Documentation for the workshop

The source code for all the programs are available on GitHub @pranitbauva1997

PS 1 : A video is given in which there are many spaces which are moving in a race. Only first 3 seconds are shown and you are to predict who will win the race.

PS 2 : Detect the direction of finger from camera and send signals to bot to move accordingly.

Day 1:-

Major portion of the day was spent in configuring the environment for opencv. Windows users installed Microsoft Visual Studio and configured it to use opencv library for their projects. Linux users were given commands to install the OpenCV library. After the break we started with OpenCV. The things covered in the class are :-

- ➔ Open a RGB (BGR) image
- ➔ Convert the coloured image to grayscale
- ➔ Make a simple RGB (BGR) image
- ➔ Make a tri color image (horizontal splitting)
- ➔ Independant mirror image
- ➔ Split mirror image
- ➔ Create a chessboard

The source codes for these are :-

- image_open.cpp : To open a BGR image

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
        if( argc != 2)
        {
        cout <<" Usage: display_image ImageToLoadAndDisplay" << endl;
        return -1;
        }

        Mat image;
        image = imread(argv[1], CV_LOAD_IMAGE_COLOR);   // Read the file

        if(! image.data )                              // Check for invalid input
        {
        cout <<  "Could not open or find the image" << std::endl ;
        return -1;
        }

        namedWindow( "Display window", WINDOW_AUTOSIZE );// Create a window for display.
        imshow( "Display window", image );                // Show our image inside it.

        waitKey(0);                                       // Wait for a keystroke in the
window
        return 0;
}
```

- plain_grayscale.cpp : To make a plain grayscale image

```cpp
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>

using namespace std;
using namespace cv;

int main(){
	Mat image(200, 400, CV_8UC1, Scalar(45));
	imshow("Display Window", image);

	waitKey(0);
	return 0;
}
```

- plain_rgb.cpp : To make a plain RGB (BGR) image

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<iostream>

using namespace std;
using namespace cv;

int main(){
        Mat image(200, 400, CV_8UC3, Scalar(0,255,0));
        imshow("Display Window", image);

        waitKey(0);
        return 0;
}
```

- tri_color.cpp : To make a tri colour image which will display red, blue and green in 3 continuous rows

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<iostream>

using namespace std;
using namespace cv;

int main(){
	Mat image(200, 400, CV_8UC3, Scalar(0,255,0));
	imshow("Display Window", image);

	waitKey(0);
	return 0;
}
```

- mirror_image.cpp : To display mirror image in an independent window

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<iostream>

using namespace std;
using namespace cv;

int main(int argc, char **argv){
	Mat straight, mirror;
	straight = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	mirror = imread(argv[1], CV_LOAD_IMAGE_COLOR);

	namedWindow("Straight", WINDOW_AUTOSIZE);
	imshow("Straight", straight);

	int i,j;
	for(i = 0; i < mirror.rows; i++){
	for(j = 0; j < mirror.cols; j++){
		mirror.at<Vec3b>(i, j) = straight.at<Vec3b>(i, mirror.cols-j - 1);
	}
	}

	namedWindow("Mirror", WINDOW_AUTOSIZE);
	imshow("Mirror", mirror);

	waitKey(0);
	return 0;
}
```

- split_mirror.cpp : To show a split mirror image

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

int main(int argc, char **argv){
        Mat image = imread(argv[1], 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, n - j - 1) = im.at<Vec3b>(i, j);

        namedWindow("Split Mirror", WINDOW_AUTOSIZE);
        imshow("Split Mirror", image);
        waitKey(0);
        return 0;
}
```

- chessboard.cpp : To make a chessboard image

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<iostream>

using namespace std;
using namespace cv;

int main(){
	Mat board(900, 900, CV_8UC1, Scalar(0));
	int i, j, x, y;
	x = y = 0;

	for(x = 0; x < board.rows; x += board.rows/15)
	for(y = 0; y < board.cols; y += board.cols/15)
		for(i = 0; i < 30; i++)
		for(j = 0; j < 30; j++)
			board.at<uchar>(x + i, y + j) = 255;

	for(x = 30; x < board.rows; x += board.rows/15)
	for(y = 30; y < board.cols; y += board.cols/15)
		for(i = 0; i < 30; i++)
		for(j = 0; j < 30; j++)
			board.at<uchar>(x + i, y + j) = 255;



	namedWindow("Chess Board", WINDOW_AUTOSIZE);
	imshow("Chess Board", board);
	waitKey(0);

	return 0;
}
```

Day 2 :-

From this day we started manipulating individual pixels numbers.

