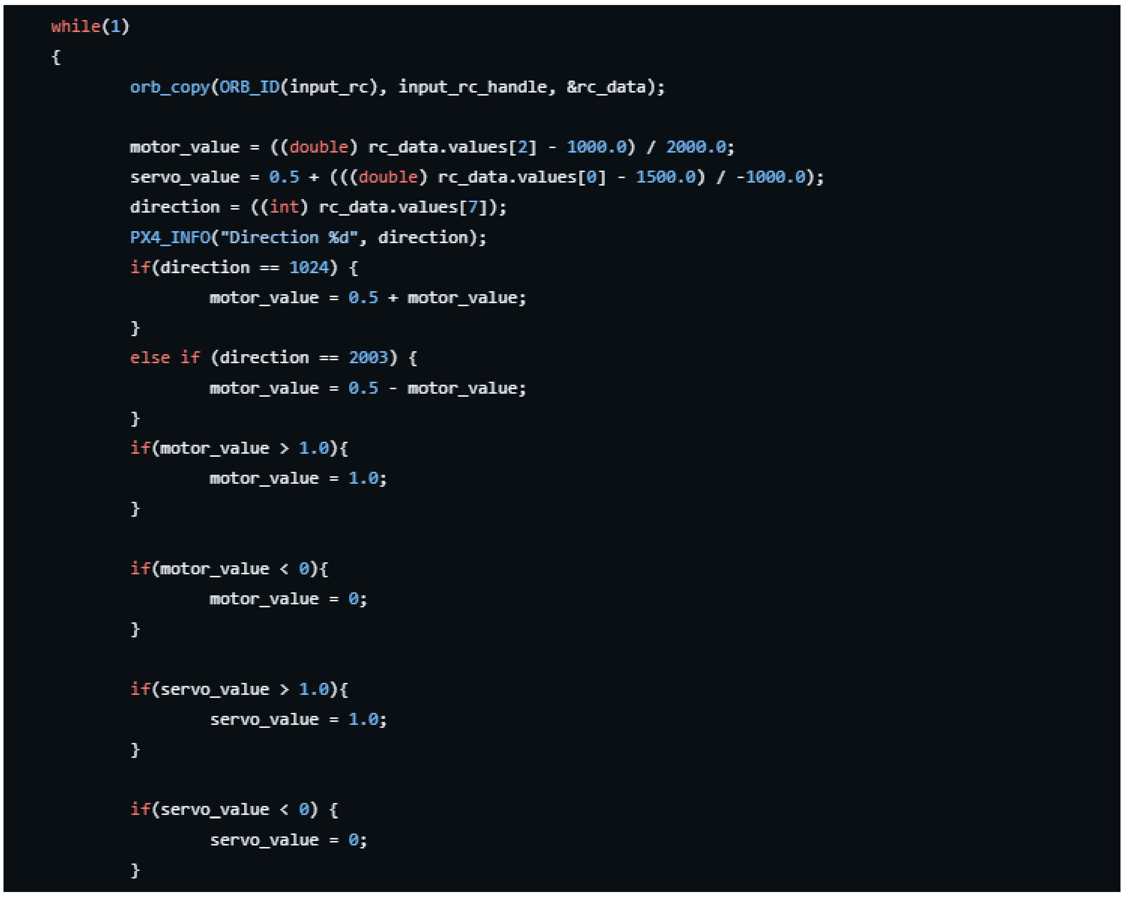
Project2 Part 1

This part is much like what we did in the following part 2 except instead of subscribing to a channel sent in by the PI, it is reading values from our RC controller. The input_rc header file was key in completing this part easily as it had a very simple format to handle all the different RC values. To begin we initialize a subscription as well as a publisher. One for listening to the RC inputs, and one for outputting to the 2 motors.



