Class:	**CPE 476 Mobile Robotics**	Semester:	**Fall 2019**
Points	Document author:	**Jett Guerrero** **David Flores** **Ivan Soto**	
	Author's email:	guerrj1@unlv.nevada.edu sotoi2@unlv.nevada.edu flored@unlv.nevada.edu	
	Document topic:	**Final Report**	
Instructor's comments:			

Final Project Video: https://youtu.be/aYKLrKXD32Q

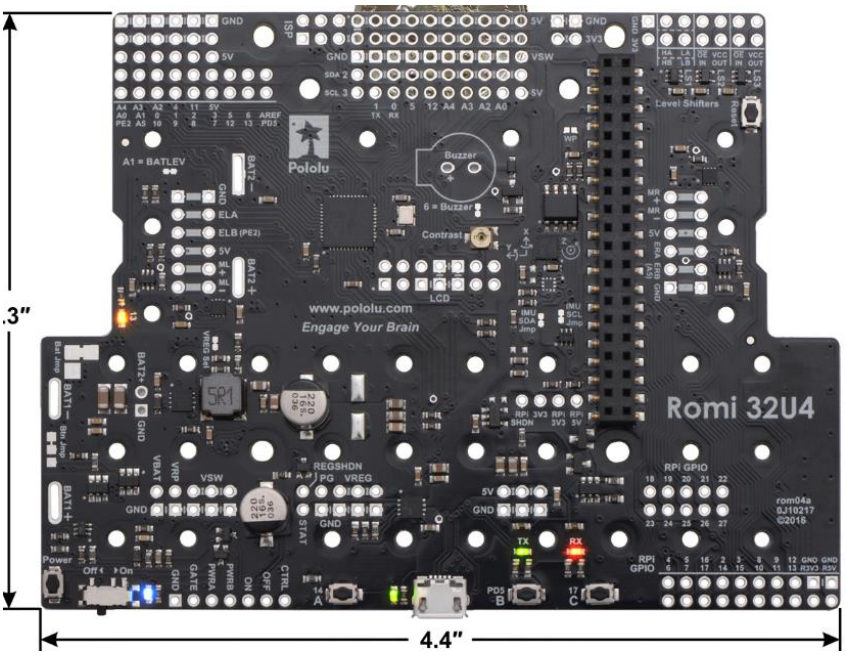
Mobile Robot Specifications:

Light Duty Robot: (Option 2)

- Robot Chassis: Romi Chassis Kit
- Motor Controller: Romi 32U4 Control Board
- Encoders: Romi Encoder Pair Kit, 12 CPR, 3.5-18V
- Motor Drivers: Builtin with Romi 32U4 Control Board
- IMU: Builtin with Romi 32U4 Control Board + LSM303D or MinIMU-9
- High-level Processing Engine/Controller: Rpi 3/4
- LIDAR Scanner: RPlidar A1 Scanners, YDLIDAR
- RGB Camera: Rpi Camera



Romi 32U4 Control Board:



General Specifications:

Size:	4.4″ × 3.3″ × 0.38″[1]
Weight:	35 g[1]
Current rating:	5 A[2]
Processor:	ATmega32U4 @ 16 MHz
RAM size:	2560 bytes
Program memory size:	32 Kbytes[3]
Motor driver:	DRV8838
Motor channels:	2
Minimum operating voltage:	2.5 V
Maximum operating voltage:	10.8 V
Continuous output current per channel:	1.8 A
Logic voltage:	5 V
Reverse voltage protection?:	Y
External programmer required?:	N

Motor and Encoder Testing:

Testing for straight path:

```
#include <Romi32U4.h>

Romi32U4Motors motors;
Romi32U4ButtonA buttonA;
Romi32U4Encoders encoders;

void setup()
{
  // Wait for the user to press button A.
  buttonA.waitForButton();

  // Delay so that the robot does not move away while the user is
  // still touching it.
  delay(1000);
}

void loop()
{
  // Run left and right motor forward.
 //  motors.setSpeeds(200,183);

  int16_t countsLeft = encoders.getCountsLeft();
  int16_t countsRight = encoders.getCountsRight();


//  for(int speed = 0; speed <=200; speed++)
//  {
          motors.setSpeeds(100, 100);
          if(countsLeft < countsRight)
          {
          motors.setSpeeds(100*4, 100);
          }

          else if(countsLeft > countsRight)
          {
          motors.setSpeeds(100, 100*4);
          }
//        else
//        {
//      motors.setLeftSpeed(speed);
//      motors.setRightSpeed(speed);
//        }
```

```
//  }

}
```

