1. **Learning to use vectors in C++:**

#include "stdafx.h"

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<vector<int>> A;

int i, j, a;

for (j = 0; j<2; ++j)

{

vector<int> temp;

for (i = 0; i<3; ++i)

{

cin >> a;

temp.push\_back(a);

}

A.push\_back(temp);

}

for (j = 0; j<2; ++j)

{

for (i = 0; i<3; ++i)

cout << A[j][i] % 10 << " ";

cout << endl;

}

cin >> a;

return 0;

}

1. **Learning to read and show an image:**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\lion.jpg", 0);

if (image.empty())

exit(1);

namedWindow("MyWindow", CV\_WINDOW\_AUTOSIZE);

imshow("MyWindow", image);

waitKey(0);

return 0;

}

1. **Indian Flag (incomplete):**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image(200, 300, CV\_8UC3);

int i, j;

for (i = 0; i < 200; ++i)

{

for (j = 0; j < 300; ++j)

{

if (i < 60)

{

image.at<Vec3b>(i, j)[0] = 10;

image.at<Vec3b>(i, j)[1] = 120;

image.at<Vec3b>(i, j)[2] = 255;

}

else if (i>140)

{

image.at<Vec3b>(i, j)[0] = 30;

image.at<Vec3b>(i, j)[1] = 255;

image.at<Vec3b>(i, j)[2] = 0;

}

else if (((j - 155)\*(j - 155) + (i - 100)\*(i - 100)<1580 && (j - 155)\*(j - 155) + (i - 100)\*(i - 100)>1124) || (j - 155)\*(j - 155) + (i - 100)\*(i-100) < 50)

{

image.at<Vec3b>(i, j)[0] = 255;

image.at<Vec3b>(i, j)[1] = 0;

image.at<Vec3b>(i, j)[2] = 0;

}

else

{

image.at<Vec3b>(i, j)[0] = 255;

image.at<Vec3b>(i, j)[1] = 255;

image.at<Vec3b>(i, j)[2] = 255;

}

}

}

namedWindow("MyWindow", CV\_WINDOW\_AUTOSIZE);

imshow("MyWindow", image);

waitKey(0);

return 0;

}

1. **BGR to binary(using global mean):**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\lion.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat binary(image.rows, image.cols, CV\_8UC1);

int i, j, sum = 0, count = 0;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

sum += (b + g + r)/3;

count++;

}

}

int thres = sum / count;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

if (26 \* b / 100 + 30 \* g / 100 + 43 \* r / 100>thres)

binary.at<uchar>(i, j) = 255;

else

binary.at<uchar>(i, j) = 0;

}

}

namedWindow("MyWindow", CV\_WINDOW\_AUTOSIZE);

imshow("MyWindow", binary);

waitKey(0);

return 0;

}

1. **Negative:**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\lion.jpg", CV\_LOAD\_IMAGE\_COLOR);

int i, j;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

image.at<Vec3b>(i, j)[0] = 255 - image.at<Vec3b>(i, j)[0];

image.at<Vec3b>(i, j)[1] = 255 - image.at<Vec3b>(i, j)[1];

image.at<Vec3b>(i, j)[2] = 255 - image.at<Vec3b>(i, j)[2];

}

}

namedWindow("MyWindow", CV\_WINDOW\_AUTOSIZE);

imshow("MyWindow", image);

waitKey(0);

return 0;

}

1. **BGR to greyscale:**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\lion.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat grey(image.rows, image.cols, CV\_8UC1);

int i, j;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (b + g + r) / 3;

}

}

namedWindow("MyWindow", CV\_WINDOW\_AUTOSIZE);

imshow("MyWindow", grey);

waitKey(0);

return 0;

}

1. **BGR to binary(using local mean):**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\blackwhite.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat binary(image.rows, image.cols, CV\_8UC1);

int i, j, k, l, sum, count, thres;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

sum = 0; count = 0;

for (k = i - 1; k <= i + 1; ++k)