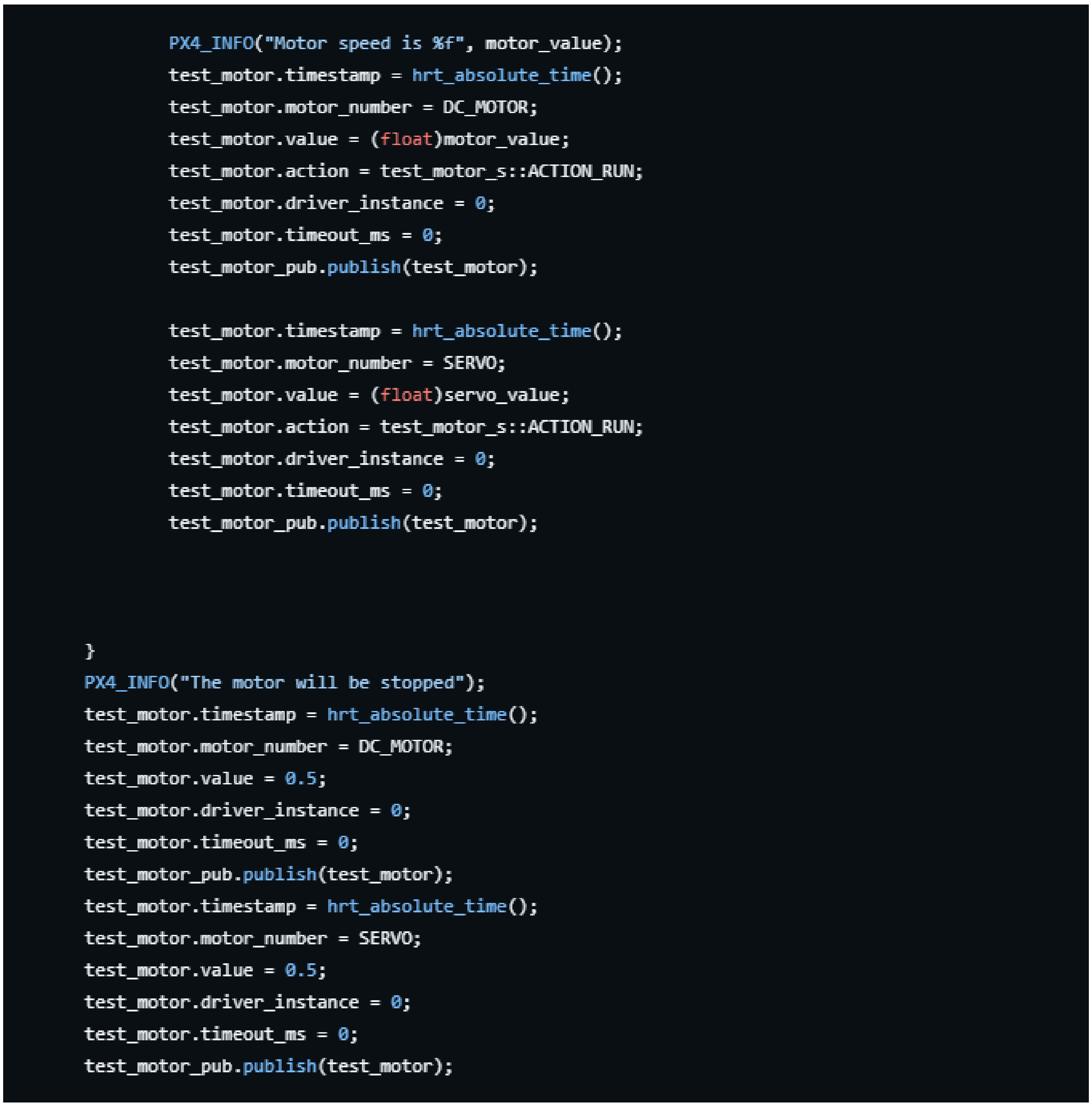
```
int input_rc_handle;
int direction;
input_rc_s rc_data;
input_rc_handle = orb_subscribe(ORB_ID(input_rc));
orb_set_interval(input_rc_handle, 200);

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

Then we enter the main loop. In the main loop, we read in the RC values using the orb_copy command. using this we can then interpret and scale botht he motor and servo values and set the accordingly. Since the incoming values are sometimes larger or smaller than the ranges in the previous labs, we also added some limits to the values in case they are larger than our specified range



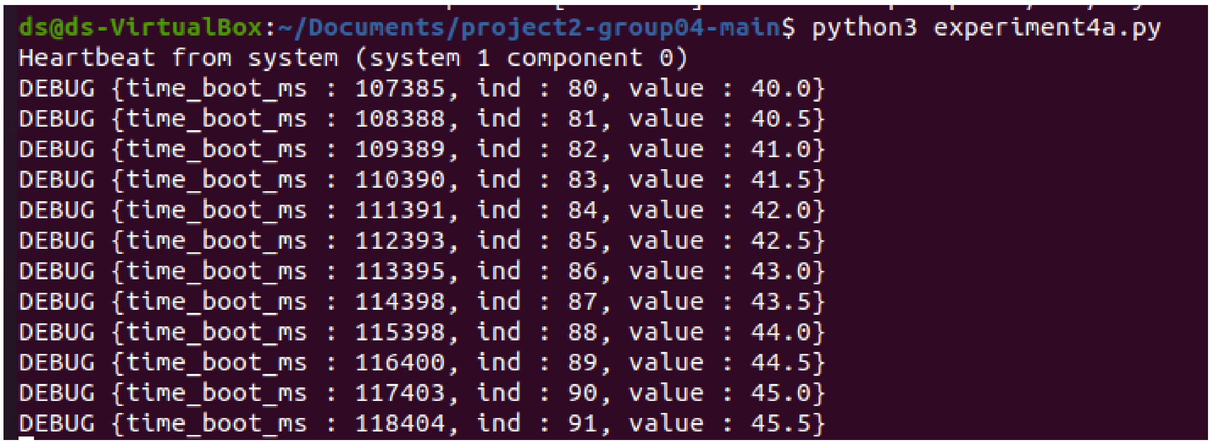
```
while(1)
{
        orb_copy(ORB_ID(input_rc), input_rc_handle, &rc_data);