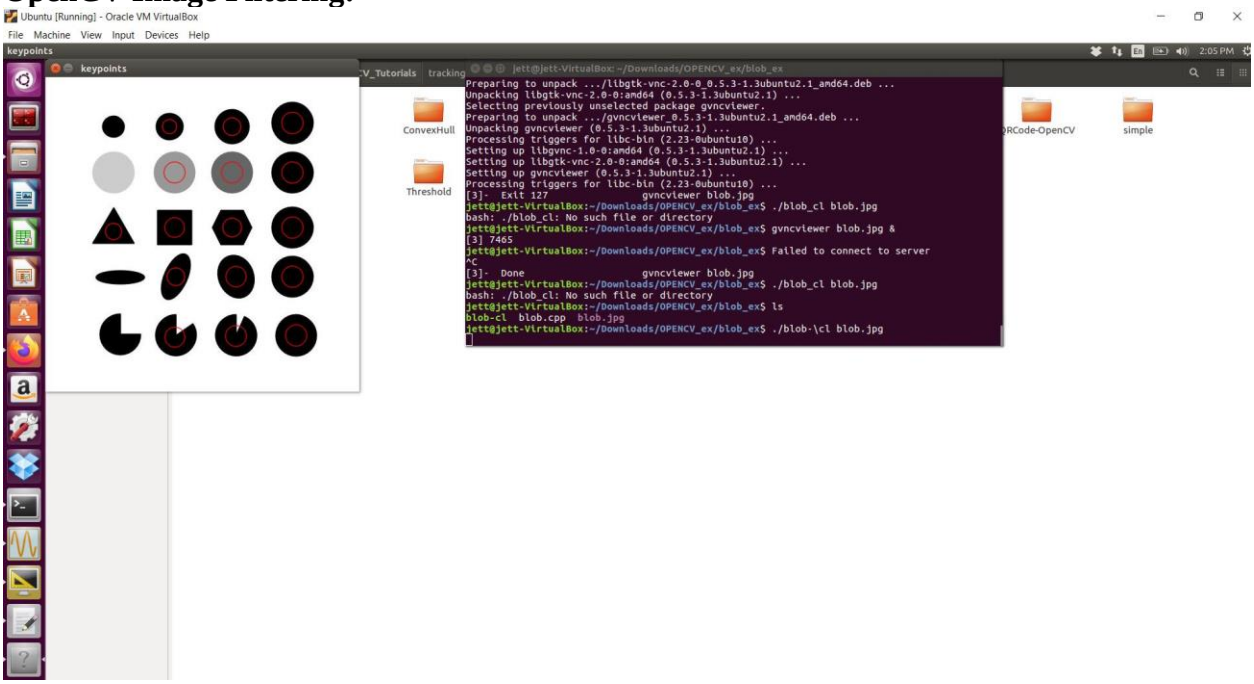
Encoder Testing:

```
// This program shows how to read the encoders on the Romi 32U4.
// The encoders can tell you how far, and in which direction each
// motor has turned.
//
// You can press button A on the Romi to drive both motors
// forward at full speed.  You can press button C to drive both
// motors in reverse at full speed.
//
// Encoder counts are printed to the LCD and to the serial
// monitor.
//
// On the LCD, the top line shows the counts from the left
// encoder, and the bottom line shows the counts from the right
// encoder.  Encoder errors should not happen, but if one does
// happen then the buzzer will beep and an exclamation mark will
// appear temporarily on the LCD.
//
// In the serial monitor, the first and second numbers represent
// counts from the left and right encoders, respectively.  The
// third and fourth numbers represent errors from the left and
// right encoders, respectively.

#include <Romi32U4.h>

Romi32U4Encoders encoders;
Romi32U4LCD lcd;
Romi32U4Buzzer buzzer;
Romi32U4Motors motors;
Romi32U4ButtonA buttonA;
Romi32U4ButtonC buttonC;

const char encoderErrorLeft[] PROGMEM = "!<c2";
const char encoderErrorRight[] PROGMEM = "!<e2";

char report[80];

void setup()
{

}
```

```
void loop()
{
  static uint8_t lastDisplayTime;
  static uint8_t displayErrorLeftCountdown = 0;
  static uint8_t displayErrorRightCountdown = 0;

  if ((uint8_t)(millis() - lastDisplayTime) >= 100)
  {
        lastDisplayTime = millis();

        int16_t countsLeft = encoders.getCountsLeft();
        int16_t countsRight = encoders.getCountsRight();

        bool errorLeft = encoders.checkErrorLeft();
        bool errorRight = encoders.checkErrorRight();

        if(encoders.checkErrorLeft())
        {
        // An error occurred on the left encoder channel.
        // Display it on the LCD for the next 10 iterations and
        // also beep.
        displayErrorLeftCountdown = 10;
        buzzer.playFromProgramSpace(encoderErrorLeft);
        }

        if(encoders.checkErrorRight())
        {
        // An error occurred on the left encoder channel.
        // Display it on the LCD for the next 10 iterations and
        // also beep.
        displayErrorRightCountdown = 10;
        buzzer.playFromProgramSpace(encoderErrorRight);
        }

        // Update the LCD with encoder counts and error info.
        lcd.clear();
        lcd.print(countsLeft);
        lcd.gotoXY(0, 1);
        lcd.print(countsRight);

        if (displayErrorLeftCountdown)
        {
        // Show an exclamation point on the first line to
        // indicate an error from the left encoder.
        lcd.gotoXY(7, 0);
        lcd.print('!');
```

```
        displayErrorLeftCountdown--;
        }

        if (displayErrorRightCountdown)
        {
        // Show an exclamation point on the second line to
        // indicate an error from the left encoder.
        lcd.gotoXY(7, 1);
        lcd.print('!');
        displayErrorRightCountdown--;
        }

        // Send the information to the serial monitor also.
        snprintf_P(report, sizeof(report),
        PSTR("%6d %6d %1d %1d"),
        countsLeft, countsRight, errorLeft, errorRight);
        Serial.println(report);
 }

 if (buttonA.isPressed())
 {
        motors.setSpeeds(300, 300);
 }
 else if (buttonC.isPressed())
 {
        motors.setSpeeds(-300, -300);
 }
 else
 {
        motors.setSpeeds(0, 0);
 }
}
```

