SortME The Computer Vision Robot

By Trevor Sherrard
Email: SherrardTr@gmail.com
Project Website: Robosort.com

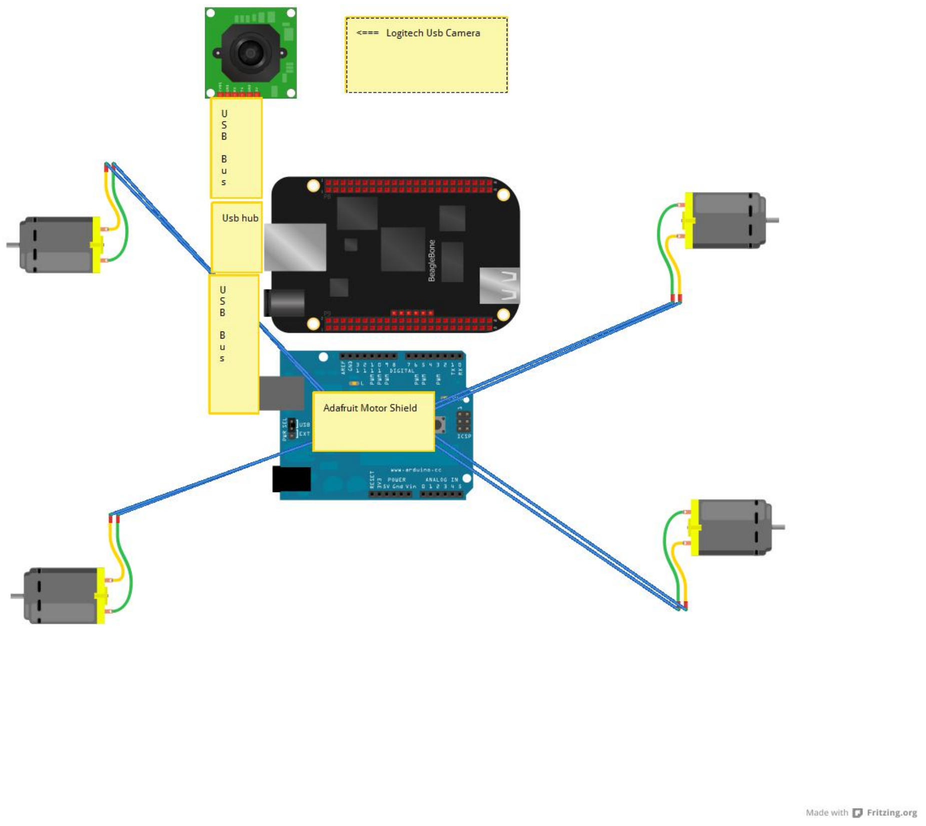
Introduction

SortME was a project originally thought of for ImagineRIT, the innovation and creativity festival at Rochester Institute of Technology. However, the project has become much more complex than originally intended. It has become a way to develop more skills in the area of computer vision. Not only have I learned more about the field, but I also am able to better showcase my current skills. Most importantly however I hope this description of this project gives you a guide, or at least peaks your interest, in the field of computer vision.