        motor_value = ((double) rc_data.values[2] - 1000.0) / 2000.0;
        servo_value = 0.5 + (((double) rc_data.values[0] - 1500.0) / -1000.0);
        direction = ((int) rc_data.values[7]);
        PX4_INFO("Direction %d", direction);
        if(direction == 1024) {
                motor_value = 0.5 + motor_value;
        }
        else if (direction == 2003) {
                motor_value = 0.5 - motor_value;
        }
        if(motor_value > 1.0){
                motor_value = 1.0;
        }

        if(motor_value < 0){
                motor_value = 0;
        }

        if(servo_value > 1.0){
                servo_value = 1.0;
        }

        if(servo_value < 0) {
                servo_value = 0;
        }
```

Once all the values our set, we then update the publisher values and send to each motor their according values. These are updating them once we commit the publish command. Outside of the loop there is a fallback of completely stopped and no turning to reset the values at system end.



```
            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);

            test_motor.timestamp = hrt_absolute_time();
            test_motor.motor_number = SERVO;
            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);



    }
    PX4_INFO("The motor will be stopped");
    test_motor.timestamp = hrt_absolute_time();
    test_motor.motor_number = DC_MOTOR;
    test_motor.value = 0.5;
    test_motor.driver_instance = 0;
    test_motor.timeout_ms = 0;
    test_motor_pub.publish(test_motor);
    test_motor.timestamp = hrt_absolute_time();
    test_motor.motor_number = SERVO;
    test_motor.value = 0.5;
    test_motor.driver_instance = 0;
    test_motor.timeout_ms = 0;
    test_motor_pub.publish(test_motor);
```

Experiment 4B: Read Custom Messages from FMU

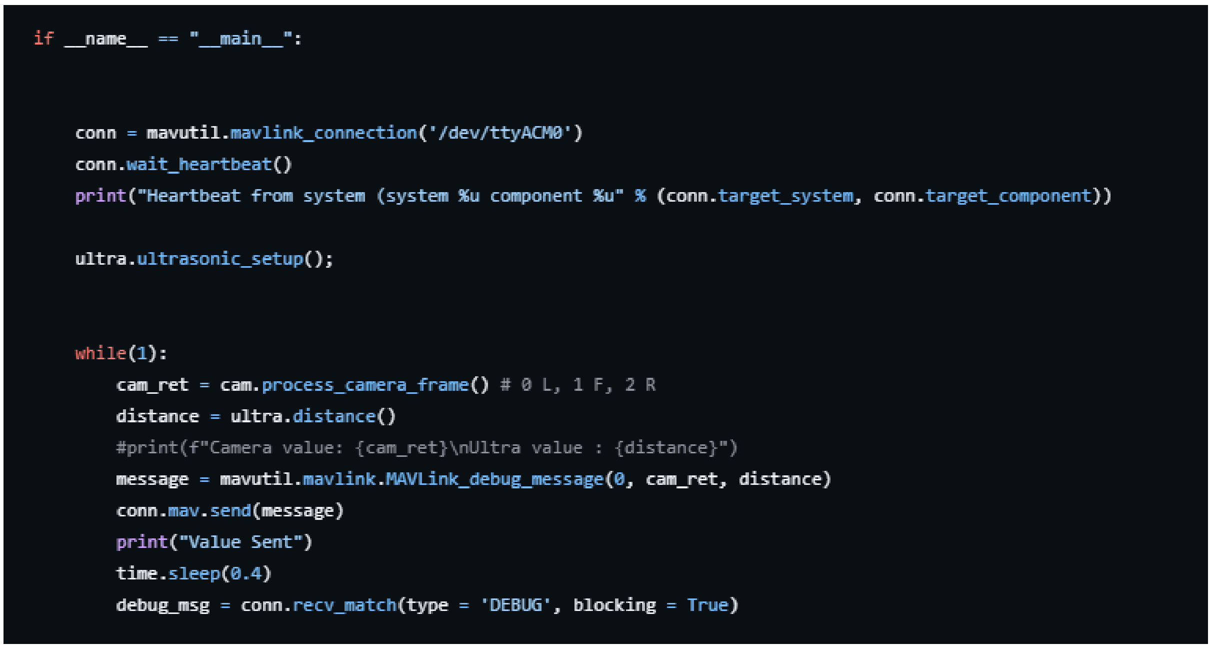


