Concordia University

Department of Mechanical, Industrial and Aerospace Engineering MECH 6311 Automation with Computer Vision

Final Report



|  |  |
| --- | --- |
| Name | ID |
| Rouzbeh Sedighi | 40085069 |
| Roshkumar Panakkal Nandakumar | 40104382 |
| Abdullah Alhoothy | 40075668 |

**Computer vision aided 3D printer :**

Contents

[Introduction 4](#_Toc7979745)

[Program Flow Chart 4](#_Toc7979746)

[small sections of program (the original and interesting parts), 5](#_Toc7979747)

[Interesting part # 1 5](#_Toc7979748)

[Interesting part # 2 6](#_Toc7979749)

[Interesting part # 3 7](#_Toc7979750)

[Interesting part # 4 8](#_Toc7979751)

[Interesting part # 5 9](#_Toc7979752)

[Appendix 10](#_Toc7979753)

[Shape Detection code 10](#_Toc7979754)

[Composite colour filter implmentation 11](#_Toc7979755)

[Figure 1 Shadow detected as Object 5](#_Toc7977893)

[Figure 2 RGB compsite filter detection 5](file:///C:\Users\abdullah-laptop\Downloads\draft%20final%20report%20Mech6631%20.docx#_Toc7977894)

[Figure 3 RGB regular color detection 5](file:///C:\Users\abdullah-laptop\Downloads\draft%20final%20report%20Mech6631%20.docx#_Toc7977895)

[Figure 4 Perfect Square radii 6](file:///C:\Users\abdullah-laptop\Downloads\draft%20final%20report%20Mech6631%20.docx#_Toc7977896)

[Figure 5 detected squar radii 6](file:///C:\Users\abdullah-laptop\Downloads\draft%20final%20report%20Mech6631%20.docx#_Toc7977897)

[Figure 6 Single Pixel color consideration 7](file:///C:\Users\abdullah-laptop\Downloads\draft%20final%20report%20Mech6631%20.docx#_Toc7977898)

[Figure 7 five Pixel color consideration 7](#_Toc7977899)

[Figure 8 color Intesnity histogram 7](#_Toc7977900)

[Figure 9 Regular thresholding 7](file:///C:\Users\abdullah-laptop\Downloads\draft%20final%20report%20Mech6631%20.docx#_Toc7977901)

[Figure 10 Multiple Ranged thresholding 7](file:///C:\Users\abdullah-laptop\Downloads\draft%20final%20report%20Mech6631%20.docx#_Toc7977902)

# Introduction

This report summrizes our project “VisionBot” and provides highlights the process flow of the program. The main aim of the project is to use camera feedback to analyis shape,colour size,location and center of an object placed on the 3d printer’s bed and send commands the 3d printer to move to the location of the object and trace it’s cerumference

We started the project using OpenCV for 3 months before knowing we are not allowed to use external packages

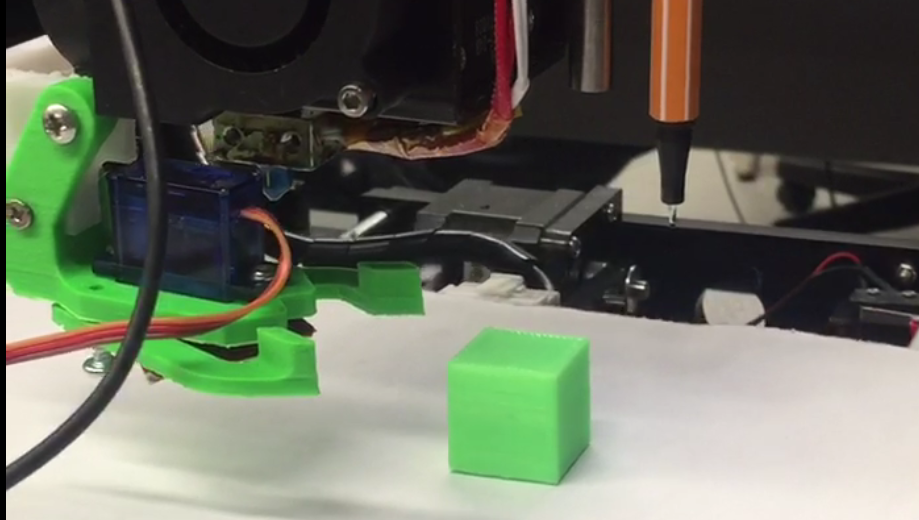
Also this project was executed by 3 members only who’s names are on the cover page

