COMPENG 4DS4 Project 2 Report

Aaron Pinto
pintoa9

Raeed Hassan
hassam41

Jingming Liu
liuj171

Jeffrey Guo
guoj69

April 7, 2023

Declaration of Contributions

Member	Contributions
Raeed	Step 0, Part 1, Part 2 C++
Jingming	Part 1, Part 2 C++
Aaron	Part 2 Python
Jeffrey	VM Setup, Mavlink Setup and Debug

Experiment 2 Setup B: Hello World Application

The NuttX terminal output for a `hello_world` application is shown in Figure 1. The application running is Part 1 instead of the `hello_world` application provided in this experiment.

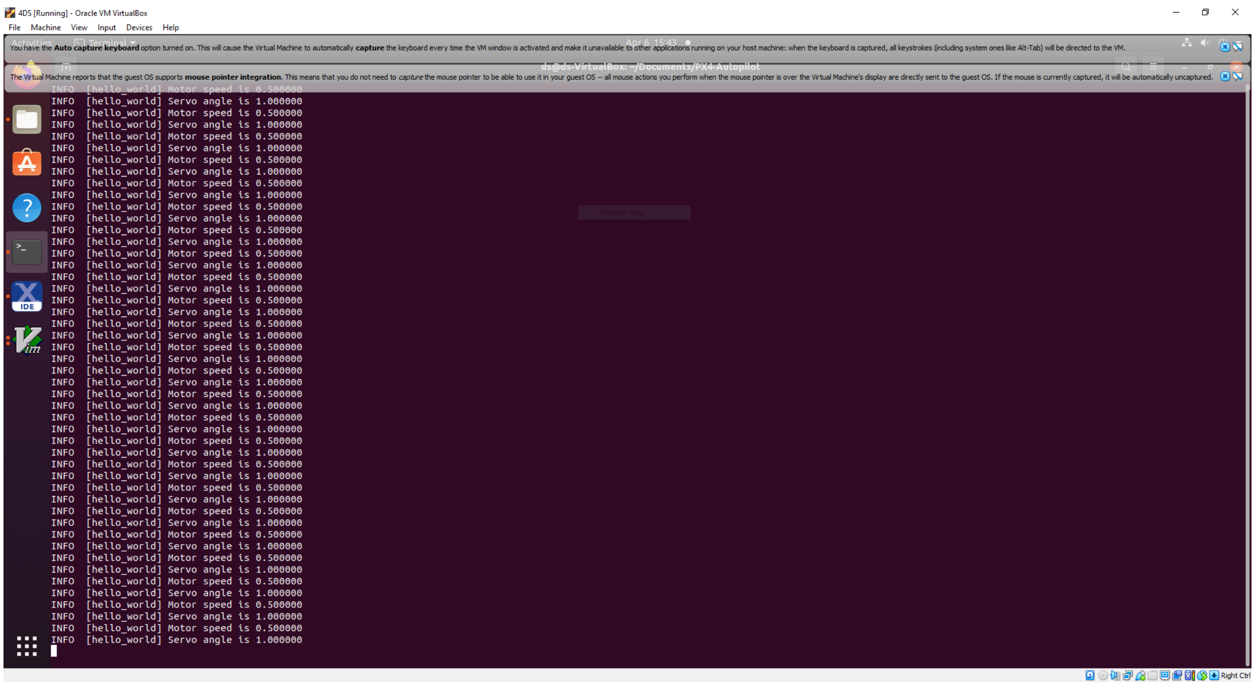

Figure 1: test

Experiment 3 Setup A: Interact with uORB

The output of reading the `sensor_gyro`, `vehicle_imu`, and `cpuload` messages are shown in Listing 1.

