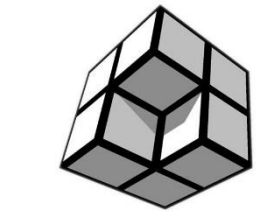


RMI SUMMER PROJECT

Autonomous Pick and place robot

Himadri Poddar

Instrumentation and Control Engineering
Roll- 110117035
NIT Tiruchirapalli
Date- May – June (2018)

INDEX

Acknowledgement

I would like to express my gratitude to the entire RMI team of NIT Tiruchirapalli to give me the opportunity to work on this summer project ' Autonomous Pick and place robot '. This project helped me in doing a lot of research and I came to know about so many things, really thankful to them.

I would like to thank my mentors Haarinathgobi, Rishabh and Harishankar who helped me a lot in finalising the project and reminding me time to time so that I could complete the project within the limited time frame. Out of their busy schedule, they spent time to guide me throughout the project and clear my doubts whenever I got struck. I extend my sincere thanks to all of them.

This had been a great learning experience for this two months. Everyday working on the project and giving weekly feedback to the mentors really helped me without which the project may not possibly be completed.

1. STAGES OF THE PROJECT

1.1. Basic Task:

To make a robotic arm with three servo motors which can move to a coordinate given to it by the user.

1.2. Advanced Task 1:

This included finding the pixel values of the centroid of any object (having a contradicting colour with the background) using OpenCV-Python.

1.3. Advanced Task 2:

This included distinguishing between two colours (I used red and blue) and finding the pixel values of their centroids separately so that the arm could pick and place the objects in two different positions.

2. SOFTWARE REQUIREMENTS

The basic software or libraries that needed to be installed are:

- **Python:** I used Python IDLE (version 2.7.13). Basic knowledge of python and its syntax is a must for this project.

- **Arduino**: The Arduino software is needed to control the microprocessor (ATMega32).

- **OpenCV:** The OpenCV library downloaded from their official site. **OpenCV** (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision.

- **Numpy:** NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices.

3. HARDWARE REQUIREMENTS

The basic hardware components that are required:

3.1. ServoMotors

3.1.1 Futaba (S3003)

-Torque (4.2 kg-cm)
- Used as the elbow motor
- controlled by the TIMER0

3.1.2 GO TECK (GS-5515 MG)

-Torque (13 kg-cm)
-Used as the shoulder motor
- Controlled by the OCR1A value of TIMER1

3.1.3 HSR(5990 TG)

-Torque (23 kg-cm)
- used as the base motor
- Controlled by the OCR1B value of TIMER1.

3.2. L293D motor driver:

The L293D Motor Driver is used to control the electromagnet.

3.3. Webcamera:

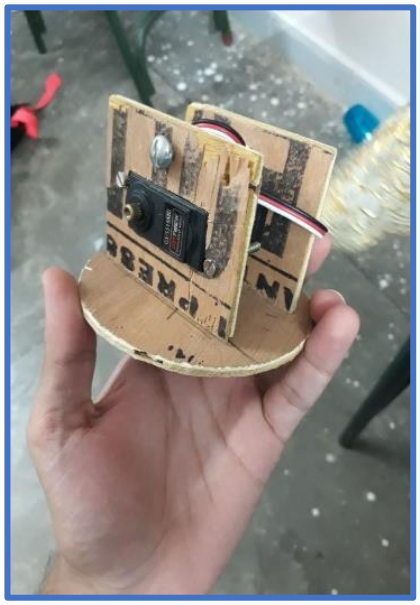
I used a Logitech C170 Webcam for taking up the image.

3.4. Microcontroller:

Arduino Uno board is used for the project. The Arduino UNO is a widely used open-source microcontroller board based on the ATmega328P microcontroller.

4. Fabrication

The plywood of 3mm thick has been cut into different shapes and been screwed or glued to get the desired structures. A few snaps of the various parts have been given. The servos are being tightly screwed so as to remain in its position even in vigorous movements.