- ➔ BGR to grayscale using average and weighted average method
- ➔ Conversion of a grayscale image to binary image
- ➔ Size multiplier to increase the size of the image by a multiplier thus dilating pixels
- ➔ Histogram to show no of pixels vs intensity of pixel for grayscale and BGR image
- ➔ Using track bars to adjust weight of colours in BGR to grayscale conversion
- ➔ Using track bars to do thresholding in binary image
- ➔ Colour extractor to extract a particular colour with tolerance track bars

- bgr_grayscale_conversion.cpp : To convert a bgr image to grayscale

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>

using namespace std;
using namespace cv;

int main(int argc, char **argv){
	Mat img1;
	img1 = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat img2(img1.rows, img1.cols, CV_8UC1, Scalar(0));
	Mat img3(img1.rows, img1.cols, CV_8UC1, Scalar(0));

	int i, j;
	for(i = 0; i < img1.rows; i++){
	for(j = 0; j < img1.cols; j++){
		img2.at<uchar>(i, j) = (img1.at<Vec3b>(i, j)[0] + img1.at<Vec3b>(i,j)[1] +
img1.at<Vec3b>(i, j)[2]) / 3;
	}
	}
	for(i = 0; i < img1.rows; i++){
	for(j = 0; j < img1.cols; j++){
		img3.at<uchar>(i, j) = 0.11*img1.at<Vec3b>(i, j)[0] + 0.59*img1.at<Vec3b>(i,
j)[1] + 0.3*img1.at<Vec3b>(i, j)[2];
	}
	}

	namedWindow("Normal", WINDOW_AUTOSIZE);
	imshow("Normal", img1);
	namedWindow("Average", WINDOW_AUTOSIZE);
	imshow("Average", img2);
	namedWindow("Weighted Average", WINDOW_AUTOSIZE);
	imshow("Weighted Average", img3);

	waitKey(0);
	return 0;
}
```

- binary_image.cpp : Convert a grayscale image to binary image

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>

using namespace std;
using namespace cv;

int main(int argc, char **argv){
	Mat img;
	img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat img1(img.rows, img.cols, CV_8UC1, Scalar(0));
	Mat img_binary(img.rows, img.cols, CV_8UC1, Scalar(0));
	cvtColor(img, img1, CV_BGR2GRAY);

	int i, j;
	for(i = 0; i < img.rows; i++)
	for(j = 0; j < img.cols; j++)
		if(img1.at<uchar>(i, j) > 127)
		img_binary.at<uchar>(i, j) = 255;

	namedWindow("Normal", WINDOW_AUTOSIZE);
	imshow("Normal", img);
	namedWindow("Binary", WINDOW_AUTOSIZE);
	imshow("Binary", img_binary);

	waitKey(0);
	return 0;
}
```

- double_image.cpp : To double the size of the image

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>

using namespace cv;
using namespace std;

int main(int argc, char **argv){
	Mat img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat img_d(img.rows*2, img.cols*2, CV_8UC3, Scalar(0, 0, 0));

	int x, y, i, j;
	for(x = 0; x < img.rows; x++)
	for(y = 0; y < img.cols; y++)
		for(i = 0; i < 2; i++)
		for(j = 0; j < 2; j++)
			img_d.at<Vec3b>(x*2+i, y*2+j) = img.at<Vec3b>(x, y);

	namedWindow("Single", WINDOW_AUTOSIZE);
	namedWindow("Double", WINDOW_AUTOSIZE);
	imshow("Single", img);
	imshow("Double", img_d);

	waitKey(0);
	return 0;
}
```

- histogram.cpp : To make a histogram of no of pixels vs intensity of pixels