Listing 1: uORB Messages

```
nsh> listener sensor_gyro

TOPIC: sensor_gyro
sensor_gyro
    timestamp: 56179390 (0.000564 seconds ago)
    timestamp_sample: 56179349 (41 us before timestamp)
    device_id: 5505042 (Type: 0x54, SPI:2 (0x00))
    x: 0.0141
    y: 0.0043
    z: 0.0032
    temperature: 36.0000
    error_count: 0
    samples: 1

nsh> listener vehicle_imu

TOPIC: vehicle_imu
vehicle_imu
    timestamp: 65859731 (0.001847 seconds ago)
    timestamp_sample: 65858298 (1433 us before timestamp)
```

```
    accel_device_id: 5373970 (Type: 0x52, SPI:2 (0x00))
    gyro_device_id: 5505042 (Type: 0x54, SPI:2 (0x00))
    delta_angle: [-0.0000, 0.0000, 0.0000]
    delta_velocity: [0.0147, 0.0021, -0.0526]
    delta_angle_dt: 4995
    delta_velocity_dt: 5002
    delta_velocity_clipping: 0
    accel_calibration_count: 2
    gyro_calibration_count: 1

nsh> listener cpuload

TOPIC: cpuload
cpuload
    timestamp: 77823225 (0.188618 seconds ago)
    load: 0.6415
    ram_usage: 0.8147
```

Step 0

To complete this task, we modify the code provided in Experiment 3B. To read the values of the RC channels, we subscribe to `rc_channels` messages instead of `sensored_combined` messages. The `rc_channel_s` structure provided in `rc_channels` uORB topic directory contains the member `channels[18]`, which will contain the channel data. In this application, we access the data in these channels. The data contained in `rc_channel_data.channels` ranges from -1 to 1. The code for Step 0 is shown below in Listing 2.

Listing 2: Step 0 Code

```
#include <px4_platform_common/px4_config.h>
#include <px4_platform_common/log.h>

#include <uORB/topics/rc_channels.h>

extern "C" __EXPORT int hello_world_main(int argc, char *argv[]);

int hello_world_main(int argc, char *argv[])
{
    int rc_channel_handle;
    rc_channels_s rc_channel_data;

    rc_channel_handle = orb_subscribe(ORB_ID(rc_channels));
    orb_set_interval(rc_channel_handle, 200);

    while (1)
    {
        orb_copy(ORB_ID(rc_channels), rc_channel_handle, &rc_channel_data);

        PX4_INFO("header = %f, ch1 = %f, ch2 = %f, ch3 = %f, ch4 = %f, ch5 = %f, ch6 = %f,
            ch7 = %f, ch8 = %f",
                double(rc_channel_data.channels[0]),
                double(rc_channel_data.channels[1]),
                double(rc_channel_data.channels[2]),
                double(rc_channel_data.channels[3]),
                double(rc_channel_data.channels[4]),
                double(rc_channel_data.channels[5]),
                double(rc_channel_data.channels[6]),
                double(rc_channel_data.channels[7]),
                double(rc_channel_data.channels[8]));

        px4_usleep(200000);
    }
```

```

    return 0;
}
```

Part 1

To control the servo motor, we define the servo motor on pin 2 as motor number 1 for `test_motor`. The servo motor can then be controlled by duplicating the DC motor code. We can additionally integrate the RC channel code from step 0 to read the RC channel values and calculate the values to publish to the DC and servo motors.

We read channels 1 and 3 (mapped to the two joysticks on the RC controller) to control the motors. The data from the channels (ranging from -1 to 1), are mapped to 0 to 1 range required by `test_motor`.The RC controller channel 3 value is offset by 0.96 first and divided by 1.96, to normalize the range from 0 to 1 (based on the values we received from the controller in step 0), then divided by 2 to make the range from 0 to 0.5. In this way the input can be calibrated for both forward and backward motor values. For the servo motor angle, we divide by 2 to reduce the range from -0.5 to 0.5, and then add 0.5 to move the range from 0 to 1.

The code for Part 1 is shown below in Listing 3.