# Program Flow Chart

# small sections of program (the original and interesting parts)

## Original Part Gripper ([Code Appendix Link)](#_RGB_trace_1)

A gripper has been attached to the 3d printer, that will grip the object and put it somewhere else



## Interesting part # 0

Program is able to trace any object not just square or circle.RGB trace([Code Appendix link](#_RGB_trace))

## Interesting part # 1

Due to shadows, binary thersholding and gray scale image processing did not prefurm well , we replaced it with RGB image processing

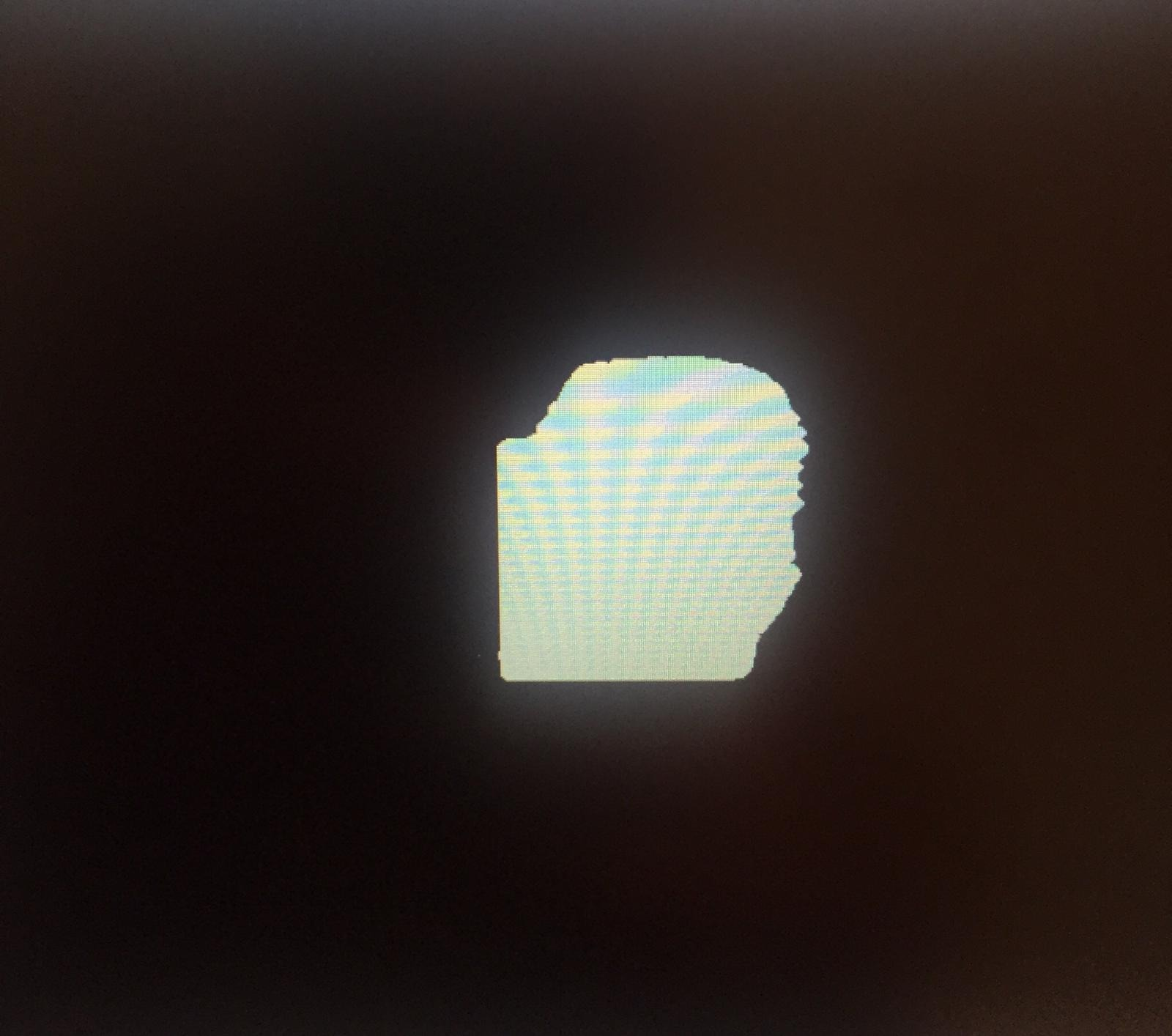
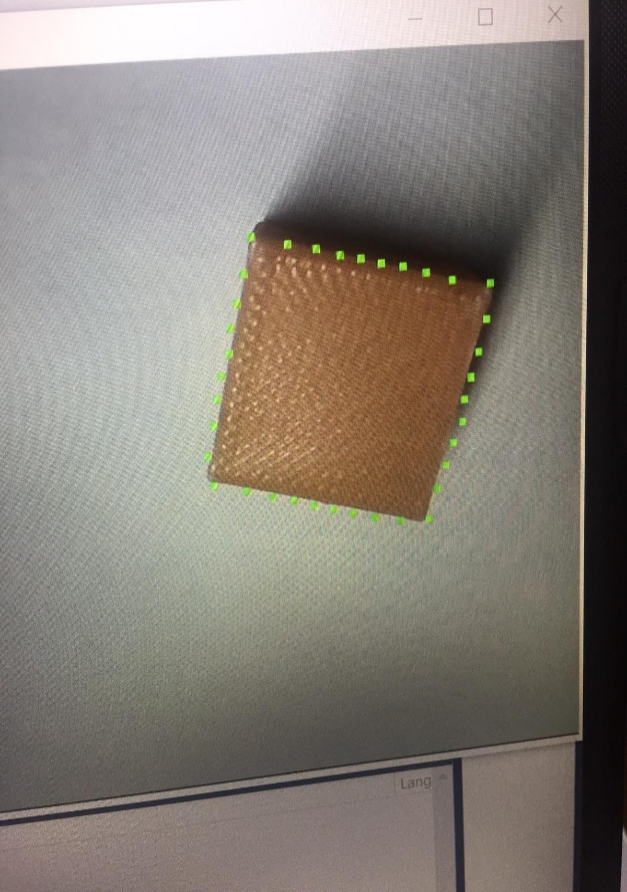