OpenCV Image Filtering:



Canny Filter Code:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
```

```
img = cv.imread('flower.jpeg',0)
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

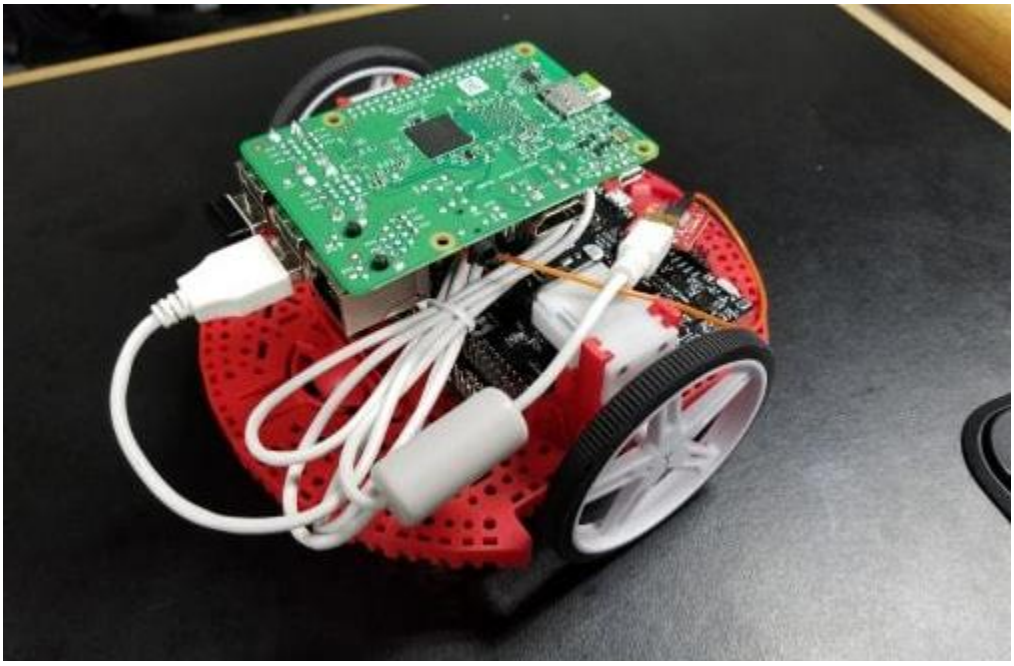
Morph Filter Code:

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):