{

for (l = j - 1; l <= j + 1; ++l)

{

if (k >= 0 && k < image.rows && l >= 0 && l < image.cols)

{

count++;

sum += (image.at<Vec3b>(k, l)[0] + image.at<Vec3b>(k, l)[1] + image.at<Vec3b>(k, l)[2])/3;

}

}

}

thres = sum / count;

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

if ((b+g+r)/3>thres)

binary.at<uchar>(i, j) = 255;

else

binary.at<uchar>(i, j) = 0;

}

}

namedWindow("Input", CV\_WINDOW\_AUTOSIZE);

imshow("Input", image);

namedWindow("MyWindow", CV\_WINDOW\_AUTOSIZE);

imshow("MyWindow", binary);

waitKey(0);

return 0;

}

1. **BGR to binary(using histogram and sum of freq concept)**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\lion.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat grey(image.rows, image.cols, CV\_8UC1);

Mat binary1(image.rows, image.cols, CV\_8UC1);

int i, j;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (b + g + r) / 3;

}

}

int freq[256];

for (i = 0; i < 256; ++i)

freq[i] = 0;

for (i = 0; i < image.rows; ++i)

for (j = 0; j < image.cols; ++j)

freq[grey.at<uchar>(i, j)]++;

int sum = 0, thres1 = 0;

while (sum < (image.rows\*image.cols) / 2)

sum += freq[thres1++];

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

if (grey.at<uchar>(i, j) > thres1)

binary1.at<uchar>(i, j) = 255;

else

binary1.at<uchar>(i, j) = 0;

}

}

namedWindow("Input", CV\_WINDOW\_AUTOSIZE);

namedWindow("Grey", CV\_WINDOW\_AUTOSIZE);

namedWindow("Freq", CV\_WINDOW\_AUTOSIZE);

imshow("Input", image);

imshow("Grey", grey);

imshow("Freq", binary1);

waitKey(0);

return 0;

}

1. **BGR to binary using trackbar:**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\lion.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat grey(image.rows, image.cols, CV\_8UC1);

Mat binary(image.rows, image.cols, CV\_8UC1);

int i, j, thres;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (b + g + r) / 3;

}

}

namedWindow("Input", CV\_WINDOW\_AUTOSIZE);

namedWindow("Grey", CV\_WINDOW\_AUTOSIZE);

imshow("Input", image);

imshow("Grey", grey);

namedWindow("Binary", CV\_WINDOW\_AUTOSIZE);

cvCreateTrackbar("Threshold", "Binary", &thres, 255);

while (1)

{

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

if (grey.at<uchar>(i, j)>thres)

binary.at<uchar>(i, j) = 255;

else

binary.at<uchar>(i, j) = 0;

}

}

imshow("Binary", binary);

int ikey = waitKey(50);

if (ikey == 27)

break;

}

waitKey(0);

return 0;

}

1. **Edge detection(naive way):**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\image.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat grey(image.rows, image.cols, CV\_8UC1);

Mat binary(image.rows, image.cols, CV\_8UC1);

int i, j, thres;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (b + g + r) / 3;

}

}

namedWindow("Input", CV\_WINDOW\_AUTOSIZE);

namedWindow("Grey", CV\_WINDOW\_AUTOSIZE);

imshow("Input", image);

imshow("Grey", grey);

namedWindow("Binary", CV\_WINDOW\_AUTOSIZE);

cvCreateTrackbar("Threshold", "Binary", &thres, 255);

while (1)

{

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

int max = 0, min = 255, k, l;

for (k = i - 1; k <= i + 1; ++k)

{

for (l = j - 1; l <= j + 1; ++l)

{

if (k >= 0 && k < image.rows && l >= 0 && l < image.cols)

{

if (grey.at<uchar>(k, l)>max)

max = grey.at<uchar>(k, l);

if (grey.at<uchar>(k, l) < min)

min = grey.at<uchar>(k, l);

