CPE476 – Mobile Robotics

Midterm 2

Name: Neel Patel

Email: pateln3@unlv.nevada.edu

Github Repository link (root): https://github.com/neelpatel114/submissions

Youtube Playlist link (root): https://youtube.com/playlist?list=PLjhbM6__bgV_OnArIwnmPu7-PxkAD8iHn

Overview:

In this midterm assignment we were tasked with multiple objective. Primary all the tasks were relevant to completely setting up the jetson nano with a ROS environment and having it communicate with the teensy. First we were tasked with installing ROS on the nano. Then I developed a ROS package to accept data from the Teensy. Then did a similar process to the Teensy package but this time it was for the BNO055 IMU. We then used the data gathered from these two devices to find the robots local position and help it navigate.

Picture of Robot:

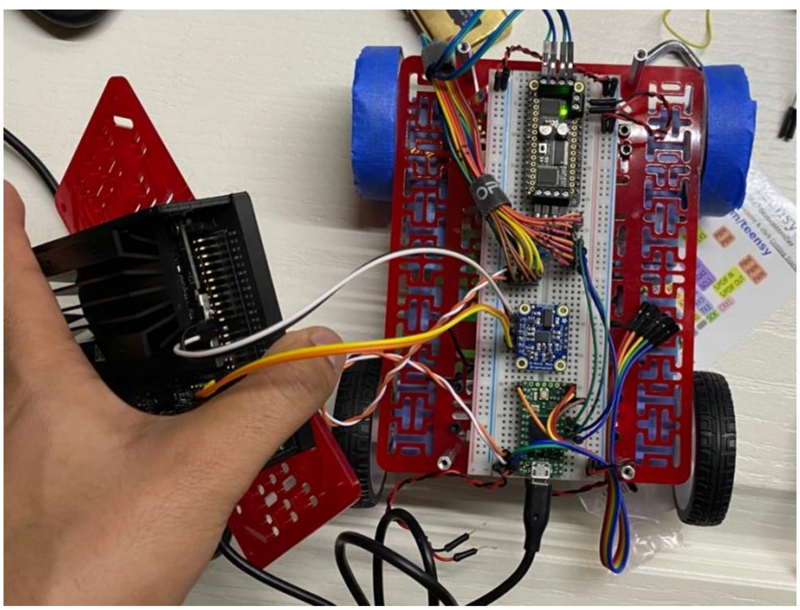

Figure 1 Picture of ROS implemented Robot

Tasks:

To implement ROS onto the board I installed jetson image of linux with a GUI. I did not do the headless version because I needed to install teensyduino and im not certain if that is possible using the command line. After the image was installed I connected it to the internet and RDPed into the jetson. This allowed to easily access the jetson without a monitor while it is connected to its own power supply on the robot. I installed Arduino and then Teensyduino to allow the jetson to push code to the teensy. This is important for the jetson to read the wheel encoder data. After installing Teensyduino I installed ros_serial which allows the ROS system to communicate with arduinos. Due to the Teensy not being an actual Arduino there had to be some adjustments to the Arduino_hardware.h file so it could read and write properly. After all the ROS files to communicate with the teensy were implemented I tested it with a blinky and hello world test code. After these confirmed the communication using the correct serial port and baud rate I implemented the code to read the encoder data and output the tics, rpm, and velocity to the ROS serial terminal. I was able to read the data using the rostopic echo /blank command where blank would be a library the code wrote to. After this was complete I began working on the IMU implementation. To do this I had to connect the IMU VCC, GND, SDA, and SCL ports to the 3.3V, GND, 5, and 3 ports on the jetson nano. After this was complete I could read the hardware address as 28 and read the data. Now it was time to implement the odometer data with the IMU so enable to robot to have motion. I was unable to get that function in time for this report.