Listing 3: Part 1 Code

```
#include <px4_platform_common/px4_config.h>
#include <px4_platform_common/log.h>

#include <drivers/drv_hrt.h>
#include <uORB/Publication.hpp>
#include <uORB/topics/test_motor.h>
#include <uORB/topics/rc_channels.h>


#define DC_MOTOR 0
#define SERVO_MOTOR 1

extern "C" __EXPORT int hello_world_main(int argc, char *argv[]);

int hello_world_main(int argc, char *argv[])
{
    px4_sleep(2);

    test_motor_s test_motor;
    double motor_value_final = 0; // a number between 0 to 1

    test_motor_s servo_motor;
    double angle_value = 0; // a number between 0 to 1

    double rc_channel_handle;
    double run_direc;
    rc_channels_s rc_channel_data;
    rc_channel_handle = orb_subscribe(ORB_ID(rc_channels));
    orb_set_interval(rc_channel_handle, 500);

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

    while (1)
    {
        orb_copy(ORB_ID(rc_channels), rc_channel_handle, &rc_channel_data);
```

```

        run_direc = double(rc_channel_data.channels[4]);

        if (run_direc < 0.5){
        motor_value_final = (double(rc_channel_data.channels[3])  + 0.96)/1.96/2.0;
        } else {
        motor_value_final = 0.5 - (double(rc_channel_data.channels[3]) + 0.96)/1.96/2.0;
        }

        angle_value = double(rc_channel_data.channels[1])/2.0 + 0.5;

        PX4_INFO("Motor speed is %f", motor_value_final);
        test_motor.timestamp = hrt_absolute_time();
        test_motor.motor_number = DC_MOTOR;
        test_motor.value = (float)motor_value_final;
        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", angle_value);
        servo_motor.timestamp = hrt_absolute_time();
        servo_motor.motor_number = SERVO_MOTOR;
        servo_motor.value = (float)angle_value;
        servo_motor.action = test_motor_s::ACTION_RUN;
        servo_motor.driver_instance = 0;
        servo_motor.timeout_ms = 0;

        test_motor_pub.publish(servo_motor);

        px4_usleep(20000);
    }

    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);

    PX4_INFO("The servo motor will be stopped");
    servo_motor.timestamp = hrt_absolute_time();
    servo_motor.motor_number = SERVO_MOTOR;
    servo_motor.value = 0.5;
    servo_motor.driver_instance = 0;
    servo_motor.timeout_ms = 0;

    test_motor_pub.publish(servo_motor);

    return 0;
}
```

Part 2

The sensor data is read from a python program on the Raspberry Pi. The python code for Part 2 is shown in Listing 4. The python program reads data from the ultrasonic sensor and camera using the algorithms provided in Experiments 5 and 6, and establishes a mavlink connection to send debug messages to the FMU (as shown in Experiment 4). The distance value is sent in a debug message with an index of 1, and the preferred direction is sent in a debug message with an index of 0. The debug messages are sent separately through mavlink as the uORB structure in the C++ code only contains one debug message, requiring two reads.

Listing 4: Part 2 Python Code

```
#Libraries
from pymavlink import mavutil
import RPi.GPIO as GPIO
import time
import cv2
import numpy as np
import math

##########################
# Ultrasonic Sensor Setup #
##########################

#set GPIO Pins
GPIO_TRIGGER = 23
GPIO_ECHO = 24


def ultrasonic_setup():
    #GPIO Mode (BOARD / BCM)
    GPIO.setmode(GPIO.BCM)

    #set GPIO direction (IN / OUT)
    GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
    GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance

###############
# Camera Setup #
###############

cap = cv2.VideoCapture(0)
StepSize = 5

def getChunks(l, n):
    """Yield successive n-sized chunks from l."""
```

```
    a = []
    for i in range(0, len(l), n):
        a.append(l[i:i + n])
    return a

def process_camera_frame():
    ret,frame = cap.read()
    img = cv2.flip(frame, 0)
    blur = cv2.bilateralFilter(img,9,40,40)
    edges = cv2.Canny(blur,50,100)
    img_h = img.shape[0] - 1
    img_w = img.shape[1] - 1
    EdgeArray = []

    for j in range(0,img_w,StepSize):
        pixel = (j,0)
        for i in range(img_h-5,0,-1):
            if edges.item(i,j) == 255:
                pixel = (j,i)
                EdgeArray.append(pixel)
                break

    if len(EdgeArray) != 0:
        chunks = getChunks(EdgeArray, math.ceil(len(EdgeArray)/3))
    else:
        return

    #c = []
    distance = []
    for i in range(len(chunks)):
        x_vals = []
        y_vals = []
        for (x,y) in chunks[i]:
            x_vals.append(x)
            y_vals.append(y)
        avg_x = int(np.average(x_vals))
        avg_y = int(np.average(y_vals))
        #c.append([avg_y,avg_x])
        distance.append(math.sqrt((avg_x - 320)**2 + (avg_y - 640)**2))
        cv2.line(img, (320, 640), (avg_x,avg_y), (0,0,255), 2)

    cv2.imshow("frame", img)
    cv2.waitKey(5)
    if(distance[0] < distance[1]):
        if(distance[0] < distance[2]):
            return 0
        else:
            return 2
    else:
        if(distance[1] < distance[2]):
            return 1
        else:
            return 2

##################################################

ultrasonic_setup();

# 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))