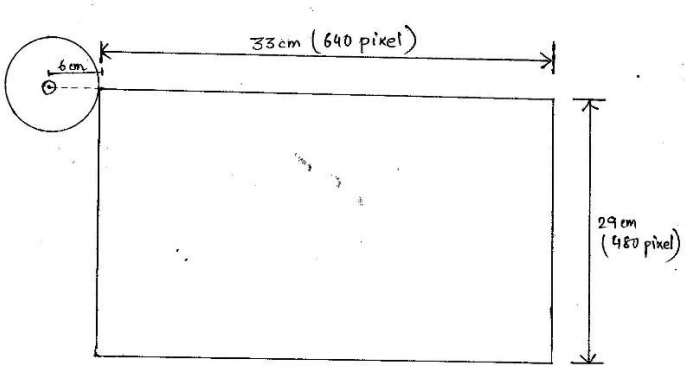


5. BASIC TASK

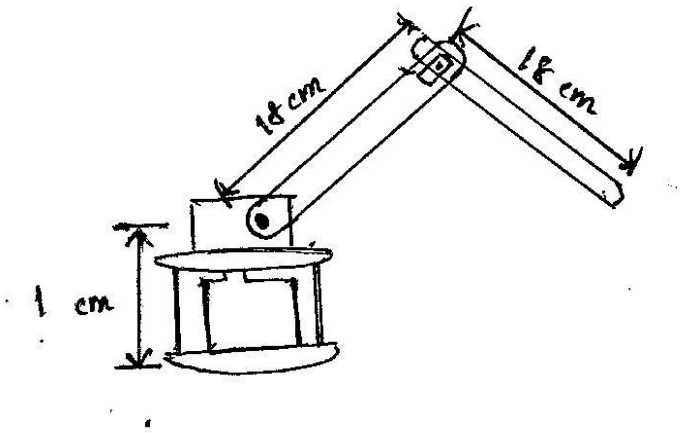
5.1. Objective: To make a robotic arm which will move to a particular coordinate.

5.2. Concept: Three servos are being used. Each will move a particular angle so that the end point of the arm will reach the required coordinate.

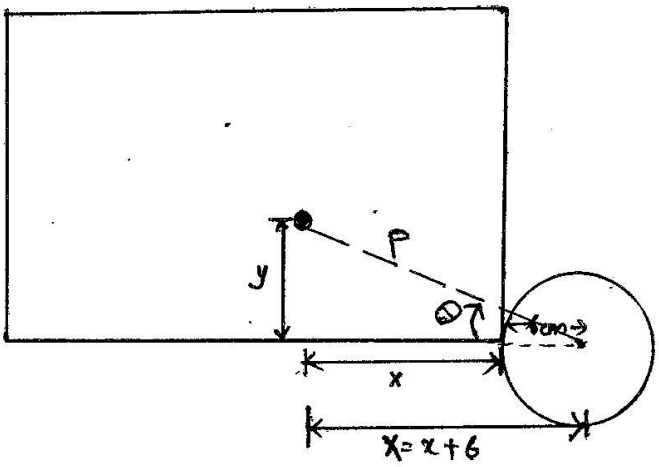
5.3. Working Area: A 33cm X 29 cm space is being used as the working area. The arm base is kept at the left top corner with its centre approx. 6 cm away from corner. The top left corner of the rectangle is considered to be the origin for the cartesian plane.