}

}

}

if (max - min > thres)

binary.at<uchar>(i, j) = 0;

else

binary.at<uchar>(i, j) = 255;

}

}

imshow("Binary", binary);

int ikey = waitKey(50);

if (ikey == 27)

break;

}

waitKey(0);

return 0;

}

1. **Removing noise from binary images(Dilation):**

Inside any code, write the following:

Mat noiseless(image.rows, image.cols, CV\_8UC1);

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

int k, l, black=0, white=0;

for (k = i - 1; k <= i + 1; ++k)

{

for (l = j - 1; l <= j + 1; ++l)

{

if (k >= 0 && k < image.rows && l >= 0 && l < image.cols)

{

if (binary1.at<uchar>(k, l) == 0)

black++;

else

white++;

}

}

}

if (white>black)

noiseless.at<uchar>(i, j) = 255;

else

noiseless.at<uchar>(i, j) = 0;

}

}

1. **Canny edge detection:**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include<opencv2\imgproc\imgproc.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\sudoku2.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat grey(image.rows, image.cols, CV\_8UC1);

int i, j;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (33 \* b + 56 \* g + 11 \* r) / 100;

}

}

int freq[256];

for (i = 0; i < 256; ++i)

freq[i] = 0;

for (i = 0; i < image.rows; ++i)

for (j = 0; j < image.cols; ++j)

freq[grey.at<uchar>(i, j)]++;

Mat cannyedge(image.rows, image.cols, CV\_8UC1);

int sum=0, thres=0;

while (sum < (image.rows\*image.cols) / 2)

sum += freq[thres++];

int threslow = thres / 3;

int threshigh = thres \* 2 / 3;

Canny(grey, cannyedge, threslow, threshigh, 3);

namedWindow("Input", CV\_WINDOW\_AUTOSIZE);

namedWindow("Grey", CV\_WINDOW\_AUTOSIZE);

namedWindow("Canny", CV\_WINDOW\_AUTOSIZE);

imshow("Input", image);

imshow("Grey", grey);

imshow("Canny", cannyedge);

waitKey(0);

return 0;

}

1. **Video Capture, Binary and Noise reduction:**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

VideoCapture vid(0);

int thres=0;

namedWindow("Output", CV\_WND\_PROP\_FULLSCREEN);

cvCreateTrackbar("Threshold", "Output", &thres, 255);

while (1)

{

Mat image;

vid.read(image);

Mat grey(image.rows, image.cols, CV\_8UC1);

Mat binary(image.rows, image.cols, CV\_8UC1);

int i, j;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (b + g + r) / 3;

}

}

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

if (grey.at<uchar>(i, j)>thres)

binary.at<uchar>(i, j) = 255;

else

binary.at<uchar>(i, j) = 0;

}

}

Mat noiseless(image.rows, image.cols, CV\_8UC1);

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

int k, l, black = 0, white = 0;

for (k = i - 1; k <= i + 1; ++k)

{

for (l = j - 1; l <= j + 1; ++l)

{

if (k >= 0 && k < image.rows && l >= 0 && l < image.cols)

{

if (binary.at<uchar>(k, l) == 0)

black++;

else

white++;

}

}

}

if (white>black)

noiseless.at<uchar>(i, j) = 255;

else

noiseless.at<uchar>(i, j) = 0;

}

}

imshow("Output", noiseless);

int ikey = waitKey(30);

if (ikey == 27)

break;

}

waitKey(0);

return 0;

}

1. **Smoothing greyscale images:**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include<opencv2\imgproc\imgproc.hpp>

#include <iostream>

using namespace std;

using namespace cv;

#define MAX\_KERNEL\_LENGTH 10

int main()

{

Mat image = imread("G:\\lion.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat grey(image.rows, image.cols, CV\_8UC1);

Mat Blur(image.rows, image.cols, CV\_8UC1);