Code:

```
#include <ros.h>
#include <std_msgs/Int16.h>

// Motor encoder output pulses per 360 degree revolution (measured manually)
#define ENC_COUNT_REV 136

// Handles startup and shutdown of ROS
ros::NodeHandle nh;

// Encoder output to Arduino Interrupt pin. Tracks the tick count.
#define ENC_IN_LEFT_A 2
#define ENC_IN_RIGHT_A 3

// Other encoder output to Arduino to keep track of wheel direction
// Tracks the direction of rotation.
#define ENC_IN_LEFT_B 4
#define ENC_IN_RIGHT_B 11

// True = Forward; False = Reverse
```

```
boolean Direction_left = true;
boolean Direction_right = true;

// Minumum and maximum values for 16-bit integers
const int encoder_minimum = -32768;
const int encoder_maximum = 32767;

// Keep track of the number of right wheel pulses
volatile long right_wheel_pulse_count = 0;

// Keep track of the number of wheel ticks
std_msgs::Int16 right_wheel_tick_count;
ros::Publisher rightPub("right_ticks", &right_wheel_tick_count);

std_msgs::Int16 left_wheel_tick_count;
ros::Publisher leftPub("left_ticks", &left_wheel_tick_count);

std_msgs::Int16 rpm_right;
ros::Publisher rpm_rightPub("RPM", &rpm_right);

std_msgs::Int16 ang_velocity_right;
ros::Publisher ang_velocity_rightPub("Ang Velocity R", &ang_velocity_right);

std_msgs::Int16 ang_velocity_right_deg;
ros::Publisher ang_velocity_degreePub("And Velocity Deg R", &ang_velocity_right_deg);


// 100ms interval for measurements
const int interval = 100;
long previousMillis = 0;
long currentMillis = 0;




const float rpm_to_radians = 0.10471975512;
const float rad_to_deg = 57.29578;
```

```
// Increment the number of ticks
void right_wheel_tick() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_IN_RIGHT_B);

  if(val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }

  if (Direction_right) {

    if (right_wheel_tick_count.data == encoder_maximum) {
      right_wheel_tick_count.data = encoder_minimum;
    }
    else {
      right_wheel_tick_count.data++;
    }
  }
  else {
    if (right_wheel_tick_count.data == encoder_minimum) {
      right_wheel_tick_count.data = encoder_maximum;
    }
    else {
      right_wheel_tick_count.data--;
    }
  }
}

// Increment the number of ticks
void left_wheel_tick() {

  // Read the value for the encoder for the left wheel
```

```
  int val = digitalRead(ENC_IN_LEFT_B);

  if(val == LOW) {
    Direction_left = true; // Reverse
  }
  else {
    Direction_left = false; // Forward
  }

  if (Direction_left) {
    if (left_wheel_tick_count.data == encoder_maximum) {
      left_wheel_tick_count.data = encoder_minimum;
    }
    else {
      left_wheel_tick_count.data++;
    }
  }
  else {
    if (left_wheel_tick_count.data == encoder_minimum) {
      left_wheel_tick_count.data = encoder_maximum;
    }
    else {
      left_wheel_tick_count.data--;
    }
  }
}

void setup() {

  // Set pin states of the encoder
  pinMode(ENC_IN_LEFT_A , INPUT_PULLUP);
  pinMode(ENC_IN_LEFT_B , INPUT);
  pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_B , INPUT);

  // Every time the pin goes high, this is a tick
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A), left_wheel_tick, RISING);
```

```
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A), right_wheel_tick, RISING);

  // ROS Setup
  nh.getHardware()->setBaud(115200);
  nh.initNode();
  nh.advertise(rightPub);
  nh.advertise(leftPub);
  nh.advertise(rpm_rightPub);
  nh.advertise(ang_velocity_rightPub);
  nh.advertise(ang_velocity_degreePub);
}

void loop() {

  // Record the time
  currentMillis = millis();

  // If 100ms have passed, print the number of ticks
  if (currentMillis - previousMillis > interval) {

    previousMillis = currentMillis;
    // Calculate revolutions per minute
    rpm_right.data = (float)(right_wheel_pulse_count * 60 / ENC_COUNT_REV);
    ang_velocity_right.data = rpm_right.data * rpm_to_radians;
    ang_velocity_right_deg.data = ang_velocity_right.data * rad_to_deg;

    rightPub.publish( &right_wheel_tick_count );
    leftPub.publish( &left_wheel_tick_count );
    rpm_rightPub.publish(&rpm_right);
    ang_velocity_rightPub.publish(&ang_velocity_right);
    ang_velocity_degreePub.publish(&ang_velocity_right_deg);

    right_wheel_pulse_count = 0;
    nh.spinOnce();
  }
}
```

Other Pictures:

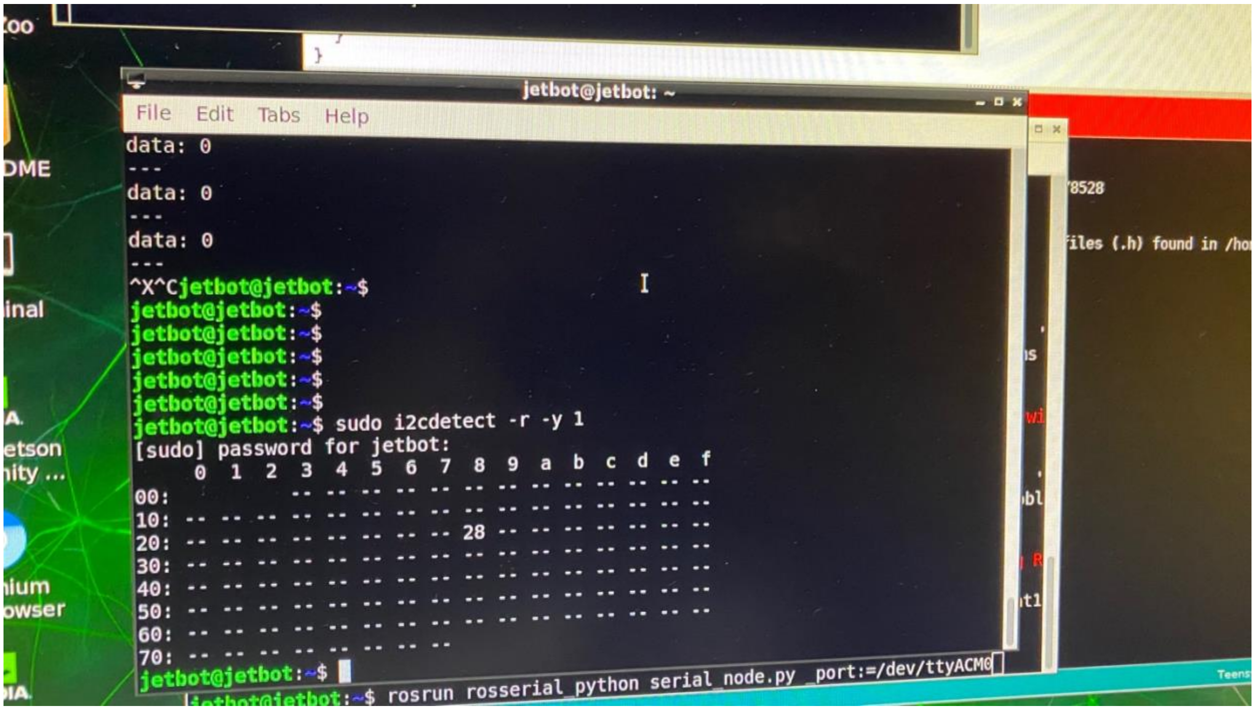

Figure 2 Confirmation of communication between nano and IMU

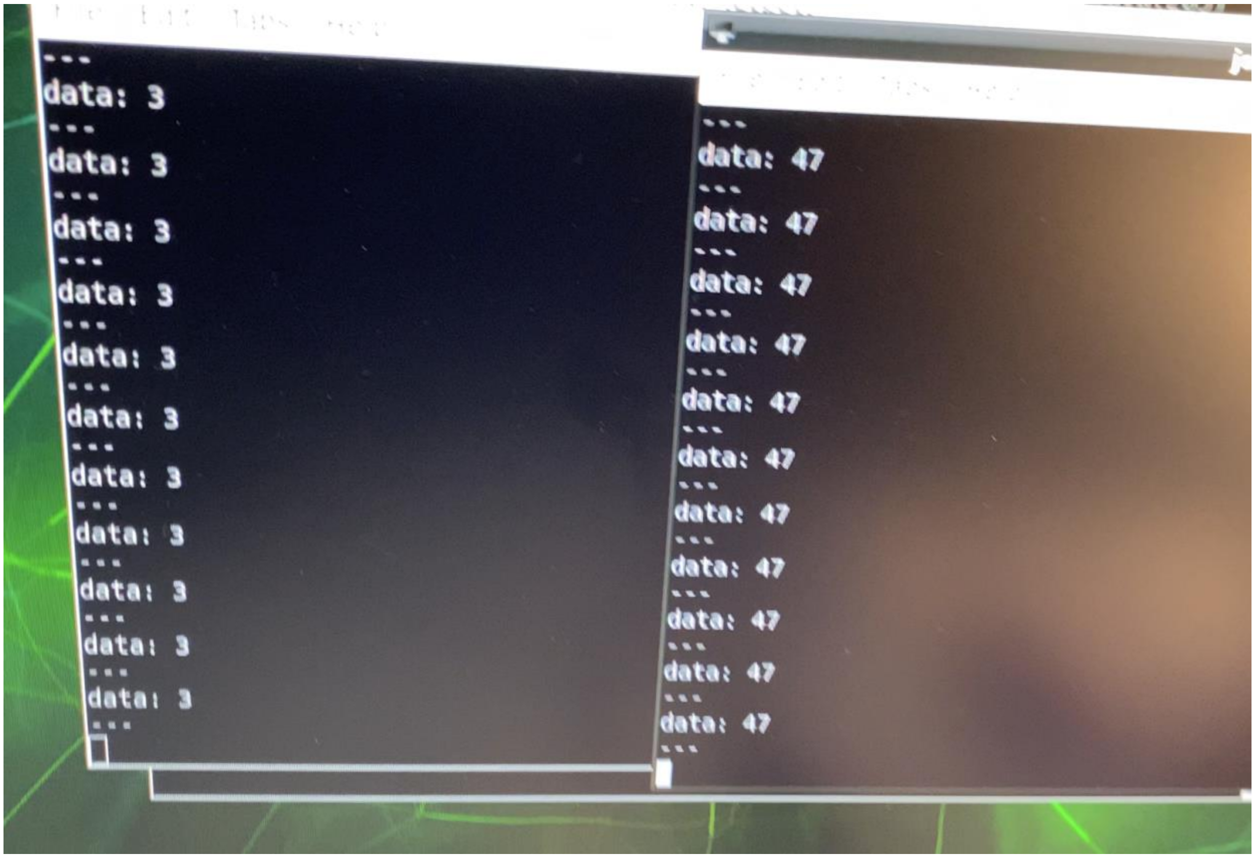

Figure 3 Wheel encoder data being read by ROS

I understand the Student Academic Misconduct Policy - http://studentconduct.unlv.edu/misconduct/policy.html

“This assignment submission is my own, original work”.
Neel Patel