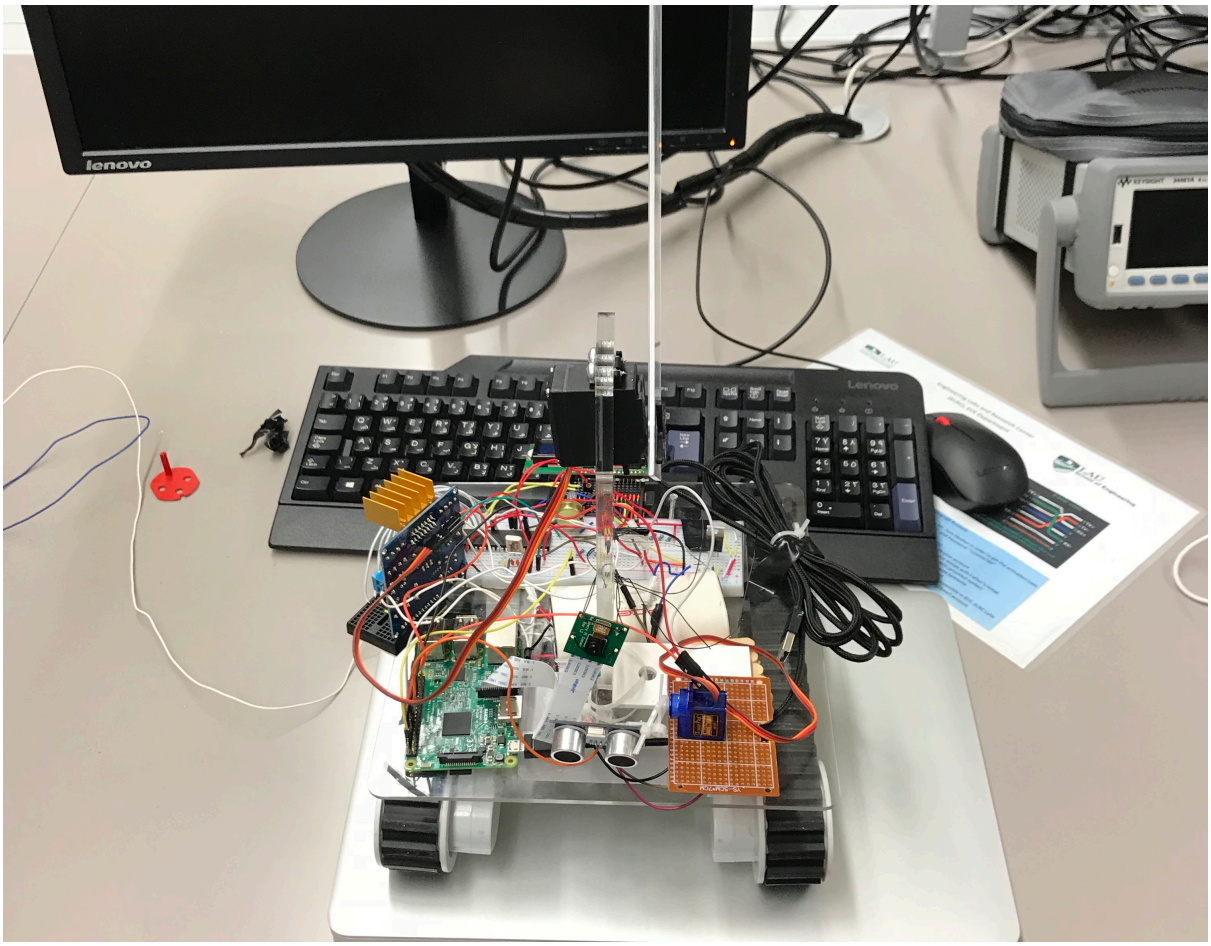
LAU School of Engineering

Dr. Zahi Nakad
Embedded Systems
December 7, 2018

Final Project Report



Automated Balloon Buster

Elliott Haddad 201406004
Charbel Chemaly 201402541
Marc Salibi 201402844
Elias Saliba 201303565

Table of Contents

Table of Figures

Introduction

In this project, we are required to combine everything we have learned throughout this semester and add a little twist to it. Mainly, we are required to let our rover pop red balloons.