Figure 1 Shadow detected as Object

[Code Snippet (Appendix link)](#_RGB_Centroid)

## Interesting part # 2

In RGB image detection, using one color indecator didn’t prefurm well if the object had color variation due to light intensity on one side and not the other. To fix it we made composite filters that detect color difference. In case of orange Orangness = R-G and Difference = Red – Green



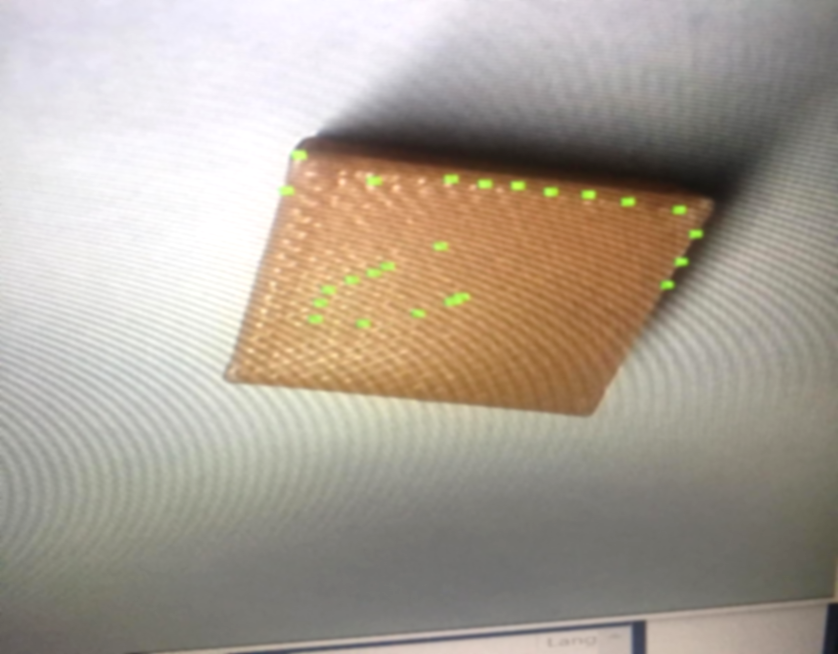
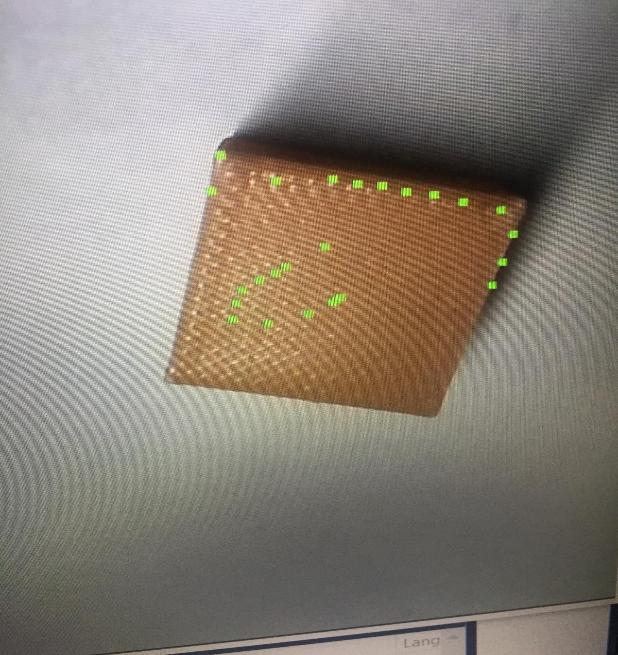