Mat Gaussian(image.rows, image.cols, CV\_8UC1);

Mat Median(image.rows, image.cols, CV\_8UC1);

Mat Bilateralfilter(image.rows, image.cols, CV\_8UC1);

int i, j;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (b + g + r) / 3;

}

}

namedWindow("Blur", CV\_WINDOW\_AUTOSIZE);

namedWindow("GaussianBlur", CV\_WINDOW\_AUTOSIZE);

namedWindow("MedianBlur", CV\_WINDOW\_AUTOSIZE);

namedWindow("Bilateral", CV\_WINDOW\_AUTOSIZE);

for (i = 1; i < MAX\_KERNEL\_LENGTH; i+=2)

{

blur(grey, Blur, Size(i, i), Point(-1, -1));

imshow("Blur", Blur);

GaussianBlur(grey, Gaussian, Size(i, i), 0, 0);

imshow("GaussianBlur", Gaussian);

medianBlur(grey, Median, i);

imshow("MedianBlur", Median);

bilateralFilter(grey, Bilateralfilter, i, 0, 0);

imshow("Bilateral", Bilateralfilter);

waitKey(0);

}

waitKey(0);

return 0;

}

1. **Colour detection(using mouse callback):**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include<opencv2\imgproc\imgproc.hpp>

#include <iostream>

using namespace std;

using namespace cv;

#define MAX\_KERNEL\_LENGTH 10

int Mx=0, My=0;

void mouse\_callback(int event, int x, int y, int, void\*)

{

if (event == CV\_EVENT\_LBUTTONDOWN)

{

Mx = x;

My = y;

}

}

int main()

{

Mat image = imread("G:\\colourful2.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat imageHSV(image.rows, image.cols, CV\_8UC3);

Mat colordetect(image.rows, image.cols, CV\_8UC1);

int i, j;

int h, s, v, h1, s1, v1;

cvtColor(image, imageHSV, CV\_BGR2HSV);

namedWindow("Input", CV\_WINDOW\_AUTOSIZE);

namedWindow("Color", CV\_WINDOW\_AUTOSIZE);

int Hthres, Sthres, Vthres;

setMouseCallback("Input", mouse\_callback, NULL);

createTrackbar("Hue Thres", "Color", &Hthres, 15);

createTrackbar("Sat Thres", "Color", &Sthres, 70);

createTrackbar("Val Thres", "Color", &Vthres, 200);

while (1)

{

imshow("Input", image);

h = imageHSV.at<Vec3b>(My, Mx)[0];

s = imageHSV.at<Vec3b>(My, Mx)[1];

v = imageHSV.at<Vec3b>(My, Mx)[2];

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

h1 = imageHSV.at<Vec3b>(i, j)[0];

s1 = imageHSV.at<Vec3b>(i, j)[1];

v1 = imageHSV.at<Vec3b>(i, j)[2];

if (h1<=(h + Hthres) && h1>=(h - Hthres) && s1<=(s + Sthres) && s1>=(s - Sthres) && v1<=(v+Vthres) && v1>=(v-Vthres))

colordetect.at<uchar>(i, j) = 255;

else

colordetect.at<uchar>(i, j) = 0;

}

}

imshow("Color", colordetect);

if (waitKey(30) == 27)

break;

}

waitKey(0);

return 0;

}

1. **Blob detection(run the following function on binary images to detect white blobs on black surroundings):**

#include<queue>

using namespace std;

using namespace cv;

typedef struct {

int x, y;

}pt;

typedef struct {

int min\_x, max\_x;

int min\_y, max\_y;

int cen\_x, cen\_y;

int n\_pixels;

int ID;

}blob;

void GetBlobs(Mat img, vector<blob> blobs)