- The Raspberry Pi will act as the eyes and brain of the system, processing each picture and detecting what should the rover do next
- Once a picture is taken, the processing starts and F, L, and R mapping to forward, left and right is sent to the PIC serially so that it can move closer to a balloon
- When forward is active, the rover should know when to stop using an ultrasonic sensor so that the arm can pop the balloon
- After popping a balloon, the rover continues the search to pop all 3 balloons

Part 1 - Image Processing

With the help of demos 5 and 6 we managed to detect a red balloon using SimpleCV

```
161         subprocess.call("raspistill -n -w %s -h %s -o project.bmp" % (640, 480), shell=True)
162         img = SimpleCV.Image("project.bmp")
163         dist = -img.hueDistance(SimpleCV.Color.RED)
164         filt= dist.stretch(225,255)
165         erode=filt.erode(2)
166         blobs=erode.findBlobs()
167         erode.save("project_red.bmp")
168         if blobs:
169                     circles=blobs.filter([b.isCircle(0.65) for b in blobs])
170                     if circles:
171                                 x= circles[-1].x
172                                 print (x)
```

Figure 1 - Red Balloon Detection

A picture is first taken then applied to a hue filter to detect the red color. We then change the threshold of the filter so that it can detect more accurate red colors instead of any shade of red. A blur is then applied to make the picture more accurate for object detection. The function find blobs which determines the edges is called to find the circles (ellipses)

The variable x is then computed to determine the center of each balloon.

Part 2 - Rover Movement

After successfully detecting a red balloon the rover should correctly move towards the ballon and this is done using serial communication between the PIC and the PI

```
173                     if x<285:
174                                 ser.write('R'.encode('UTF-8')) # because of crossed wiring we changed the left and right
175                                 d=320-x
176                                 d=d/35
177                                 m=str(d)
178                                 ser.write(m.encode('UTF-8'))
179                                 print("Left")
180
181
182                     else:
183                             if x>355:
184                                 ser.write('L'.encode('UTF-8'))
185                                 d=x-320
186                                 d=d/35
187                                 m=str(d)
188                                 ser.write(m.encode('UTF-8'))
189                                 print("Right")
190                             else:
191                                 ser.write('F'.encode('UTF-8'))
192                                 print("pre")
193                                 moved = 0
194                                 loop()
195
196
197
198                                 print("post")
199
200
201             else:
202                     print ('no')
203                     search()
204
```

Figure 2 - Rover Movement to Red Balloon

Once the center is detected in the 640x480 picture we have to move rover depending on the pixels in the image. If x is between 285 and 355, it means that the

balloon is in the center of the image and and ‘F’ is sent to the PIC so that it can move the rover forward.
If x is less that 285, the rover should move left but in our case the ‘R’ letter is sent because we crossed the wires and didn’t have easy access to reverse our move. That’s why we changed it through the code.
Finally if x > 355 the rover will move Right

```
106  lcddata(input);
107   switch (input) {
108      case 'F' :
109                     MR1= 0;
110                     MR2= 1;
111                     ML1= 0;
112                     ML2= 1;
113
114         break;
115      case 'P' :
116                     MR1= 1;
117                     MR2= 0;
118                     ML1= 0;
119                     ML2= 1;
120
121         break;
122      case 'L' :
123                     pic=0;
124                     ang = RCREG;
125                     ang=ang-48;
126                     right(ang);
127                     pic=1;
128
129         break;
130      case 'R' :
131                     pic=0;
132                     ang = RCREG;
133                     ang=ang-48;
134                     left(ang);
135                     pic=1;
136
137         break;
138      case 'S' :
139                     MR1= 0;
140                     MR2= 0;
141                     ML1= 0;
142                     ML2= 0;
143
144         break;
145      default :
```

Figure 3 - C Code for Rover Movement

Once the PIC receives a character from the PI it processes it as follows:

- F: to move forward
- P: to search in case no balloon was found
- L and R: for left and right respectively
- S: to stop the rover

Part 3 - Ultrasonic Sensor and Arm

To pop a balloon, we had to use an ultrasonic sensor so that a threshold can be created for the arm to move and destroy the balloon.

