

CPE476 – FALL 2019

Final Report

Student Names: Jacob Patrick Reed & Cody McDonald
Student Emails: reedj35@unlv.nevada.edu & mcdonc4@unlv.nevada.edu
Primary Github address: <https://github.com/reedjacobp>
Directory: <https://github.com/reedjacobp/WMR>

1. Introduction

The main goal for this semester was to build a wheel mobile robot that could maneuver from point A to point B while avoiding obstacles. Over the course of the semester, several difficulties were encountered, and the main goal became being able to interface the robot with a Raspberry Pi in order to send commands through ROS to control the robot. For our robot, we were able to accomplish several tasks being: travel in a straight line, travel in the shape of a square bidirectionally, PID control, interface the IMU, and control the robot's movements wirelessly.

2. Tasks

Straight Line

In order to travel in a straight line, we had to write code for the Romi that commanded the motors to move at the same speed. Since each motor has different physical characteristics, they both do not move at the same speed if commanded the same speed value. We used encoder feedback which measured the encoder tick count of each encoder and would compare them to see which one was less than the other. Whichever motor had a smaller encoder tick count is the one we would command to travel faster. These measurements were made as fast as the encoder could transmit and was fast enough to ensure that the robot traveled in a straight line. The code can be found below.



```

#include <Romi32U4.h>

Romi32U4Encoders encoders;
Romi32U4Motors motors;
Romi32U4ButtonA buttonA;

void setup()
{
  buttonA.waitForButton();
  delay(1000);
}

void loop()
{
  // Run left and right motor forward.
  motors.setSpeeds(100, 100);
  if (encoders.getCountsLeft() > encoders.getCountsRight())
  {
    // increase speed of right motor momentarily to correct for faster left motor
    motors.setSpeeds(100, 100*4);
  }
  else if (encoders.getCountsLeft() < encoders.getCountsRight())
  {
    // increase speed of left motor momentarily
    motors.setSpeeds(100*4, 100);
  }
}

```

Bidirectional Square

In order to create a square, we found the trim values for both motors and wrote code for the robot to travel in a straight line for a specific amount of time, stopped, reversed the right motor, stopped, forward again, and then repeat in this order until the square was completed. We then did the same as above in reverse to go in the other direction. This code did not use encoder feedback and can be found below.



```
// This was tested on hardwood floor.  
//  
// You can press button A on the Romi to make the box  
//
```

```
#include <Romi32U4.h>
```

```
Romi32U4Encoders encoders;  
Romi32U4Motors motors;  
Romi32U4ButtonA buttonA;
```

```
void setup()  
{  
}
```

```
void loop()  
{  
  if(buttonA.isPressed())  
  {  
    delay(1000);
```

```
////////////////////////////////////  
//   BEGIN FORWARD BOX   //  
////////////////////////////////////
```

```

// Run left and right motor forward.
motors.setSpeeds(100, 101);
delay(1375); //To travel 1 foot forward
motors.setSpeeds(0,0);
delay(1000);

////////////////////////////////////

//      END OF FIRST SIDE      //

////////////////////////////////////

////////////////////////////////////
// From here, the radius of the robot is 2.875 in //
// To make a full rotation, the wheel must travel  $2\pi r = 6\pi = 18.064\text{in}$  //
// To make a 90 degree rotation ( $360/4$ ) then the wheel must travel //
//  $18.064\text{in}/4 = 4.516\text{ in.}$  //
// since it takes 1375ms to travel 12 in,  $1375\text{ms}/12\text{in} = 114.583\text{ ms/in}$  //
// So to travel 4.516in,  $(114.583\text{ ms/in}) \times (4.516\text{in}) = 517.4568\text{ms}$  //
// The full equation is  $((\pi/2) \times r) \times (\text{time it takes to move 1 foot})$  //
// where r is the radius of the robot. //
// *It looks as if theres a little error with the above time. 490ms is good //
////////////////////////////////////

////////////////////////////////////
//      BEGIN SECOND SIDE      //

////////////////////////////////////

motors.setSpeeds(-101, 100); //left rotation
delay(515);
motors.setSpeeds(0,0);
delay(1000);

motors.setSpeeds(100, 101); //forward 1 foot
delay(1375);
motors.setSpeeds(0,0);
delay(1000);

////////////////////////////////////

//      END OF SECOND SIDE      //

////////////////////////////////////

////////////////////////////////////
//      BEGIN THIRD SIDE      //

////////////////////////////////////

motors.setSpeeds(-101, 100); //left rotation
delay(515);
motors.setSpeeds(0,0);
delay(1000);