5.4. Robotic arm design:

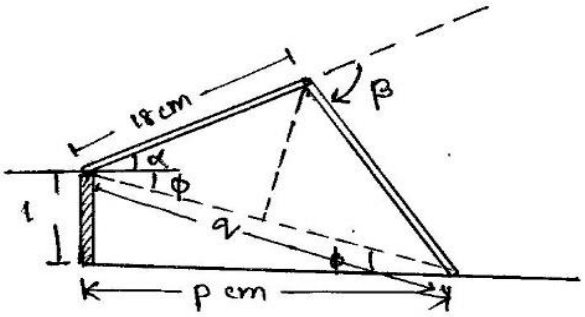


5.5. Inverse Kinematics:



$$P = \sqrt{X^2 + Y^2}$$

$$\theta = \tan^{-1}\left(\frac{Y}{X}\right)$$



$$q = \sqrt{p^2 + (10)^2}$$

$$\phi = \tan^{-1}\left(\frac{10}{p}\right)$$

$$\cos(\alpha + \phi) = \left(\frac{q/2}{18}\right)$$

$$\Rightarrow (\alpha + \phi) = \cos^{-1}\left(\frac{q}{36}\right)$$

$$\therefore \alpha = \cos^{-1}\left(\frac{q}{36}\right) - \tan^{-1}\left(\frac{10}{p}\right)$$

and, $\beta = 2(\alpha + \phi)$

$$= 2\cos^{-1}\left(\frac{q}{36}\right)$$

Base servo will sweep angle θ.
Shoulder servo will sweep angle α upwards.
Elbow servo will sweep angle β downwards.

5.6. ARDUINO CODE FOR BASIC TASK

```
#include<avr/io.h>
#include<math.h>
#include<util/delay.h>


// simple wait function
void Wait()
{
  uint8_t i;
  for(i=0;i<150;i++)
  {
    _delay_loop_2(0);
    _delay_loop_2(0);
    _delay_loop_2(0);
  }
}


int main()
{
  float cx=94,cy=336; // give the pixel values


  float x,y,X,Y;
  float m,n,o,p,q,l,z,a;
  x=cx * 0.052; // conversion of the pixel values to coordinates
  y=cy * 0.061; // conversion of the pixel values to coordinates
  X=x+6;
  Y=y;

// inverse kinematics part

  p=sqrt(X*X+ Y*Y);
  m=atan(Y/X)*57.3;
  q=sqrt(p*p + 100);
  n=acos(q/36)*57.3;
  a=(acos(q/36) - atan(10/p))*57.3;
  o=(2*n);
  z=180-o;


  // Configure the TIMER0
  TCCR0A |= (1<<COM0A1) |(1<<WGM01) | (1<< WGM00);
  TCCR0B |=  (1<< CS02) ;
  DDRD= 0xFF;   // setting the entire port D as output
  DDRC= 0xFF;   // setting the entire Port C as output
```

```
//Configure TIMER1
   TCCR1A|=(1<<COM1A1)|(1<<COM1B1)|(1<<WGM11);        //NON Inverted PWM
   TCCR1B|=(1<<WGM13)|(1<<WGM12)|(1<<CS11)|(1<<CS10); //PRESCALER=64 MODE
14(FAST PWM)
   ICR1=4999;  //fPWM=50Hz (Period = 20ms).


    DDRB|=0xFF;   //PWM Pins as Out


   while(1)
   {


      OCR1B=(180+(2.28*m))+15;//BASE SERVO
      OCR1A=(180+(2.28*a))-8; //SHOULDER SERVO
      OCR0A=(50+(0.56*z));  // ELBOW SERVO
      Serial.print(OCR0A);
      PORTC=0b00000001;
      Wait();

      while(1)
      {
        OCR1B=170;  //BASE SERVO
        OCR1A=260; //SHOULDER SERVO
        OCR0A=90;   // ELBOW SERVO
        PORTC=0b00000001;

        Wait();
        PORTC=0b00000000;
        PORTC=0b00000000;

        while(1)

        {
            OCR1B=180;  //BASE SERVO
            OCR1A=370; //SHOULDER SERVO
            OCR0A=150; // ELBOW SERVO

        }
      }

   }
}
```

6. Configuration of the TIMERS

Timers can be used as a waveform generator if the OC1A and OC1B pins are set as output. The TCCR registers of the timer determine how the waveform generator works. When the TCNTn register reaches the top or compare match, the waveform generator is informed. Then the waveform generator changes the state of the OC0 pin according to the mode.