{

int i, j, k, l, r = img.rows, c = img.cols, id = 1;

vector<vector<int> > pixel\_ID(r, vector<int>(c, -1)); //ID of unvisited pixel is -1

queue<pt> open\_list;

for (i = 1; i<r - 1; i++){

for (j = 1; j<c - 1; j++){

if (pixel\_ID[i][j]>-1)

continue;

if (img.at<uchar>(i, j) == 0)

{

pixel\_ID[i][j] = 0;

continue;

}

pt start = { j, i };

open\_list.push(start);

int sum\_x = 0, sum\_y = 0, n\_pixels = 0, max\_x = 0, max\_y = 0;

int min\_x = c + 1, min\_y = r + 1;

while (!open\_list.empty()){

//Dequeue the element at the head of the queue

pt top = open\_list.front();

open\_list.pop();

pixel\_ID[top.y][top.x] = id;

n\_pixels++;

//To obtain the bounding box of the blob w.r.t the original image

min\_x = (top.x<min\_x) ? top.x : min\_x;

min\_y = (top.y<min\_y) ? top.y : min\_y;

max\_x = (top.x>max\_x) ? top.x : max\_x;

max\_y = (top.y>max\_y) ? top.y : max\_y;

sum\_y += top.y; sum\_x += top.x;

//Add the 8-connected neighbours that are yet to be visited, to the queue

for (k = top.y - 1; k <= top.y + 1; k++){

for (l = top.x - 1; l <= top.x + 1; l++){

if (img.at<uchar>(k, l) == 0 || pixel\_ID[k][l]>-1)

continue;

pt next = { l, k };

pixel\_ID[k][l] = id;

open\_list.push(next);

}

}

}

if (n\_pixels < 20) //At least 20 pixels

continue;

blob nextcentre = { min\_x, max\_x, min\_y, max\_y, sum\_x / n\_pixels, sum\_y / n\_pixels, id };

blobs.push\_back(nextcentre);

id++;

}

}

cout << blobs.size();

}

1. **Video Trackbar:**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

VideoCapture vid("G:\\images\\ClayCarUpdate2.mp4");

int pos = 0;

namedWindow("Output", CV\_WND\_PROP\_FULLSCREEN);

int fps = vid.get(CV\_CAP\_PROP\_FPS);

int maxtime = (vid.get(CV\_CAP\_PROP\_FRAME\_COUNT)) / fps;

cvCreateTrackbar("Position", "Output", &pos, 1000\*maxtime);

while (1)

{

Mat image;

vid.read(image);

imshow("Output", image);

setTrackbarPos("Position", "Output", pos);

vid.set(CV\_CAP\_PROP\_POS\_MSEC, pos);

pos += 1000 / fps;

int ikey = waitKey(30);

if (ikey == 27)

break;

}

waitKey(0);

return 0;

}

1. **Arduino code:**

int pin\_e1 = 52;

int pin\_in1\_1 = 50;

int pin\_in1\_2 = 48;

int pin\_e2 = 42;

int pin\_in2\_1 = 40;

int pin\_in2\_2 = 38;

void full\_right()

{

digitalWrite(pin\_in1\_1, HIGH);

digitalWrite(pin\_in1\_2, LOW);

digitalWrite(pin\_in2\_1, HIGH);

digitalWrite(pin\_in2\_2, LOW);

}

void full\_left()

{

digitalWrite(pin\_in1\_1, LOW);

digitalWrite(pin\_in1\_2, HIGH);

digitalWrite(pin\_in2\_1, LOW);

digitalWrite(pin\_in2\_2, HIGH);

}

void backward()

{

digitalWrite(pin\_in1\_1, LOW);

digitalWrite(pin\_in1\_2, HIGH);

digitalWrite(pin\_in2\_1, HIGH);

digitalWrite(pin\_in2\_2, LOW);

}

void forward()

{

digitalWrite(pin\_in1\_1, HIGH);

digitalWrite(pin\_in1\_2, LOW);

digitalWrite(pin\_in2\_1, LOW);

digitalWrite(pin\_in2\_2, HIGH);