```

```

motors.setSpeeds(100, 101); //forward 1 foot
delay(1375);
motors.setSpeeds(0,0);
delay(1000);
////////////////////////////////////
//      END OF THIRD SIDE      //
////////////////////////////////////

////////////////////////////////////
//      BEGIN FOURTH SIDE      //
////////////////////////////////////
motors.setSpeeds(-101, 100); //left rotation
delay(515);
motors.setSpeeds(0,0);
delay(1000);
motors.setSpeeds(100, 101); //forward 1 foot
delay(1375);
motors.setSpeeds(0,0);
delay(1000);
motors.setSpeeds(-101, 100); //left rotation
delay(515);
motors.setSpeeds(0,0);
delay(1000);
////////////////////////////////////
//      END OF FORWARD BOX    //
////////////////////////////////////

////////////////////////////////////
//      BEGIN REVERSE BOX      //
////////////////////////////////////
motors.setSpeeds(101, -100); //right rotation
delay(525);
motors.setSpeeds(0,0);
delay(1000);
motors.setSpeeds(-99, -102); //backward 1 foot
delay(1375);
motors.setSpeeds(0,0);
delay(1000);
////////////////////////////////////
//      END OF FIRST SIDE      //
////////////////////////////////////

////////////////////////////////////

```

```

//      BEGIN SECOND SIDE      //
////////////////////////////////////
motors.setSpeeds(101, -100); //right rotation
delay(525);
motors.setSpeeds(0,0);
delay(1000);
motors.setSpeeds(-99, -102); //backward 1 foot
delay(1375);
motors.setSpeeds(0,0);
delay(1000);
////////////////////////////////////
//      END OF SECOND SIDE      //
////////////////////////////////////

////////////////////////////////////
//      BEGIN THIRD SIDE      //
////////////////////////////////////
motors.setSpeeds(101, -100); //right rotation
delay(525);
motors.setSpeeds(0,0);
delay(1000);
motors.setSpeeds(-99, -102); //backward 1 foot
delay(1375);
motors.setSpeeds(0,0);
delay(1000);
////////////////////////////////////
//      END OF THIRD SIDE      //
////////////////////////////////////

////////////////////////////////////
//      BEGIN FOURTH SIDE      //
////////////////////////////////////
motors.setSpeeds(101, -100); //right rotation
delay(525);
motors.setSpeeds(0,0);
delay(1000);
motors.setSpeeds(-99, -102); //backward 1 foot
delay(1375);
motors.setSpeeds(0,0);
delay(1000);
////////////////////////////////////
//      END OF REVERSE BOX      //
////////////////////////////////////

```

```
}  
}
```

PID Control

For PID control, we used the Arduino sketch given to us in class. We did not have to change the Kp, Ki, and Kd values for the robot to travel in a straight line. The main sketch can be found below. Please note the header files that were used can be found in our Github.

```
#include  
<Romi32U4.h>  
  
#include <PololuRPISlave.h>  
  
Romi32U4Motors motors;  
Romi32U4Encoders encoders;  
Romi32U4ButtonA buttonA;  
  
void setup() {  
  Serial.begin(57600);  
  
  // put your setup code here, to run once:  
  ledYellow(false);  
  ledGreen(true);  
  ledRed(false);  
}  
  
float _debug_linear_ms = 0.25;  
float _debug_angle_rs = 0.0;  
void _DEBUG_PID_CONTROL() {  
  static float _linear_ms_change = 0.1;  
  
}  
  
void loop() {  
  
  set_twist_target(_debug_linear_ms, _debug_angle_rs);  
  // put your main code here, to run repeatedly:  
  if (everyNmillisec(10)) {  
    // ODOMETRY  
    calculateOdom();  
    doPID();  
  }  
}
```

```
}
```