Figure 2 RGB regular color detection

Figure 3 RGB compsite filter detection

Code Snippet ([Appendix link](#_Composite_colour_filter))

if (colorName == "red")

{ int difference = r - b;

int orangeness = r - g;

if (r > 100 && orangeness < 70 && difference > 25 && b < 110)

//orange detection && g < 110//if (r > 120 && g < 110 && b < 100)

{ n++;

int i = q % 640;

Si += i; Sj += (q - i) / 640; }}

## Interesting part # 3

we decided to go an extra step and do shape detection relying on radius measurment to make decisions on the object in question. By using radius measurement from 8 differen’t angle starting with the maximum radius and then incrementing 45degrees

Note: a perfect square has maxium distance from centroid to one of it’s corners = r , distance to the shortest size = sin45\*r

Below is a histogram showing radius distribution over 360degrees or 6.28 rad. Difference between expected and actual was added as error margin in the code

Figure 4 detected squar radii

Figure 5 Perfect Square radii

**Code Snippet (**[**Appendix Link**](#_Shape_Detection_code)**)**

if (r\_max\*.90 < r\_120 && r\_120 < r\_max\*1.10 && shape\_detected == 0)

{if (r\_max\*.90 < r\_240 && r\_240 < r\_max\*1.10)

{// circle shape detection

if (r\_max\*.90 < r\_45 && r\_45 < r\_max\*1.10 && shape\_detected == 0)

{if (r\_max\*.90 < r\_90 && r\_90 < r\_max\*1.10)

{if (r\_max\*.90 < r\_180 && r\_180 < r\_max\*1.10)

{if (r\_max\*.90 < r\_225 && r\_225 < r\_max\*1.10)

{if (r\_max\*.90 < r\_270 && r\_270 < r\_max\*1.10)

{if (r\_max\*.90 < r\_315 && r\_315 < r\_max\*1.10)

{printf("\nshape is circle");

shape\_detected = 1;

## nteresting part # 4

becuase of different lighting conditions sometimes light reflection appearsd as white color on the object to resolve this condition we used a method of processing 5 pixel in a row to ensure that we reached to the edge(edge detection function)