```
#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(int argc, char **argv){
	Mat img;
	img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat img1(img.rows, img.cols, CV_8UC1, Scalar(0));
	cvtColor(img, img1, CV_BGR2GRAY);
	Mat histo(500, 500, CV_8UC1, Scalar(255));

	long intensity[256];
	int i, j;


	for(i = 0; i < 256; i++)
	intensity[i] = 0;

	for(i = 0; i < img1.rows; i++)
	for(j = 0; j < img1.cols; j++)
		intensity[img1.at<uchar>(i, j)]++;

	for(i = 0; i < 256; i++)
	intensity[i] /= 100;

	for(i = 0; i < 256; i++)
	histo.at<uchar>(400-intensity[i], i) = 0;

	namedWindow("Histogram", WINDOW_AUTOSIZE);
	imshow("Histogram", histo);

	waitKey(0);
	return 0;
}
```

- binary_image_trackbar.cpp : Track bars to threshold the binary image

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>

using namespace std;
using namespace cv;

int main(int argc, char **argv){
	Mat img;
	img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat img1(img.rows, img.cols, CV_8UC1, Scalar(0));
	cvtColor(img, img1, CV_BGR2GRAY);
	Mat final_img(img.rows, img.cols, CV_8UC1, Scalar(0));

	namedWindow("Binary Image", WINDOW_AUTOSIZE);
	int threshold = 127;
	createTrackbar("track1", "Binary Image", &threshold, 255);
	int i, j;
	while(1){
	for(i = 0; i < img1.rows; i++){
		for(j = 0; j < img1.cols; j++){
		if(img1.at<uchar>(i, j) > threshold)
			final_img.at<uchar>(i, j) = 255;
		else
			final_img.at<uchar>(i, j) = 0;
		}
	}
	imshow("Binary Image", final_img);
	if(waitKey(10) == 'q')
		break;
	}

	return 0;
}
```

- bgr_grayscale_conversion_trackbar.cpp : Change weights by track bars

```
 #include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>

using namespace std;
using namespace cv;

int main(int argc, char **argv){
	Mat img;
	img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat final_img(img.rows, img.cols, CV_8UC3, Scalar(0, 0, 0));

	namedWindow("Hey", WINDOW_AUTOSIZE);
	int a, b, c;
	a = b = c = 1;
	int i, j;
	createTrackbar("blue", "Hey", &a, 100);
	createTrackbar("green", "Hey", &b, 100);
	createTrackbar("red", "Hey", &c, 100);
	while(1){
	for(i = 0; i < img.rows; i++){
		for(j = 0; j < img.cols; j++){
		final_img.at<Vec3b>(i, j)[0] = a*img.at<Vec3b>(i, j)[0]/100;
		final_img.at<Vec3b>(i, j)[1] = b*img.at<Vec3b>(i, j)[1]/100;
		final_img.at<Vec3b>(i, j)[2] = c*img.at<Vec3b>(i, j)[2]/100;
		}
	}
	imshow("Hey", final_img);
	if(waitKey(10) == 'q')
		break;
	}

	return 0;
}
```

- color_extractor.cpp : To extract a particular colour with tolerance values using track bars

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>

using namespace std;
using namespace cv;

int range(int x, int y, int tol){
	if(x > y - tol && x < y + tol)
	return 1;
	else if(y > x - tol && y < x + tol)
	return 1;
	else
	return 0;
}

int main(int argc, char **argv){
	Mat img;
	img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat final_img(img.rows, img.cols, CV_8UC3, Scalar(0, 0, 0));

	namedWindow("Color Extractor", WINDOW_AUTOSIZE);
	int red, blue, green, tol;
	red = blue = green = tol = 0;
	int i, j;