}

void left()

{

digitalWrite(pin\_in1\_1, HIGH);

digitalWrite(pin\_in1\_2, LOW);

digitalWrite(pin\_in2\_1, LOW);

digitalWrite(pin\_in2\_2, LOW);

}

void right()

{

digitalWrite(pin\_in1\_1, LOW);

digitalWrite(pin\_in1\_2, LOW);

digitalWrite(pin\_in2\_1, HIGH);

digitalWrite(pin\_in2\_2, LOW);

}

void stall()

{

digitalWrite(pin\_in1\_1, LOW);

digitalWrite(pin\_in1\_2, LOW);

digitalWrite(pin\_in2\_1, LOW);

digitalWrite(pin\_in2\_2, LOW);

}

void freerun()

{

digitalWrite(pin\_e1,LOW);

digitalWrite(pin\_e2,LOW);

}

void setup()

{

Serial.begin(9600);

pinMode(pin\_e1, OUTPUT);

pinMode(pin\_in1\_1, OUTPUT);

pinMode(pin\_in1\_2, OUTPUT);

digitalWrite(pin\_e1, HIGH);

pinMode(pin\_e2, OUTPUT);

pinMode(pin\_in2\_1, OUTPUT);

pinMode(pin\_in2\_2, OUTPUT);

digitalWrite(pin\_e2, HIGH);

pinMode(22,OUTPUT);

pinMode(24,OUTPUT);

}

void loop()

{

if(Serial.read()=='w')

forward();

else if(Serial.read()=='s')

backward();

else if(Serial.read()=='a')

left();

else if(Serial.read()=='d')

right();

else

stall();

}

1. **Shape Detection (circle, triangle, square, rectangle):**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include<opencv2\imgproc\imgproc.hpp>

#include <iostream>

#include<queue>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\images\\shapes.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat grey(image.rows, image.cols, CV\_8UC1);

Mat binary(image.rows, image.cols, CV\_8UC1);

Mat cannyedge(image.rows, image.cols, CV\_8UC1);

int i, j;

int b, g, r;

//making grayscale

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (b + g + r) / 3;

}

}

//histogram to make binary

int freq[256];

for (i = 0; i < 256; ++i)

freq[i] = 0;

for (i = 0; i < image.rows; ++i)

for (j = 0; j < image.cols; ++j)

freq[grey.at<uchar>(i, j)]++;

int sum = 0, thres1 = 0;

while (sum < (image.rows\*image.cols) / 2)

sum += freq[thres1++];

int threslow = thres1 / 3;

int threshigh = 2 \* thres1 / 3;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

if (grey.at<uchar>(i, j) > thres1)

binary.at<uchar>(i, j) = 255;

else

binary.at<uchar>(i, j) = 0;

}

}

//find edges

Canny(grey, cannyedge, threslow, threshigh, 3);

//find contours

vector<vector<Point>> contours0;

findContours(cannyedge, contours0, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE);

vector<Point> approx;

for (int k = 0; k < contours0.size(); ++k)

{

approxPolyDP(Mat(contours0[k]), approx, arcLength(contours0[k], true)\*0.02, true);

if (approx.size() == 3)

{

cout << k + 1 << "is a triangle" << endl;

cout << approx[0].x << " " << approx[1].x << " " << approx[2].x << " " << endl;

cout << approx[0].y << " " << approx[1].y << " " << approx[2].y << " " << endl;

continue;

}

if (approx.size()>7)

{

cout << k + 1 << "is a circle" << endl;

continue;

}

if (approx.size() == 4)