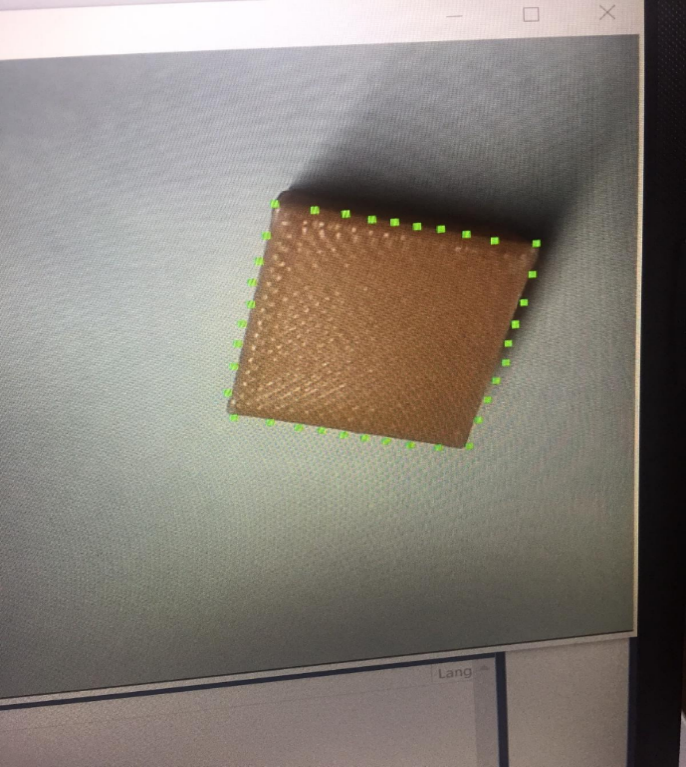
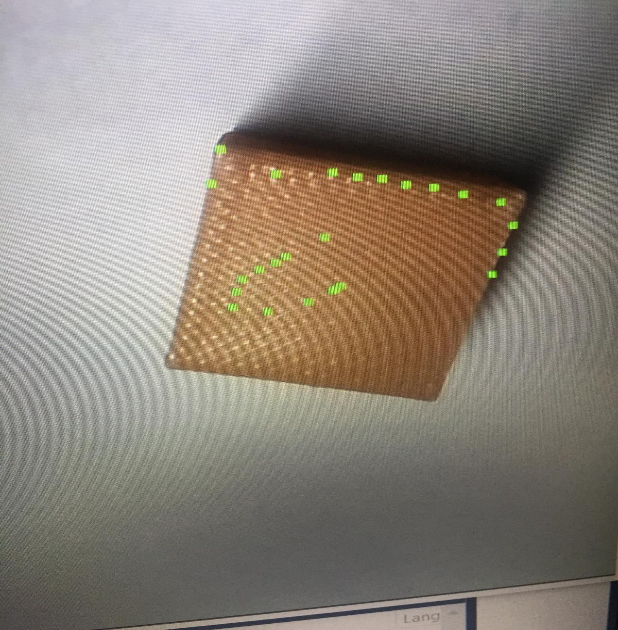


Figure 7 Single Pixel color consideration

Figure 6 after 5 pixel processing

Figure 8 five Pixel color consideration

**Code Snippet (**[**Appendix Link**](#_RGB_five_point)**)**

if (R > 110 && orangeness < 60 && difference > 25 && B < 110)

{ rTemp = r;

pn = 0; }

else

{ if (pn > 5)

{ r = rTemp;

break; }

else{ pn++;}}}

## Interesting part # 5

To remove some dark shadow we defined a new threshold function. This function applys differen’t thresholds to differen’t ranges of colors

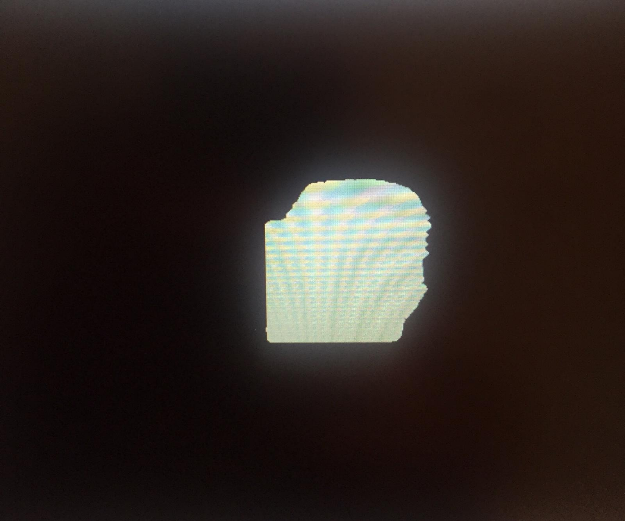
Figure 9 color Intesnity histogram

Figure 11 Multiple Ranged thresholding

Figure 10 Regular thresholding

Code Snippet ([Appendix Link)](#_Multi-range_thresholding)