PID & IMU Interface

We tried to perform PID control using feedback from the IMU's gyro and accelerometer, but the robot's motion was jerky and inefficient. This is due to excessive noise affecting the IMU measurements. To filter out the noise, we tried using an averaging filter which would use the 5 previous measurements and average out to a value which was used as feedback. This did not work well, so we just used encoder tick counts. The code can be found below.

```
int
G;

    int vel_left;
    int vel_right;
    int countsLeft;
    int countsRight;
    int G1 = 0;
    int G2 = 0;
    int G3 = 0;
    int G4 = 0;
    int G5 = 0;
    int K = 10000;

#include <Wire.h>
#include <LSM6.h>
#include <Romi32U4.h>
#include <PIDController.h>

LSM6 imu;
Romi32U4Motors motors;
Romi32U4ButtonA buttonA;
Romi32U4Encoders encoders;
PIDController pid_left, pid_right;

void setup()
{
    Serial.begin(9600);
    pid_left.begin();
    pid_left.setpoint(0);
    pid_left.tune(0, 0, 0);
    pid_left.limit(0,100);

    pid_right.begin();
    pid_right.setpoint(0);
```



```

pid_right.tune(0, 0, 0);
pid_right.limit(-100,0);
Wire.begin();
if (!imu.init())
{
    ledRed(1);
    while (1)
    {
        Serial.println(F("Failed to detect the LSM6."));
        delay(100);
    }
}

imu.enableDefault();

imu.writeReg(LSM6::CTRL2_G, 0b10001000);

imu.writeReg(LSM6::CTRL1_XL, 0b10000100);

buttonA.waitForButton();
delay(2000);
}

void loop()
{
    imu.read();
    countsLeft = encoders.getCountsLeft();
    countsRight = encoders.getCountsRight();
    // 5 frame averager filter to filter sensor noise
    G5 = G4;
    G4 = G3;
    G3 = G2;
    G2 = G1;
    G1 = (int)imu.g.z;
    G = (G1 + G2 + G3 + G4 + G5)/5;

    int vel_left = 100 - pid_left.compute(G);
    int vel_right = 100 + pid_right.compute(G);

    // K is the gain of the controller
    // vel_left = 100 - G;
    // vel_right = 100 + G;
    motors.setSpeeds((int)vel_left, (int)vel_right);
}

```

```
// if (countsLeft > countsRight)
// {
//   motors.setSpeeds((int)vel_left, (int)vel_right*4);
// }
// else if (countsLeft < countsRight)
// {
//   motors.setSpeeds((int)vel_left*4, (int)vel_right);
// }

delay(50); //Run at 20Hz
}
```

Wireless Control via Raspberry Pi

To control the robot wirelessly, we needed the following components:

- Romi
- Raspberry Pi 3
- PC
- Access Point
- FTDI



To better describe how to achieve wireless control of the robot, the following steps must be followed:

Prerequisite Steps:

1. Ensure Raspberry Pi is set up to connect to a local access point that will be the same SSID that is connected to the Remote PC.

For us, we are connecting to our TP-Link Access Point with the following information:

SSID: BusBot_2.4GHz

Password: busbotbaby

2. Raspberry Pi must be connected to the Romi board via the header pins made specifically for it. In order to use the "teleop" feature from ROS however, we are not communicating via I2C, see next step for more information.
3. Insert a FTDI converter to a Raspberry Pi USB port and connect the TX & RX pins on the FTDI to the TX & RX pins on the Romi.
4. If needed, connect a monitor to the Raspberry Pi, open a terminal, and run:

```
ifconfig
```

And look at the wlan0 IP address to see the IP address you need to connect to for SSH.

5. On Remote PC, SSH into the Raspberry Pi with the following commands:

```
ssh <computername>@<IPaddress>
```

In our case the command looks like this:

```
ssh ubuntu@192.168.0.169
```

Main Steps:

1. Open the "motor_control_tele.cpp" file located in CpE476_demos/ROS_tutorials/arduino_bridge_tests/src and copy all of the code.
2. Open the Arduino IDE and start a new sketch. Paste the copied code and replace the line of code at the top that says "#include <ArduinoHardware.h>" with "#include <Romi32U4.h>"
3. Upload the sketch to the Romi and disconnect the cable.
4. On the Remote PC, open three terminal windows and SSH into the Raspberry Pi in each one.