Structural Design

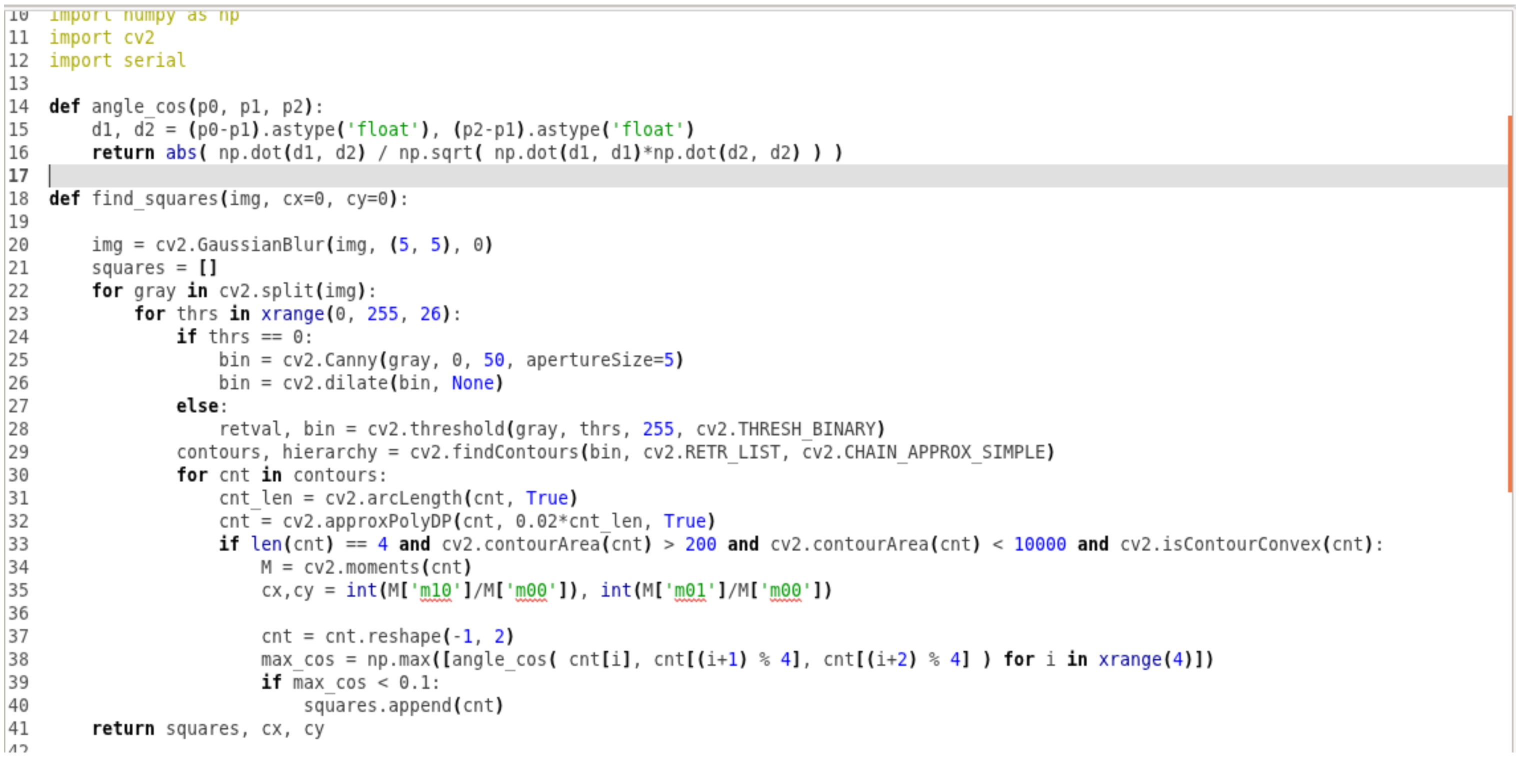
The current SortME chassis is a simple piece of ABS plastic with micro gear motors mounted on the underside(See figure 1). Mecanum wheels were used (See figure 2). Although the robot is currently structurally sound, plans are in the words to make a 3D printed robot chassis.

Electrical Design



Python Code Walk Through

To preface, this I understand that all of the code on my github is written in C++, but i am actively translating it to python, as pySerial was the only way i could get the serial communication between the beaglebone and arduino to work reliably.



```
10  import numpy as np
11  import cv2
12  import serial
13
14  def angle_cos(p0, p1, p2):
15      d1, d2 = (p0-p1).astype('float'), (p2-p1).astype('float')
16      return abs( np.dot(d1, d2) / np.sqrt( np.dot(d1, d1)*np.dot(d2, d2) ) )
17
18  def find_squares(img, cx=0, cy=0):
19
20      img = cv2.GaussianBlur(img, (5, 5), 0)
21      squares = []
22      for gray in cv2.split(img):
23          for thrs in xrange(0, 255, 26):
24              if thrs == 0:
25                  bin = cv2.Canny(gray, 0, 50, apertureSize=5)
26                  bin = cv2.dilate(bin, None)
27              else:
28                  retval, bin = cv2.threshold(gray, thrs, 255, cv2.THRESH_BINARY)
29              contours, hierarchy = cv2.findContours(bin, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
30              for cnt in contours:
31                  cnt_len = cv2.arcLength(cnt, True)
32                  cnt = cv2.approxPolyDP(cnt, 0.02*cnt_len, True)
33                  if len(cnt) == 4 and cv2.contourArea(cnt) > 200 and cv2.contourArea(cnt) < 10000 and cv2.isContourConvex(cnt):
34                      M = cv2.moments(cnt)
35                      cx,cy = int(M['m10']/M['m00']), int(M['m01']/M['m00'])
36
37                      cnt = cnt.reshape(-1, 2)
38                      max_cos = np.max([angle_cos( cnt[i], cnt[(i+1) % 4], cnt[(i+2) % 4] ) for i in xrange(4)])
39                      if max_cos < 0.1:
40                          squares.append(cnt)
41      return squares, cx, cy
42
```

lines 10 through 12- these lines import the numpy, openCV and pySerial libraries.