6.1. TIMER1:

TIMER1 has been set to the fast PWM mode. The timer counts up until it reaches the top(here the ICR1) and then rolls over to zero. For the fast PWM mode, the WGM11, WGM12 and WGM13 bits are set to 1. The COM1A1 and COM1B1 bits are set to get a non inverted PWM. In this mode the OC1A and OC1B values clear on Compare match, and the output is high when the compare level is above the waveform level and low otherwise.

6.2. TIMER0:

The Timer0 is also used in fast PWM non- inverted mode but with a prescaler of 256. The COM0A1 bit has been set for the non-inverted mode. WGM00 and WGM01 are set for the fast PWM mode.

6.3. OCR values and servo angle:

6.3.1. GoTeck(GS-5515MG)

- **6.3.1.1.** 0 deg =150(OCR1A)
- **6.3.1.2.** 90 deg=370(OCR1A)
- **6.3.1.3.** 180 deg=590(OCR1A)

6.3.2. HSR 5990 TG

- **6.3.2.1.** 0 deg=180(OCR1B)
- **6.3.2.2.** 90 deg=390(OCR1B)
- **6.3.2.3.** 180 deg=600(OCR1B)

6.3.3. Futaba S3003

- **6.3.3.1.** 0 deg = 50 (OCR0A)
- **6.3.3.2.** 90 deg = 100 (OCR0A)
- **6.3.3.3.** 180 deg = 150(OCR0A)

7. ADVANCED TASK 1

7.1. **Objective:** To get the pixel values of the centroid of any shape of an object.

7.2. **Approach:** Firstly**,** here the image is being captured by an external camera and pre-processed in python compiler using opencv library.

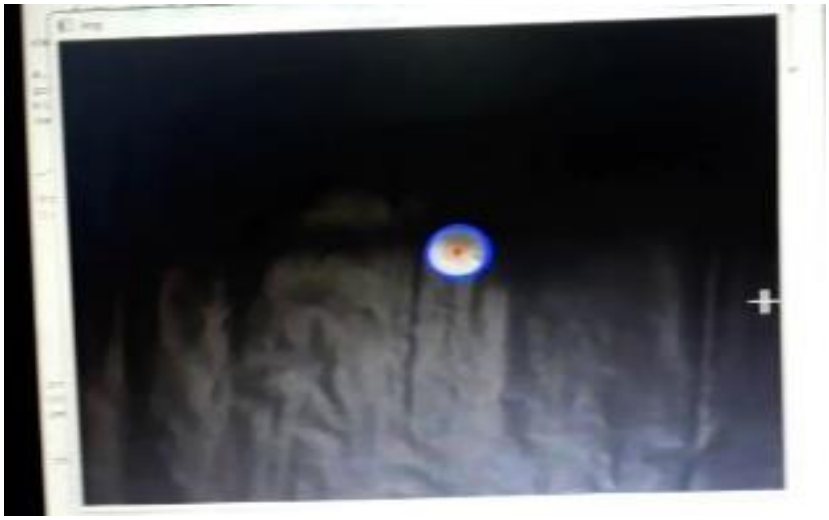
The outline of the object is being traced and later the center of the contour i.e. the centroid location is determinrd.