```
ds@ds-VirtualBox:~/Documents/project2-group04-main$ python3 experiment4a.py
Heartbeat from system (system 1 component 0)
DEBUG {time_boot_ms : 107385, ind : 80, value : 40.0}
DEBUG {time_boot_ms : 108388, ind : 81, value : 40.5}
DEBUG {time_boot_ms : 109389, ind : 82, value : 41.0}
DEBUG {time_boot_ms : 110390, ind : 83, value : 41.5}
DEBUG {time_boot_ms : 111391, ind : 84, value : 42.0}
DEBUG {time_boot_ms : 112393, ind : 85, value : 42.5}
DEBUG {time_boot_ms : 113395, ind : 86, value : 43.0}
DEBUG {time_boot_ms : 114398, ind : 87, value : 43.5}
DEBUG {time_boot_ms : 115398, ind : 88, value : 44.0}
DEBUG {time_boot_ms : 116400, ind : 89, value : 44.5}
DEBUG {time_boot_ms : 117403, ind : 90, value : 45.0}
DEBUG {time_boot_ms : 118404, ind : 91, value : 45.5}
```

Project2 Part 2

For this part of the project, we were to establish the communication between the Raspberry PI and the FMU to interpret and react to incoming data. The RPI is connected to the ultrasonic sensors and cameras and outputs this data to the FMU using mavlink and uORB protocols.

Raspberry PI

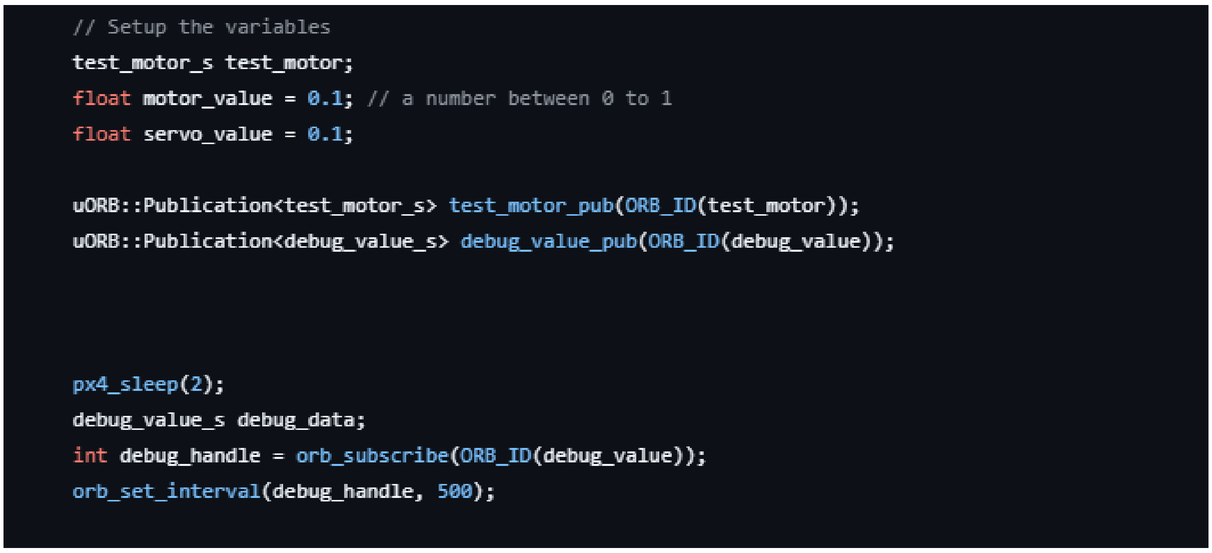