lines 14 through 16- these lines are a function that compute the cosine of the squares using the dot product to make sure the contours found are "square enough".

line 18- this line contains the function definition for the function that...well...finds squares. parameters are the image obtained from the camera, and two default parameters for the center x and y coordinates.

line 20- A gaussian blur is applied to the image. this allows the sharper details of the image that may cause issues later on in the detections processes to be eliminated.

line 21- Initializes an array to hold the square contours when they are found.

lines 22 through 28- These lines apply a threshold to the image, allowing the FindContours function call to work properly, as the FindContours function needs a binary Image.

line 29- This line finds possible contours in the binary image, and stores them in the "contours" variable.

line 30- This line is the loop declaration for the loop that loops through all potential contours.

line 31- This line gets the arc length or the perimeter of the current contour, and stores i in the cnt_len variable.

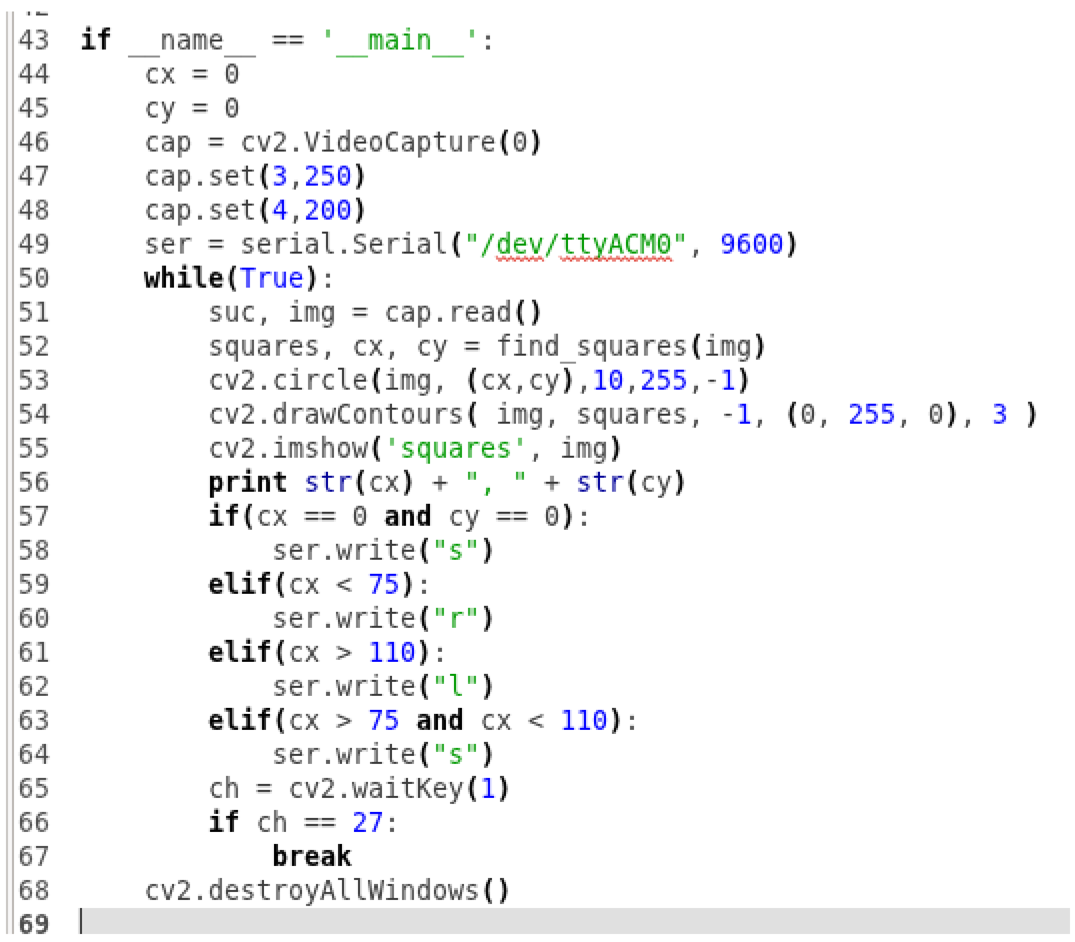
line 32- This line calls the ApproxPolyDP function which approximates the shape of the current contour. The function then returns a contour with that approximation, which is overwritten in the cnt variable.