    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_red = np.array([30,150,50])
    upper_red = np.array([255,255,180])

    mask = cv2.inRange(hsv, lower_red, upper_red)
    res = cv2.bitwise_and(frame,frame, mask= mask)

    kernel = np.ones((5,5),np.uint8)
    erosion = cv2.erode(mask,kernel,iterations = 1)
    dilation = cv2.dilate(mask,kernel,iterations = 1)

    cv2.imshow('Original',frame)
    cv2.imshow('Mask',mask)
    cv2.imshow('Erosion',erosion)
    cv2.imshow('Dilation',dilation)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
    break

cv2.destroyAllWindows()
cap.release()
```

-

Setting up RPI: Use the raspberry pi 3+/4 to setup Raspbian + ROS (Kinetic/Medoldic) + OpenCV.

Romi Robot with Raspberry Pi 3 using RX and TX from the Romi board using FTDI

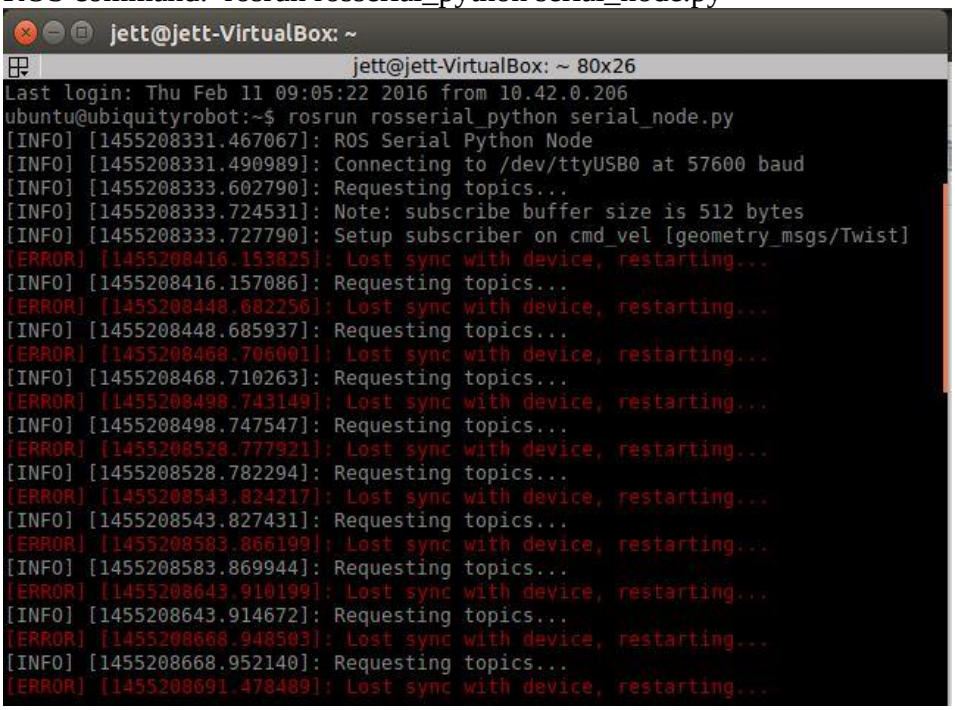


Romi-NoRpi-Debug:

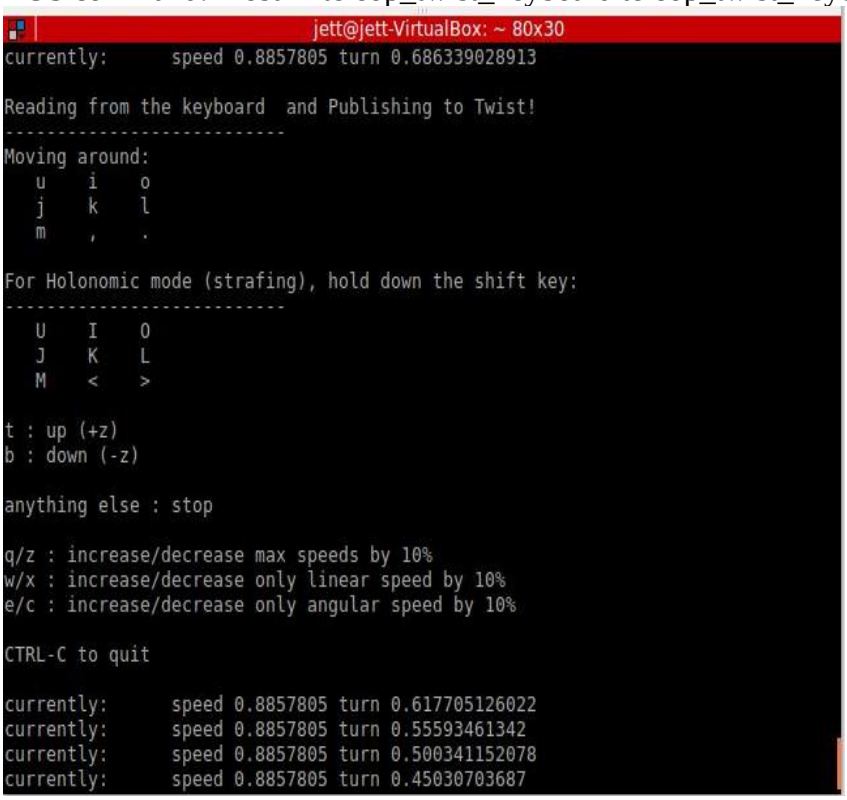
```
#include <Romi32U4.h>
#include <PololuRPiSlave.h>