```
if __name__ == "__main__":


    conn = mavutil.mavlink_connection('/dev/ttyACM0')
    conn.wait_heartbeat()
    print("Heartbeat from system (system %u component %u" % (conn.target_system, conn.target_component))

    ultra.ultrasonic_setup();


    while(1):
        cam_ret = cam.process_camera_frame() # 0 L, 1 F, 2 R
        distance = ultra.distance()
        #print(f"Camera value: {cam_ret}\nUltra value : {distance}")
        message = mavutil.mavlink.MAVLink_debug_message(0, cam_ret, distance)
        conn.mav.send(message)
        print("Value Sent")
        time.sleep(0.4)
        debug_msg = conn.recv_match(type = 'DEBUG', blocking = True)
```

The main function of the raspberry pi first establishes the mavlink connection with the FMU. Once it has completed that it also completes the ultrasonic setup for the sensors. The function for the ultrasonic sensors was predefined in a previous step of the lab and we imported it in for this portion as well as the camera functions. We then enter the system loop where we interpret the camera frames using edge detection, going to the lowest average of the edge detection. We also retrieve the perceived distance from the ultrasonic sensors. This data is then passed into a debug message with the direction to proceed as the index value, and the distance set to the debug.value. We sleep for 0.4 seconds, and we complete this process until someone terminates the program. The FMU will now interpret these DEBUG messages.

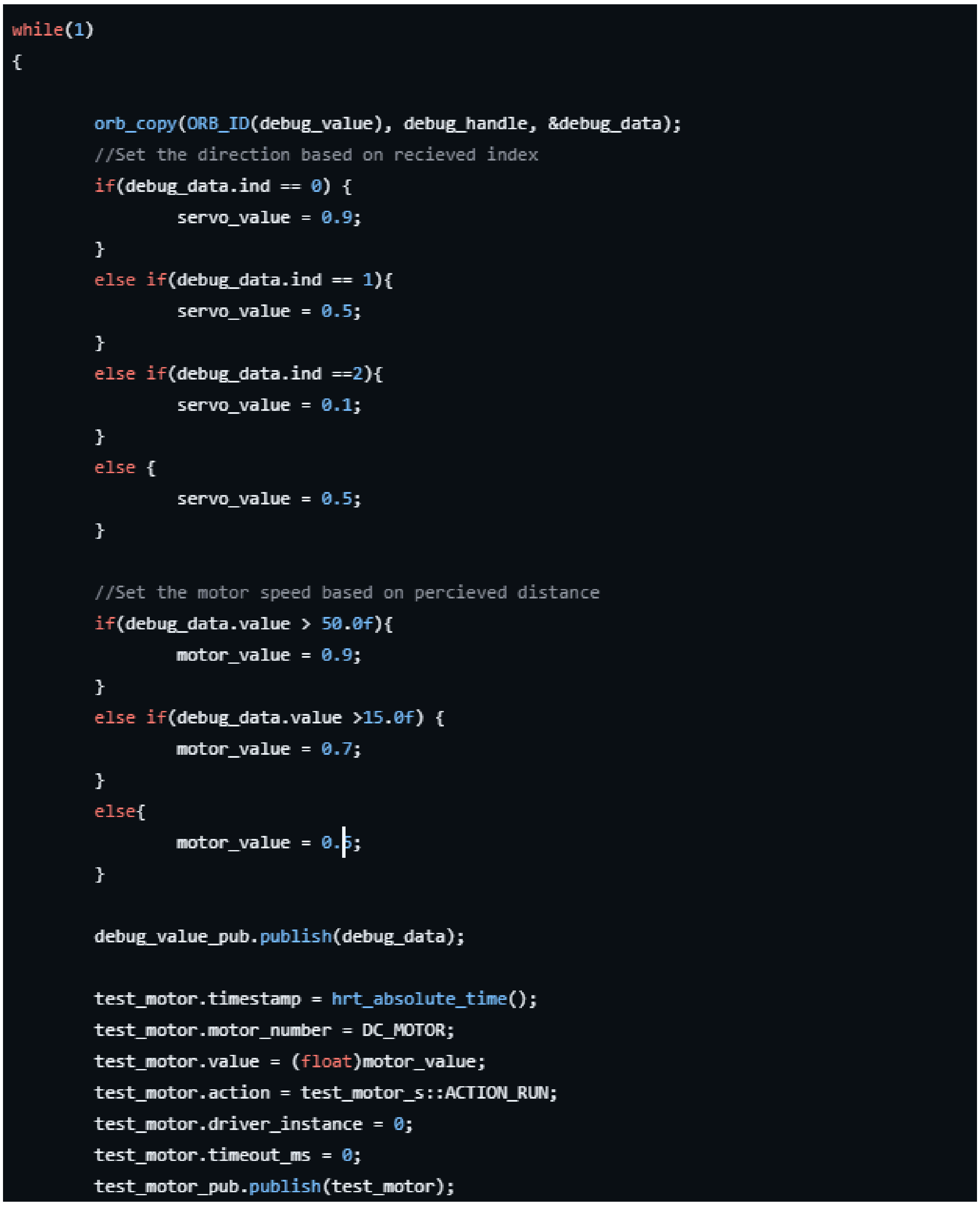
FMU

On the FMU end we first initialize some values for the motors. These will be altered in the system loop to determine the direction and speed of the car. We also create two publication uORB channels to publish values to the debug as well as the motors. The debug publishing may not be used in the final program as it was mostly used for debugging purposes. Finally we also create a uORB subscription to the debug values that will be handled in our system loop.

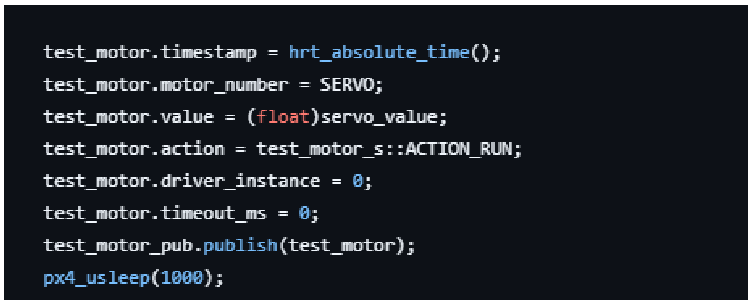