line 33- Checks if the approximated contour has 4 sides, has a pixel area between 200 and 10,000 and checks if the contour is closed.

lines 34 through 35- gets the moment of the current contour. line 35 then gets the x and y center of mass from this moment.

lines 37 through 40- makes sure each angle does not exceed the max cosine that would allow it to be closed. In other words it makes sure the contour is "square enough". If it is square enough, it is added to the squares list.

line 41- returns the squares found in the current image, and the x and y coordinate of the most recently found square. Although only returning the most recent center of mass coordinates may seem like a bad idea, However, I have found that in most cases, the most recently found square in each loop iteration is the biggest, and thus the square the robot should be looking for in the test environment.



```
43 if __name__ == '__main__':
44     cx = 0
45     cy = 0
46     cap = cv2.VideoCapture(0)
47     cap.set(3,250)
48     cap.set(4,200)
49     ser = serial.Serial("/dev/ttyACM0", 9600)
50     while(True):
51         suc, img = cap.read()
52         squares, cx, cy = find_squares(img)
53         cv2.circle(img, (cx,cy),10,255,-1)
54         cv2.drawContours( img, squares, -1, (0, 255, 0), 3 )
55         cv2.imshow('squares', img)
56         print str(cx) + ", " + str(cy)
57         if(cx == 0 and cy == 0):
58             ser.write("s")
59         elif(cx < 75):
60             ser.write("r")
61         elif(cx > 110):
62             ser.write("l")
63         elif(cx > 75 and cx < 110):
64             ser.write("s")
65         ch = cv2.waitKey(1)
66         if ch == 27:
67             break
68     cv2.destroyAllWindows()
69 
```

lines 43 through 45- Initializes main, and creates a variable for the center of mass coordinates (x,y).

line 46- constructs a VideoCapture object using the default camera, which in this case is the 0th camera. This was very hard to figure out when I started using openCV so I'll clarify it now, when using VideoCapture on a laptop with a built in camera, or a system with two cameras, initialize the object with 0 for the default camera and 1,2… respectively for other capture devices. (-1 will initialize the first available camera).

line 47 and 48- These lines restrict the captured image to have a height of 200 and a width of 250 pixels.

line 49- This line initializes a pySerial object which on most linux machines will be /dev/ttyACM0. I would suggest a baud rate of 9600. It doesn't matter what you use, just make sure the arduino program that is running on your arduino has a matching baud rate.

line 51- reads an images and stores it in img. the other variable is a boolean that represents whether the capture was successful or not.

line 52- calls our findSquares function passing it the image we just captured the returned contours and x and y coordinates are stored.

line 53- draws a blue circle on our image to mark the center of the contours (cx, cy).

line 54- draws over our square contours obtained from findSquares, in green.

line 55- shows the original image with the contour and center point drawn on it.

line 56- prints the ordered pair of our contours center of mass for the current loop iteration.

line 57 through 64- checks whether the x-center of mass is in the center, to the left, or to the right of the screen. If the center of mass is in the center, or if the center of mass is equivalent (0,0) (ie, there is no square found), an ‘s’ is sent to the arduino. the char ‘s’ in my arduino program made the motors stop. otherwise an ‘l’ or’r’ was sent to the arduino. In my arduino program an ‘l’ made the robot strafe left and an ‘r’ made the robot strafe right.

Note about the arduino program: all you need to do for your arduino program is read from the serial monitor using the same baud rate that you used in your python program, in my case 9600. you can then respond however you want with your arduino (i.e motor movements, lights, e.t.c).

line 65- This line is key, It waits one millisecond, and stores and keystrokes that are entered during that loop iteration. **Without the delay you will not see an image.**

lines 67 and 68- This line checks if the keystroke that may have been entered is equivalent to the ASCII ‘27’ or the ESC key. If it is, the loop is broken out of, and the window opened to see the image is closed.

See This Code in Action

Youtube link to robot demo: Link to demo

Notice how the wheels stop when the robot is centered, and then spin in opposite directions according to what side of the robot the piece of paper is on.