5. Run the following commands, one in each window:

```
$ rosrunc rosserial_python serial_node.py /dev/ttyUSB0  
$ rosrunc teleop_twist_keyboard teleop_twist_keyboard.py
```

In the third window, you can see what is being published by running:

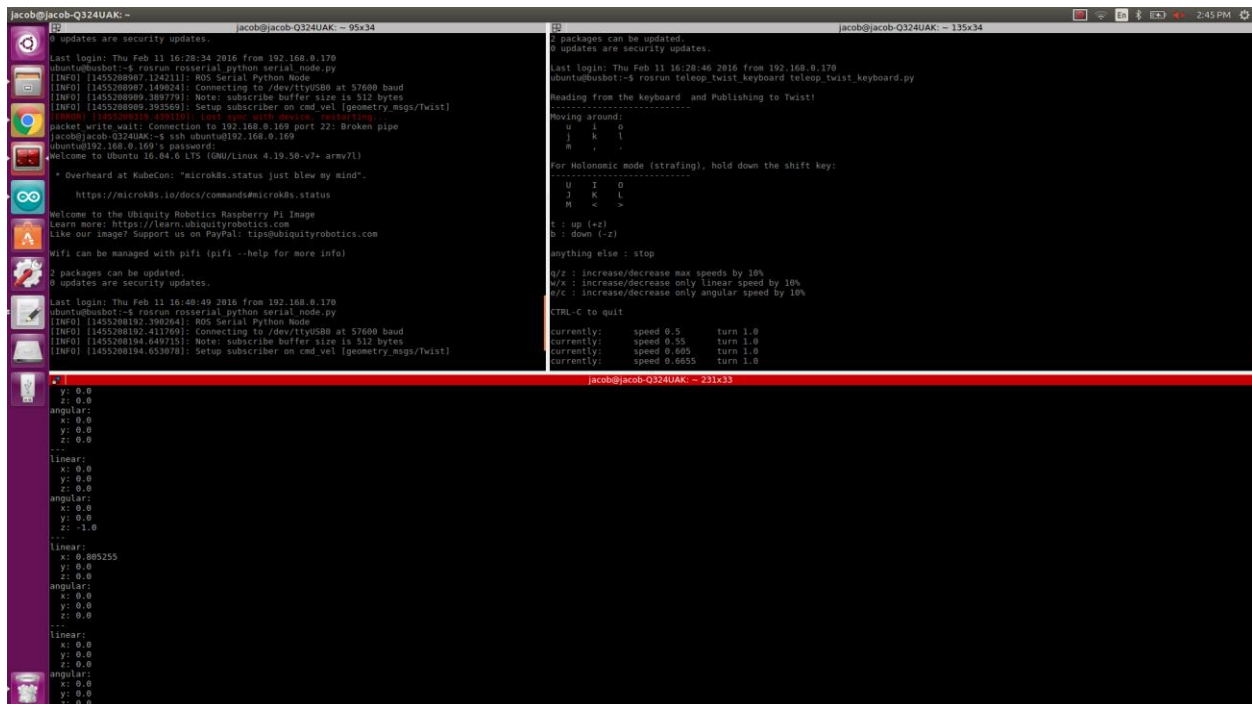
```
$ rostopic list
```

In this list, we see `/cmd_vel` and we can see what that is publishing by running:

```
$ rostopic echo /cmd_vel
```

This is interesting to see because we can observe what the linear velocities and angular velocities are being commanded to the robot at any given time from the "teleop" command in ROS.

6. Control the robot in the terminal running "teleop" and you can control the linear and angular velocities for both wheels.



The image shows three terminal windows on a Raspberry Pi. The left window shows the ROS environment setup with `rosserial_python` and `serial_node.py` running on `/dev/ttyUSB0`. The middle window shows the `teleop_twist_keyboard.py` running, displaying a control interface with keys for movement (U, I, O, J, K, L, M, <, >) and speed adjustments (q/z, w/x, v/c). The right window shows the output of `rostopic echo /cmd_vel`, displaying a stream of `geometry_msgs/Twist` messages with linear and angular velocity values.

Since there are many sketches, scripts, and header files that this task depends on, we will not be pasting the code in this report. The code can be found at the Github linked at the beginning of the report.

3. Conclusion

To conclude, this class was rewarding for us to learn how to build an autonomous robot. We may not have been able to fully complete the main task, but just learning how to use ROS could a course all on its own. We received a basic understanding of how ROS can be used to integrate all of components and code in one workspace so that the robot can function exactly how the engineer wants it to. It can be very difficult learning all the different commands that can be utilized with ROS as well as how to synthesize scripts and sketches to create a working product.

4. Video Links of each Demo

Straight Line: <https://youtu.be/MMFMRqzkSn4>

Bidirectional Square: <https://youtu.be/qQlm8Cx6fz8>

Screen Recording of teleop how-to: <https://youtu.be/ad7WaATWuOw>

Robot Operation using teleop: <https://youtu.be/kSMUKFLbJQo>