```
    // Setup the variables
    test_motor_s test_motor;
    float motor_value = 0.1; // a number between 0 to 1
    float servo_value = 0.1;

    uORB::Publication<test_motor_s> test_motor_pub(ORB_ID(test_motor));
    uORB::Publication<debug_value_s> debug_value_pub(ORB_ID(debug_value));



    px4_sleep(2);
    debug_value_s debug_data;
    int debug_handle = orb_subscribe(ORB_ID(debug_value));
    orb_set_interval(debug_handle, 500);
```

Our main system loop proceeds by first receiving the uORB message that it is subscribed to. For our implementation this debug message is formatted to have the Index as the direction value and the distance in the debug value. Based on the incoming set direction, we set the servo motor value to the corresponding value, 0 for left, 1 for straight and 2 for right. Based on the percieved distance we set a speed. At lower than 15cm, we completely stop, at below 50cm we go slightly faster, and above 50cm we go our max speed. These values our updated in our if statements. Then to actually commit the updates to the car we need to publish them to the uORB channel. We do this by setting the corresponding values and then committing a uORB publish. This is done for both the servo and the DC motor separately. After that our system loop is complete. This loop allows the incoming data to be interpreted by the PI, and communicated via mavlink to our FMU, enabling the car to be autonomous in nature.



```
while(1)
{

        orb_copy(ORB_ID(debug_value), debug_handle, &debug_data);
        //Set the direction based on recieved index
        if(debug_data.ind == 0) {
                servo_value = 0.9;
        }
        else if(debug_data.ind == 1){
                servo_value = 0.5;
        }
        else if(debug_data.ind ==2){
                servo_value = 0.1;
        }
        else {
                servo_value = 0.5;
        }

        //Set the motor speed based on percieved distance
        if(debug_data.value > 50.0f){
                motor_value = 0.9;
        }
        else if(debug_data.value >15.0f) {
                motor_value = 0.7;
        }
        else{
                motor_value = 0.5;
        }

        debug_value_pub.publish(debug_data);

        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);
```



```
test_motor.timestamp = hrt_absolute_time();
test_motor.motor_number = SERVO;
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);
px4_usleep(1000);
```