```

```python
# Once connected, use 'the_connection' to get and send messages
value = 0

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

    message = mavutil.mavlink.MAVLink_debug_message(0, 0, dist)
    the_connection.mav.send(message)

    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')

    message = mavutil.mavlink.MAVLink_debug_message(0, 1, ret)
    the_connection.mav.send(message)

    time.sleep(0.1)
    # time.sleep(1)
    print("Message sent")
```

The motor actuation is handled in the C++ code using the code from Part 1. The C++ code for Part 2 is shown in Listing 5. The code is modified to read debug messages (as shown in Experiment 3C) instead of the RC values. When the debug messages are read by the C++ application, the index is used to determined if the data copied is the distance data or the preferred direction. The servo angle is set based on the preferred direction. The motor value is set based on the distance. The motor will stop if the distance is below 15cm, with operate at roughly 40% speed when between 15cm and 50cm, and at 100% speed above 50cm.

Listing 5: Part 2 C++ Code

```cpp
#include <px4_platform_common/px4_config.h>
#include <px4_platform_common/log.h>

#include <drivers/drv_hrt.h>
#include <uORB/Publication.hpp>
#include <uORB/topics/test_motor.h>
#include <uORB/topics/debug_value.h>

#define DC_MOTOR 0
#define SERVO_MOTOR 1

extern "C" __EXPORT int hello_world_main(int argc, char *argv[]);

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);

    test_motor_s test_motor;
    double motor_value = 0; // a number between 0 to 1

    test_motor_s servo_motor;
    double angle_value = 0.5; // a number between 0 to 1

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

```

```
    int dist = 0;
    int ret = 0;

    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", angle_value);
    servo_motor.timestamp = hrt_absolute_time();
    servo_motor.motor_number = SERVO_MOTOR;
    servo_motor.value = (float)angle_value;
    servo_motor.action = test_motor_s::ACTION_RUN;
    servo_motor.driver_instance = 0;
    servo_motor.timeout_ms = 0;

    test_motor_pub.publish(servo_motor);

    while (1)
    {
        orb_copy(ORB_ID(debug_value), debug_handle, &debug_data);

        //get motor and angle values from rc_data
        if (debug_data.ind == 0) dist = debug_data.value;
        if (debug_data.ind == 1) ret = debug_data.value;

        // do something with dist and ret
        if (dist >= 50) {
            motor_value = 1;
        } else if ( dist >= 15 && dist < 50 ) {
            motor_value = 0.7;
        } else if (dist < 15) {
            motor_value = 0.5;
        }

        if (ret == 0) {
            angle_value = 1;
        } else if(ret == 1){
            angle_value = 0.5;
        } else if(ret == 2){
            angle_value = 0;
        }

        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", angle_value);
        servo_motor.timestamp = hrt_absolute_time();
        servo_motor.motor_number = SERVO_MOTOR;
        servo_motor.value = (float)angle_value;
        servo_motor.action = test_motor_s::ACTION_RUN;
        servo_motor.driver_instance = 0;
        servo_motor.timeout_ms = 0;

        test_motor_pub.publish(servo_motor);

```

```
        px4_usleep(20000);
    }

    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);

    PX4_INFO("The servo motor will be stopped");
    servo_motor.timestamp = hrt_absolute_time();
    servo_motor.motor_number = SERVO_MOTOR;
    servo_motor.value = 0.5;
    servo_motor.driver_instance = 0;
    servo_motor.timeout_ms = 0;

    test_motor_pub.publish(servo_motor);


    return 0;
}
```