Once the forward command is called, a small servo holding our sensor will start rotating to detect any close balloons. If something is detected the small servo stop its rotation and the big servo holding the arm will rotate and pop the balloon.

```
OFFSE_DUTY = 0.5        #define pulse offset of servo
SERVO_MIN_DUTY = 2.5+OFFSE_DUTY     #define pulse duty cycle for minimum angle of servo
SERVO_MAX_DUTY = 12.5+OFFSE_DUTY    #define pulse duty cycle for maximum angle of servo
smallServoPin = 12
bigServoPin = 32

def map( value, fromLow, fromHigh, toLow, toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

trigPin = 16
echoPin = 18
MAX_DISTANCE = 220          #define the maximum measured distance
timeOut = MAX_DISTANCE*60   #calculate timeout according to the maximum measured distance

def pulseIn(pin,level,timeOut): # function pulseIn: obtain pulse time of a pin
    t0 = time.time()
    while(GPIO.input(pin) != level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    t0 = time.time()
    while(GPIO.input(pin) == level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    pulseTime = (time.time() - t0)*1000000
    return pulseTime

def getSonar():     #get the measurement results of ultrasonic module,with unit: cm
    GPIO.output(trigPin,GPIO.HIGH)      #make trigPin send 10us high level
    time.sleep(0.00001)     #10us
    GPIO.output(trigPin,GPIO.LOW)
    pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)   #read plus time of echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0     # the sound speed is 340m/s, and calculate distance
    return distance
```

Figure 4 - Servo and Ultrasonic Initializations (1)

Both of our servos are defined with the same duty cycles and pulse offset but the differ in the frequency at which they rotate. A PWM signal is then sent out with the corresponding angle to rotate each of the servos.

As for the Ultrasonic sensor, it sends out a trigger signal and requires some time to receive an echo signal signal. Therefore we have a small delay to read correct (no junk values for example 0.12 cm and nothing is in front of the sensor).

```
def setup():
    global p
    global w
    print ('Program is starting...')
    GPIO.setmode(GPIO.BOARD)       #numbers GPIOs by physical location
    GPIO.setup(trigPin, GPIO.OUT)   #
    GPIO.setup(echoPin, GPIO.IN)    #
    GPIO.setup(smallServoPin, GPIO.OUT)   # Set smallServoPin's mode is output
    GPIO.setup(bigServoPin, GPIO.OUT) # Set bigServoPin's mode as output
    GPIO.output(smallServoPin, GPIO.LOW)  # Set smallServoPin to low
    GPIO.output(bigServoPin, GPIO.LOW) # Set bigServoPin to Low

    p = GPIO.PWM(smallServoPin, 20)     # set Frequency to 40Hz of small servo to rotate while keeping the ultrasonic healthy to catch
        distances
    p.start(0)                     # Duty Cycle = 0
    w = GPIO.PWM(bigServoPin, 50)   # frequency of the servo that is mnoving the arm
    w.start(0)

def servoWriteP(angle):      # make the ultrasonic servo rotate to specific angle (0-180 degrees)
    if(angle<0):
        angle = 0
    elif(angle > 90):
        angle = 90
    p.ChangeDutyCycle(map(angle,0,90,SERVO_MIN_DUTY,SERVO_MAX_DUTY))#map the angle to duty cycle and output it

def servoWriteW(angle):  # make the ultrasonic servo rotate to specific angle (0-180 degrees)
    if(angle<0):
        angle = 0
    elif(angle > 180):
        angle = 180
    w.ChangeDutyCycle(map(angle,0,180,SERVO_MIN_DUTY,SERVO_MAX_DUTY))
```

Figure 5 - Servo and Ultrasonic Initializations (2)

In figure 5, we initialized the ultrasonic and both servos in the setup() function by assigning each one of them the correct GPIO pin.

Ultrasonic:

For the ultrasonic echo pin we applied a voltage division to transform 5V into 3.3V so that the PI doesn't burn its GPIO input pin.

Servos:

The PI can generate 2 PWM signals from pins 12 and 32 and this is where the signal wire of the servos was wired.

```
def loop():
    GPIO.setup(11,GPIO.IN)
    print("post: " + str(moved))
    while(moved == 0):
        print("new " + str(moved))
        distance = getSonar()
        #if(distance>32):
        moveSensorServoDown(60)
        #if(distance>32):
        moveSensorServoUp(60)
```