{

float a = sqrt((approx[0].x - approx[1].x)\*(approx[0].x - approx[1].x) + (approx[0].y - approx[1].y)\*(approx[0].y - approx[1].y));

float b = sqrt((approx[1].x - approx[2].x)\*(approx[1].x - approx[2].x) + (approx[1].y - approx[2].y)\*(approx[1].y - approx[2].y));

float c = sqrt((approx[2].x - approx[3].x)\*(approx[2].x - approx[3].x) + (approx[2].y - approx[3].y)\*(approx[2].y - approx[3].y));

float d = sqrt((approx[0].x - approx[3].x)\*(approx[0].x - approx[3].x) + (approx[0].y - approx[3].y)\*(approx[0].y - approx[3].y));

if (fabs(a - b) <= 2)

cout << k + 1 << "is a square" << endl;

else

cout << k + 1 << "is a rectangle" << endl;

cout << approx[0].x << " " << approx[1].x << " " << approx[2].x << " " << approx[3].x << endl;

cout << approx[0].y << " " << approx[1].y << " " << approx[2].y << " " << approx[3].y << endl;

}

}

namedWindow("Input", CV\_WINDOW\_AUTOSIZE);

imshow("Input", image);

waitKey(0);

return 0;

}

1. **Hough lines:**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include<opencv2\imgproc\imgproc.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("G:\\images\\blobs2.jpg", CV\_LOAD\_IMAGE\_COLOR);

//first find canny edges

Mat grey(image.rows, image.cols, CV\_8UC1);

int i, j, k, l;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (33 \* b + 56 \* g + 11 \* r) / 100;

}

}

int freq[256];

for (i = 0; i < 256; ++i)

freq[i] = 0;

for (i = 0; i < image.rows; ++i)

for (j = 0; j < image.cols; ++j)

freq[grey.at<uchar>(i, j)]++;

Mat cannyedge(image.rows, image.cols, CV\_8UC1);

int sum = 0, thres = 0;

while (sum < (image.rows\*image.cols) / 2)

sum += freq[thres++];

int threslow = thres / 3;

int threshigh = thres \* 2 / 3;

Canny(grey, cannyedge, threslow, threshigh, 3);

//cannyedge has white edges on black background

//use HoughLines to find lines

namedWindow("Input", CV\_WINDOW\_AUTOSIZE);

imshow("Input", cannyedge);

namedWindow("Output", CV\_WINDOW\_AUTOSIZE);

Mat output(image.rows, image.cols, CV\_8UC1);

vector<Vec2f> lines; //lines is a vector of r rows and 2 columns storing float values

int thres2 = 40;

createTrackbar("Threshold", "Output", &thres2, sqrt(image.rows\*image.rows + image.cols\*image.cols));

while (1)

{

HoughLines(cannyedge, lines, 1, CV\_PI / 180, thres2);

for (i = 0; i < image.rows; ++i)

for (j = 0; j < image.cols; ++j)

output.at<uchar>(i, j) = 255;

for (i = 0; i < lines.size(); ++i)

{

double rho = lines[i][0];

double theta = lines[i][1];

Point pt1, pt2;

double a = cos(theta), b = sin(theta);

double x0 = a\*rho, y0 = b\*rho;

pt1.x = cvRound(x0 + 1500 \* (-b));

pt1.y = cvRound(y0 + 1500 \* (a));

pt2.x = cvRound(x0 - 1500 \* (-b));

pt2.y = cvRound(y0 - 1500 \* (a));

line(output, pt1, pt2, Scalar(0, 0, 255), 3, CV\_AA);

/\*for (k = 0; k < image.rows; ++k)

{

for (l = 0; l < image.cols; ++l)

{

if (cannyedge.at<uchar>(k, l) == 0)

continue;

if (fabs(rho - l \* a - k \* b) <= 0.05)

{

int m, n;

for (m = k - 1; m <= k + 1; ++m)

{

for (n = l - 1; n <= l + 1; ++n)

{

if (m>=0 && m<image.rows && n>=0 && n<image.cols)

output.at<uchar>(m, n) = 0;

}

}

}

}

}\*/

}

imshow("Output", output);

if (waitKey(30) == 27)

break;

}

waitKey(0);

return 0;

}