// threshold operation

for (i = 0; i < size; i++) {

if (pa[i] < maxtvalue && pa[i]> mintvalue) pb[i] = 0;

else pb[i] = 255;}

return 0;}

# Appendix

## Shape Detection code

r\_45 = object\_radiusRGB5points(ic, jc, th\_max + 0.78535, colorName, rgb);

r\_90 = object\_radiusRGB5points(ic, jc, th\_max + 1.5707, colorName, rgb);

r\_120 = object\_radiusRGB5points(ic, jc, th\_max + 2.0942, colorName, rgb);

r\_135 = object\_radiusRGB5points(ic, jc, th\_max + 2.35605, colorName, rgb);

r\_180 = object\_radiusRGB5points(ic, jc, th\_max + 3.1414, colorName, rgb);

r\_225 = object\_radiusRGB5points(ic, jc, th\_max + 3.92675, colorName, rgb);

r\_240 = object\_radiusRGB5points(ic, jc, th\_max + 4.18853, colorName, rgb);

r\_270 = object\_radiusRGB5points(ic, jc, th\_max + 4.7121, colorName, rgb);

r\_315 = object\_radiusRGB5points(ic, jc, th\_max + 5.49745, colorName, rgb);

// triangle shape detection

if (r\_max\*.90 < r\_120 && r\_120 < r\_max\*1.10 && shape\_detected == 0)

{if (r\_max\*.90 < r\_240 && r\_240 < r\_max\*1.10)

{// circle shape detection

if (r\_max\*.90 < r\_45 && r\_45 < r\_max\*1.10 && shape\_detected == 0)

{if (r\_max\*.90 < r\_90 && r\_90 < r\_max\*1.10)

{if (r\_max\*.90 < r\_180 && r\_180 < r\_max\*1.10)

{if (r\_max\*.90 < r\_225 && r\_225 < r\_max\*1.10)

{if (r\_max\*.90 < r\_270 && r\_270 < r\_max\*1.10)

{if (r\_max\*.90 < r\_315 && r\_315 < r\_max\*1.10)

{printf("\nshape is circle");

shape\_detected = 1;

}}}}}}

if (shape\_detected == 0)

{printf("\nshape is triangle");

shape\_detected = 1;

}}}

// square shape detection

if (r\_max\*.90 < r\_90 && r\_90 < r\_max\*1.10 && shape\_detected == 0)

{if (r\_max\*.90 < r\_180 && r\_180 < r\_max\*1.10)

{if (r\_max\*.90 < r\_270 && r\_270 < r\_max\*1.10)

{printf("\nshape is square");

shape\_detected = 1;

}}}

if (shape\_detected == 0)

{printf("\n unknown shape");

}

## Composite colour filter implmentation

if (colorName == "red")

{ int difference = r - b;

int orangeness = r - g;

if (r > 100 && orangeness < 70 && difference > 25 && b < 110) //orange detection && g < 110//if (r > 120 && g < 110 && b < 100)

{ n++;

int i = q % 640;

Si += i;

Sj += (q - i) / 640;

}}

else if (colorName == "green")

{ int difference = g - r;

int greenness = g - b;

if (r < 60 && greenness > 30 && difference > 25 && g > 50 && b < 100)

{n++;

int i = q % 640;

Si += i;

Sj += (q - i) / 640;

}

## RGB Centroid

int RGBcentroid(image &rgb, double &ic, double &jc, string colorName){

int q = 0;

int k = 0;

int n = 0;

int Si = 0;

int Sj = 0;

ibyte r, g, b;

ic = 0;

jc = 0;

// number of pixels in image

int size = rgb.height \* rgb.width \* 3;

for (k = 0; k < size; k += 3) {

// components are stored in the order B-G-R

// B0 G0 R0 B1 G1 R1 .... Bsize-1 Gsize-1 Rsize-1

b = rgb.pdata[k];

g = rgb.pdata[k + 1];

r = rgb.pdata[k + 2];

if (colorName == "red")

{ int difference = r - b;

int orangeness = r - g;

if (r > 100 && orangeness < 70 && difference > 25 && b < 110) //orange detection && g < 110

{ n++;

int i = q % 640;

Si += i;

Sj += (q - i) / 640;

}}

## RGB five point

double object\_radiusRGB5points(double ic,double jc,double beta,string colorName,image &rgb)