	createTrackbar("Blue", "Color Extractor", &blue, 255);
	createTrackbar("Green", "Color Extractor", &green, 255);
	createTrackbar("Red", "Color Extractor", &red, 255);
	createTrackbar("Tolerance", "Color Extractor", &tol, 50);
	while(1){
	for(i = 0; i < img.rows; i++){
		for(j = 0; j < img.cols; j++){
		if(range(img.at<Vec3b>(i, j)[0], blue, tol) && range(img.at<Vec3b>(i, j)[1],
green, tol) && range(img.at<Vec3b>(i, j)[2], red, tol)){
			final_img.at<Vec3b>(i, j) = img.at<Vec3b>(i, j);
		}
		else{
			final_img.at<Vec3b>(i, j)[0] = 255;
			final_img.at<Vec3b>(i, j)[1] = 255;
			final_img.at<Vec3b>(i, j)[2] = 255;
		}
		}
	}
	imshow("Color Extractor", final_img);
	if(waitKey(10) == 'q')
		break;
	}
	return 0;
}
```

Day 3 :

We started manipulating kernels, applying linear filters, edge detection, dilation and erosion.
Things done :-

- ➔ Apply linear filter with a variable kernel size
- ➔ Apply linear filter using library function
- ➔ Use Gaussian filter with a fixed kernel size
- ➔ Implement edge detection using simple max and min in kernel
- ➔ Implement edge detection using Sobel Operator
- ➔ Implement edge detection using various other kernels (like Scharr)
- ➔ Use Canny edge detection
- ➔ Remove noise by dilation and erosion

- linear_filter.cpp : Apply a linear filter on an image

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<iostream>

using namespace std;
using namespace cv;

int main(int argc, char **argv){
	Mat img;
	img = imread("bikini.jpg", CV_LOAD_IMAGE_COLOR);
	//Mat final_img(img.rows, img.cols, CV_8UC3, Scalar(0, 0, 0));
	int kernel_size = 3;
	//float kernel = (kernel_size*kernel_size);
	float sumb, sumg, sumr;
	int i, j, x, y;
	float kernel[3][3]={{1.0/16.0, 1.0/8.0, 1.0/16.0},
		{1.0/8.0, 1.0/4.0, 1.0/8.0},
		{1.0/16.0, 1.0/8.0, 1.0/16.0}};

	for(i = 1; i < img.rows-1; i++){
	for(j = 1; j < img.cols-1; j++){
	sumb = sumg = sumr = 0.0;
	for(x = -1; x < kernel_size-1; x++){
	for(y = -1; y < kernel_size-1; y++){
		sumb += (int)(img.at<Vec3b>(i + x, j + y)[0] * kernel[1+x][y+1]);
		sumg += (int)(img.at<Vec3b>(i + x, j + y)[1] * kernel[1+x][1+y]);
		sumr += (int)(img.at<Vec3b>(i + x, j + y)[2] * kernel[1+x][1+y]);
	}
	}
	img.at<Vec3b>(i, j)[0] = sumb;
	img.at<Vec3b>(i, j)[1] = sumg;
	img.at<Vec3b>(i, j)[2] = sumr;
	}
	}

	namedWindow("Filter", WINDOW_AUTOSIZE);
	imshow("Filter", img);
	waitKey(0);
}
```

- linear_filter_lib_func.cpp : Apply linear filter using library function

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;

/** @function main */
int main ( int argc, char** argv )
{
  /// Declare variables
  Mat src, dst;

  Mat kernel;
  Point anchor;
  double delta;
  int ddepth;
  int kernel_size;
  char* window_name = "filter2D Demo";

  int c;

  /// Load an image
  src = imread( argv[1] );

  if( !src.data )
  { return -1; }

  /// Create window
  namedWindow( window_name, CV_WINDOW_AUTOSIZE );

  /// Initialize arguments for the filter
  anchor = Point( -1, -1 );
  delta = 0;
  ddepth = -1;

  /// Loop - Will filter the image with different kernel sizes each 0.5 seconds
  int ind = 0;
  while( true )
        {
        c = waitKey(500);
        /// Press 'ESC' to exit the program
        if( (char)c == 27 )
        { break; }

        /// Update kernel size for a normalized box filter
        kernel_size = 3 + 2*( ind%5 );
        kernel = Mat::ones( kernel_size, kernel_size, CV_32F )/
(float)(kernel_size*kernel_size);

        /// Apply filter
        filter2D(src, dst, ddepth , kernel, anchor, delta, BORDER_DEFAULT );
        imshow( window_name, dst );
        ind++;
        }

  return 0;
}
```

- edge_detection_simple.cpp : Implement a simple edge detection using max and min in kernel

```
#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(int argc, char **argv){
	Mat img;
	img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat final_img(img.rows, img.cols, CV_8UC1, Scalar(255));
	Mat img1(img.rows, img.cols, CV_8UC1, Scalar(0));
	cvtColor(img, img1, CV_BGR2GRAY);