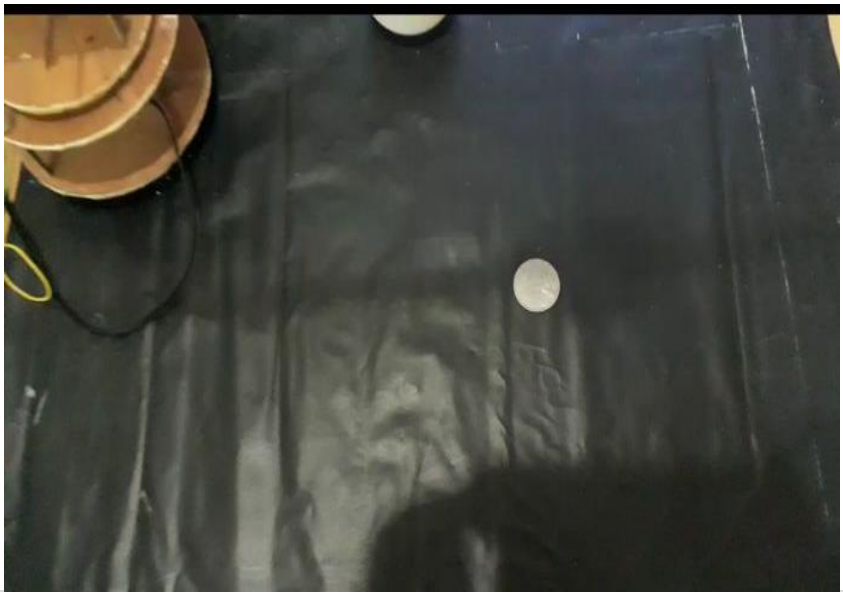
7.2.1. The image is converted to grayscale.

7.2.2. Blurring to reduce the background noise so as to percept the object boundary clearly.

7.2.3. Binarization(here used threshing)

7.2.4. Compute the centre of the contour.





8. OpenCV-Python Code for finding centroid

```
import numpy as np
import cv2


cam = cv2.VideoCapture(1)

s,frame=cam.read()
gray= cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
blurred=cv2.GaussianBlur(gray,(5,5),0)
ret,thresh=cv2.threshold(blurred,100,255,0)


_,contours,_=
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

#cnt = contours[0]
for c in contours:
    M=cv2.moments(c)
    cx= int(M["m10"]/ M["m00"])
    cy= int(M["m01"]/ M["m00"])
    cv2.drawContours(frame, [c], -1, (255,0,0), 3)
    cv2.circle(frame,(cx,cy),5,(0,0,255),-1)
    print cx
    print cy



cv2.imshow('img',frame)
cv2.imshow('thresh',thresh)
cv2.imshow('thresh',blurred)
k=cv2.waitKey(0)


cv2.destroyAllWindows()
```

9. ADVANCED TASK 2

9.1.Objective:

To differentiate between two colours and find the centroid separately

9.2. Approach:

- Blurred the image using the cv2.Gaussianblur command
- The blurred image is then threshed with a certain threshold whose value is being decided on the lightning condition.
- Here the image is not converted to grayscale but to HSV
- The HSV converted image will be handled twice separately, one where it will detect only blue colour and the other when it will detect only red colour.
- For the colour detection, lower and upper ranges are fixed for each colour.
- The images are then separately dealt with for finding the contours.

9.3. OpenCV-Python Code for finding centroid for different colours

```
import cv2
import numpy as np


cam = cv2.VideoCapture(0)


# Take each frame
_, frame=cam.read()
blurred=cv2.GaussianBlur(frame,(5,5),0)
ret,thresh=cv2.threshold(blurred,150,255,0)

# Convert BGR to HSV
hsv = cv2.cvtColor(thresh, cv2.COLOR_BGR2HSV)

# define range of blue color in HSV
lower_red = np.array([0,100,100])
upper_red = np.array([20,255,255])
lower_blue = np.array([50,50,50])
upper_blue = np.array([120,255,255])


# Threshold the HSV image to get only blue colors
maskb = cv2.inRange(hsv, lower_blue, upper_blue)
maskr = cv2.inRange(hsv, lower_red, upper_red)

# Bitwise-AND mask and original image
resb = cv2.bitwise_and(frame,frame, mask= maskb)
resr = cv2.bitwise_and(frame,frame, mask= maskr)

mask_invb=cv2.bitwise_not(resb)
mask_invr=cv2.bitwise_not(resr)

grayb= cv2.cvtColor(mask_invb,cv2.COLOR_BGR2GRAY)
grayr= cv2.cvtColor(mask_invr,cv2.COLOR_BGR2GRAY)
#blurred1=cv2.GaussianBlur(gray,(5,5),0)
#ret,thresh1=cv2.threshold(blurred1,50,255,0)

finalb=cv2.bitwise_not(grayb)
finalr=cv2.bitwise_not(grayr)
_,contoursb,_=
cv2.findContours(finalb,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