{ double r = 0.0, dr, r\_max, rTemp;

int i, j, k, width, height, q, pn = 0;

ibyte \*pa;

width = rgb.width;

height = rgb.height;

pa = rgb.pdata;

dr = 0.5; // use 0.5 pixels just to be sure

r\_max = 180; // limit the max object radius size to something reasonable

for (r = dr; r < r\_max; r += dr) {

i = ic + r\*cos(beta);

j = jc + r\*sin(beta);

//cout << "\n I = " << i << "\nJ = " << j;

// limit i and j in case it gets out of bounds -> wild pointer

if (i < 0) i = 0;

if (i >= width) i = width;

if (j < 0) j = 0;

if (j >= height) j = height;

// convert i,j to image coord k

int B = pa[3 \* (width\*(int)(j)+(int)(i))];

int G = pa[3 \* (width\*(int)(j)+(int)(i)) + 1];

int R = pa[3 \* (width\*(int)(j)+(int)(i)) + 2];

if (colorName == "red")

{ int difference = R - B;

int orangeness = R - G;

//R > 100 && orangeness < 70 && difference > 25 && B < 110

if (R > 110 && orangeness < 60 && difference > 25 && B < 110)

{ rTemp = r;

pn = 0; }

else

{ if (pn > 5)

{ r = rTemp;

break; }

else

{ pn++;

}}}

## Multi-range thresholding

int threshold\_new(image &a, image &b, int maxtvalue, int mintvalue)

// binary threshold operation

// a - greyscale image

// b - binary image

// tvalue - threshold value

{ i4byte size, i;

ibyte \*pa, \*pb;

// initialize pointers

pa = a.pdata;

pb = b.pdata;

// check for compatibility of a, b

if (a.height != b.height || a.width != b.width) {

printf("\nerror in threshold: sizes of a, b are not the same!");

return 1;}

if (a.type != GREY\_IMAGE || b.type != GREY\_IMAGE) {

printf("\nerror in threshold: input types are not valid!");

return 1;}

// number of bytes

size = (i4byte)a.width \* a.height;

// threshold operation

for (i = 0; i < size; i++) {

if (pa[i] < maxtvalue && pa[i]> mintvalue) pb[i] = 0;

else pb[i] = 255;}

return 0;}

## RGB trace

void perimeter2(image &rgb, double &ic, double &jc, string colorName, float GeometricRatio, double PerimeterDistance[36][2])

{ double X0distance = 0.0;

double Y0distance = 0.0;

double X1distance = 0.0;

double Y1distance = 0.0;

//double r1 = object\_radius(ic, jc, 0, 1, label, rgb0);

double r0 = object\_radiusRGB5points(ic, jc, 0, colorName, rgb);

//cout << "\nradius 1=" << r1 / GeometricRatio;

// x and y direction actual distance from centroid

PerimeterDistance[0][0] = r0\*cos(3.14159 \* 0 / 180) / GeometricRatio; // r1 in this case since angle is zero

PerimeterDistance[0][1] = r0\*sin(3.14159 \* 0 / 180) / GeometricRatio; // 0 in this case since angle is zero

X0distance = PerimeterDistance[0][0];

Y0distance = PerimeterDistance[0][1];

int n = 0;

for (int i = 10; i < 351; i += 10)

{ n++;

double r1 = object\_radiusRGB5points(ic, jc, 3.14159 \* i / 180, colorName, rgb);

X1distance = r1\*cos(3.14159 \* i / 180) / GeometricRatio;

Y1distance = r1\*sin(3.14159 \* i / 180) / GeometricRatio;

PerimeterDistance[n][0] = X1distance;

PerimeterDistance[n][1] = Y1distance;}}

## Gripper close code

void closeGripper(HANDLE &h2)

{ char buffer[64];

char s1[1000];

// declare an output string stream of max NMAX charaters

ostrstream sout2(buffer, 64);

// declare an input string stream of max SMAX charaters

istrstream sin2(s1, 1000);

// set sout position to beginning so we can use it again

sout2.seekp(0);

sout2 << "2"; // move without extrusion

sout2 << '\0'; // terminate the string so strlen can function

// note the C++ compiler seems to hate "\0"

n = strlen(buffer); // number of bytes to send (excludes \0)

// for debugging

cout << "\nn = " << n;

cout << "\nbuffer = " << buffer;

// send the close command to the gripper

serial\_send(buffer, n, h2);

Sleep(100);}