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;


  //set_twist_target(_debug_linear_ms, _debug_angle_rs);
}

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();
  }

}
```

Final Romi Robot:

ROS command: rosrun rosserial_python serial_node.py



ROS command: rosurn teleop_twist_keyboard teleop_twist_keyboard.py



Final Robot Code:

```
#include <Romi32U4.h>
#include <ros.h>
#include <geometry_msgs/Twist.h>


#define EN_L 9
#define IN1_L 10
#define IN2_L 11

#define EN_R 8
#define IN1_R 12
#define IN2_R 13


double w_r=0, w_l=0;

//wheel_rad is the wheel radius ,wheel_sep is
double wheel_rad = 0.0325, wheel_sep = 0.295;


ros::NodeHandle nh;
int lowSpeed = 200;
int highSpeed = 50;
double speed_ang=0, speed_lin=0;

void messageCb( const geometry_msgs::Twist& msg){
  speed_ang = msg.angular.z;
  speed_lin = msg.linear.x;
  w_r = (speed_lin/wheel_rad) + ((speed_ang*wheel_sep)/(2.0*wheel_rad));
  w_l = (speed_lin/wheel_rad) - ((speed_ang*wheel_sep)/(2.0*wheel_rad));
}


ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel", &messageCb );
void Motors_init();
void MotorL(int Pulse_Width1);
void MotorR(int Pulse_Width2);

void setup(){
 Motors_init();
 nh.initNode();
 nh.subscribe(sub);
}
```

```
void loop(){
 MotorL(w_l*10);
 MotorR(w_r*10);
 nh.spinOnce();
}


void Motors_init(){

 pinMode(EN_L, OUTPUT);
 pinMode(EN_R, OUTPUT);
 pinMode(IN1_L, OUTPUT);
 pinMode(IN2_L, OUTPUT);
 pinMode(IN1_R, OUTPUT);
 pinMode(IN2_R, OUTPUT);
 digitalWrite(EN_L, LOW);
 digitalWrite(EN_R, LOW);
 digitalWrite(IN1_L, LOW);
 digitalWrite(IN2_L, LOW);
 digitalWrite(IN1_R, LOW);
 digitalWrite(IN2_R, LOW);
}

void MotorL(int Pulse_Width1){
 if (Pulse_Width1 > 0){
        analogWrite(EN_L, Pulse_Width1);
        digitalWrite(IN1_L, HIGH);
        digitalWrite(IN2_L, LOW);
 }

 if (Pulse_Width1 < 0){
        Pulse_Width1=abs(Pulse_Width1);
        analogWrite(EN_L, Pulse_Width1);
        digitalWrite(IN1_L, LOW);
        digitalWrite(IN2_L, HIGH);
 }

 if (Pulse_Width1 == 0){
        analogWrite(EN_L, Pulse_Width1);
        digitalWrite(IN1_L, LOW);
        digitalWrite(IN2_L, LOW);
 }
}
```

```
void MotorR(int Pulse_Width2){
 if (Pulse_Width2 > 0){
        analogWrite(EN_R, Pulse_Width2);
        digitalWrite(IN1_R, LOW);
        digitalWrite(IN2_R, HIGH);
 }

 if (Pulse_Width2 < 0){
        Pulse_Width2=abs(Pulse_Width2);
        analogWrite(EN_R, Pulse_Width2);
        digitalWrite(IN1_R, HIGH);
        digitalWrite(IN2_R, LOW);
 }

 if (Pulse_Width2 == 0){
        analogWrite(EN_R, Pulse_Width2);
        digitalWrite(IN1_R, LOW);
        digitalWrite(IN2_R, LOW);
 }

}
```

Final Project Video: https://youtu.be/aYKLrKXD32Q