```
for b in contoursb:
    M=cv2.moments(b)
    cxb= int(M["m10"]/ M["m00"])
    cyb= int(M["m01"]/ M["m00"])
    cv2.drawContours(frame, [b], -1, (0,255,0), 3)
    cv2.circle(frame,(cxb,cyb),5,(0,255,0),-1)
    print("for blue")
    print cxb
    print cyb
    break

_,contours,_=
cv2.findContours(finalr,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

for r in contoursr:
    R=cv2.moments(r)
    cxr= int(R["m10"]/ R["m00"])
    cyr= int(R["m01"]/ R["m00"])
    cv2.drawContours(frame, [r], -1, (0,255,0), 3)
    cv2.circle(frame,(cxr,cyr),5,(0,255,0),-1)
    print ("for red")
    print cxr
    #print cyr
    break


cv2.imshow('frame',frame)
cv2.imshow('mask',mask_inv)
cv2.imshow('resb',finalb)
cv2.imshow('resr',finlr)
k=cv2.waitKey(0)


cv2.destroyAllWindows()
```

10. Project Report

10.1. May 2nd week- getting the components for the fabrication of the arm. Thought and made out the design of the structure. The unavailability of acrylic sheet made me to use plywood. The entire week was involved in cutting the plies and screwing it to get the desired design.

10.2. May 3rd week- studying the embedded c basics (here I have referred to the book by Md. Ali Mazidi for the concepts of timers, pwm types and so on.

10.3. May 4th week- started writing the Arduino code. Controlling three servos with a single microcontroller was a first time for me. The coding took a lot of time as I faced many problems in the accuracy of the timer0 as it's a 8 bit timer. However, fixed it and my bot can move to a coordinate provided. That's my basic task completed here.

10.4. June 1st week – downloaded python and opencv library for the image processing part. Studied the basic syntaxes of python language as its new to me.

10.5. June 2nd week- started reading the opencv documentation from their site and finally made out with the code of finding the centroid by contour detection. Here done with the advanced task 1.

10.6. June 3rd week – finding a gripper was the most gruelling task for me. Searched a lot but in vain. Later thought of making a gripper myself. But then also getting a mini servo in my locality was not so easy. At last, my mentors suggested to use an electromagnet and a coin.

10.7. June 4th week – worked on the ip part for the colour detection so that I can get the coordinates of the centroid of two coins (one coloured blue and the other red) separately. Here done with the Advanced task 2.

10.8. July 1st week- made necessary changes in the Arduino code so that my bot first can pick the blue coin place it in a container then move again to the red one and place it in another container.

11. Problem faced

1st problem- timer 0 is a 8 bit timer and I was not getting enough precision for one of the three servos. As the other two are being controlled by the OCR1A and OCR1B of timer 1. Later tried many prescalers and sorted out one. Decided to use the timer 0 controlled servo in the elbow part as if also less precise its going to work fine.

2nd problem- I was getting many noises while finding the centroid. Solved it by blurring the image

3rd problem- for the advanced task 2, differentiating the colour and finding their centroids separately was very troublesome. Solved it by creating two windows one filtering the red colour and the other with the blue only. Found their centroid separately and then clubbed them in another single window.

12. PIN DIAGRAM