Figure 6 - Loop() function

After calling forward the loop() function is then triggered as a routine for the ultrasonic and arm movements. The ultrasonic sensor starts scanning for a value less that 32cm while its servo denoted by smallServo start rotating with an angle of 60 degrees up and down respectively.

```
95
96  def moveSensorServoUp(angle):
97      distance = getSonar()
98      print("The distance is : %.2f cm"%(distance))
99      for dc in range(0,angle+1, 1):
100         #print("Up")
101         servoWriteP(dc)
102         time.sleep(0.001)
103         if(distance<32 and distance>2): # default sensor value 0
104             global moved
105             moveArm()
106             moved = 1
107             print(moved)
108             break
109
110     time.sleep(0.5)
111
112 def moveSensorServoDown(angle):
113     distance = getSonar()
114     print("The distance is : %.2f cm"%(distance))
115     for dc in range(angle+1, -1, -1): #make servo rotate from 180 to 0 deg
116         servoWriteP(dc)
117         time.sleep(0.001)
118         #print("Down")
119         if(distance<32 and distance>2):
120             global moved
121             moveArm()
122             moved = 1
123             print(moved)
124             break
125
126     time.sleep(0.5)
```

Figure 7 - Ultrasonic Servo Movement

While moving the smallServo up and down, an if statement occurs to check wether the distance measured by the servo is between 32cm and 2cm: 32 being the distance for a perfect balloon pop and 2 is to filter out bad receptions. If this statement occurs, the global variable moved is changed to 1 so that the moveServoSensor functions can be broken and the scanning for another balloon happens. And moveArm() is called, which moves the bigServo 120 degrees twice to secure a balloon pop.

Part 4 - Schematics

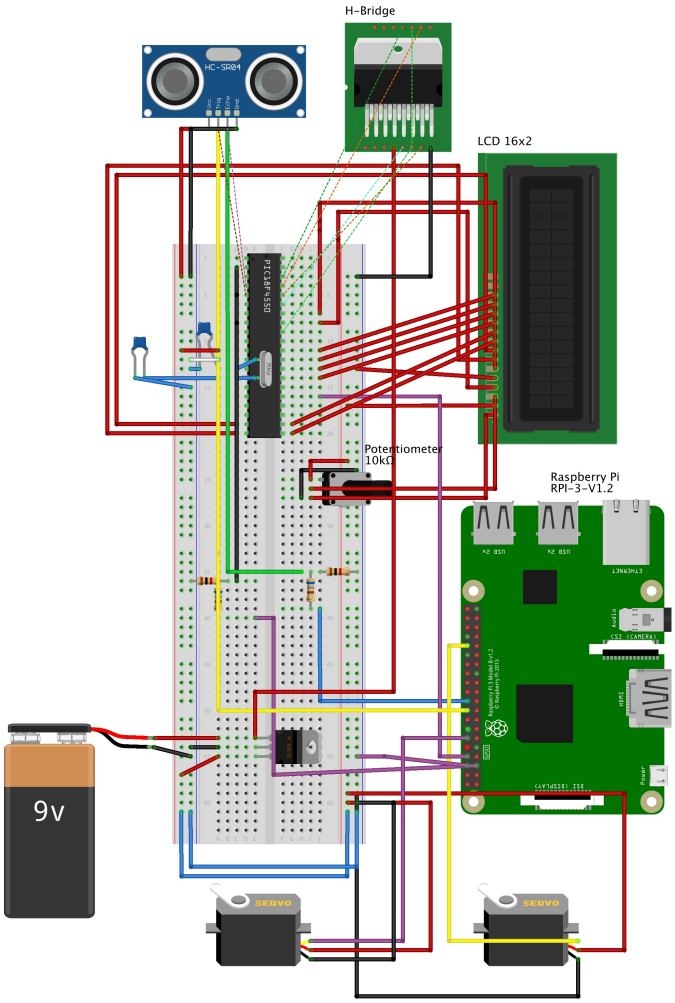


Figure 8 - BreadBoard Schematic

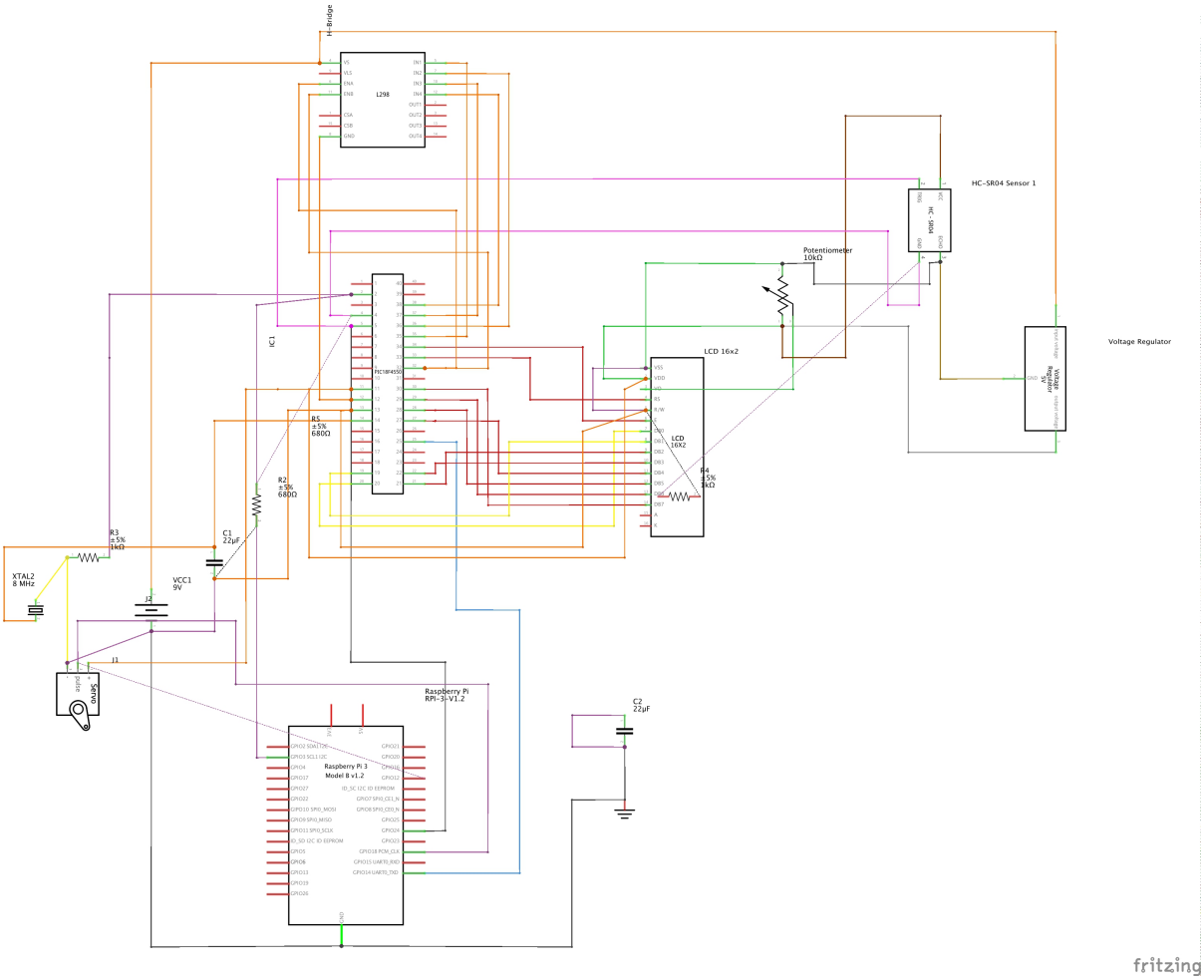


Figure 9 - Schematic

Both schemas where drawn using Fritzing Software

References

- Matt. (2015, March 08). Quick Guide To nano Text Editor On The Raspberry Pi. Retrieved from https://www.raspberrypi-spy.co.uk/2013/11/quick-guide-to-nano-text-editor-on-the-raspberry-pi/
- Pinout.xyz. (2018). *BCM 18 (PWM0) at Raspberry Pi GPIO Pinout*. [online] Available at: https://pinout.xyz/pinout/pin12_gpio18 [Accessed 7 Dec. 2018].
- Pi?, H. (2018). *How can I test if the serial / UART is good on a Raspberry Pi?*. [online] Raspberry Pi Stack Exchange. Available at: https://raspberrypi.stackexchange.com/questions/26593/how-can-i-test-if-the-serial-uart-is-good-on-a-raspberry-pi [Accessed 7 Dec. 2018].
- Servocity.com. (2018). *Servo Power & Speed*. [online] Available at: https://www.servocity.com/servo-power-speed [Accessed 7 Dec. 2018].
- Electronicoscaldas.com. (2018). [online] Available at: https://www.electronicoscaldas.com/datasheet/MG995_Tower-Pro.pdf [Accessed 7 Dec. 2018].
- Fritzing.org. (2018). *Fritzing*. [online] Available at: http://fritzing.org/home/ [Accessed 7 Dec. 2018].