	int threshold = 50;
	int i, j, x, y;
	int max, min;
	namedWindow("Edge Detection", WINDOW_AUTOSIZE);
	namedWindow("Original", WINDOW_AUTOSIZE);
	imshow("Original", img1);
	createTrackbar("Threshold", "Edge Detection", &threshold, 255);
	while(1){
	for(i = 1; i < img.rows - 1; i++){
		for(j = 1; j < img.cols - 1; j++){
		for(x = i - 1; x < i + 1; x++){
			for(y = j -1; y < j + 1; y++){
			max = img1.at<uchar>(i - 1, j - 1);
			if(img1.at<uchar>(x, y) > max)
				max = img1.at<uchar>(x, y);
			min = img1.at<uchar>(i - 1, j - 1);
			if(img1.at<uchar>(x, y) < min)
				min = img1.at<uchar>(x, y);
			}
		}
		if( (max - min) > threshold )
			final_img.at<uchar>(i, j) = 0;
		}
	}
	imshow("Edge Detection", final_img);
	if(waitKey(20) == 'q')
		break;
	for(i = 0; i < img.rows; i++)
		for(j = 0; j < img.cols; j++)
		final_img.at<uchar>(i, j) = 255;
	}

	return 0;
}
```

- edge_detection.cpp : Implement edge detection using directional gradients

```
#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(int argc, char **argv){
	Mat img;
	img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat final_img(img.rows, img.cols, CV_8UC1, Scalar(255));
	Mat img1(img.rows, img.cols, CV_8UC1, Scalar(0));
	cvtColor(img, img1, CV_BGR2GRAY);

	int threshold = 50;
	int i, j, x, y;
	int max, min;
	namedWindow("Edge Detection", WINDOW_AUTOSIZE);
	namedWindow("Original", WINDOW_AUTOSIZE);
	imshow("Original", img1);
	createTrackbar("Threshold", "Edge Detection", &threshold, 255);
	while(1){
	for(i = 1; i < img.rows - 1; i++){
		for(j = 1; j < img.cols - 1; j++){
		for(x = i - 1; x < i + 1; x++){
			for(y = j -1; y < j + 1; y++){
			max = img1.at<uchar>(i - 1, j - 1);
			if(img1.at<uchar>(x, y) > max)
				max = img1.at<uchar>(x, y);
			min = img1.at<uchar>(i - 1, j - 1);
			if(img1.at<uchar>(x, y) < min)
				min = img1.at<uchar>(x, y);
			}
		}
		if( (max - min) > threshold )
			final_img.at<uchar>(i, j) = 0;
		}
	}
	imshow("Edge Detection", final_img);
	if(waitKey(20) == 'q')
		break;
	for(i = 0; i < img.rows; i++)
		for(j = 0; j < img.cols; j++)
		final_img.at<uchar>(i, j) = 255;
	}

	return 0;
}
```

- dilation_erosion.cpp : Noise removal by dilating and then eroding the image

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>

using namespace std;
using namespace cv;

int main(int argc, char **argv){
	Mat img;
	img = imread(argv[1], WINDOW_AUTOSIZE);
	Mat img1(img.rows, img.cols, CV_8UC1, Scalar(0));
	Mat img2(img.rows, img.cols, CV_8UC1, Scalar(0));
	Mat final_img(img.rows, img.cols, CV_8UC1, Scalar(255));
	cvtColor(img, img1, CV_BGR2GRAY);

	int i, j, x, y, white, black;
	// Binary image
	for(i = 0; i < img1.rows; i++)
	for(j = 0; j < img1.cols; j++)
		if(img1.at<uchar>(i, j) > 127)
		img2.at<uchar>(i, j) = 255;

	namedWindow("Binary", WINDOW_AUTOSIZE);
	imshow("Binary", img1);
	for(i = 1; i < img2.rows-1; i++){
	for(j = 1; j < img2.cols-1; j++){
		white = black = 0;
		for(x = -1; x < 2; x++){
		for(y = -1; y < 2; y++){
			if(img2.at<uchar>(i + x, j + y) == 255)
			black++;
			else
			white++;
		}
		}
		if(white > black){
		for(x = -1; x < 2; x++){
			for(y = -1; y < 2; y++){
			final_img.at<uchar>(i + x, j + y) = 0;
			}
		}
		}
	}
	}
	for(i = 1; i < img2.rows-1; i++){
	for(j = 1; j < img2.cols-1; j++){
		white = black = 0;
		for(x = -1; x < 2; x++){
		for(y = -1; y < 2; y++){
			if(img2.at<uchar>(i + x, j + y) == 255)
			black++;
			else
			white++;
		}
		}
		if(black > 0){
		for(x = -1; x < 2; x++){
			for(y = -1; y < 2; y++){
			final_img.at<uchar>(i + x, j + y) = 255;
```

```
                        }
                }
                }
        }
        }

        namedWindow("Dilate", WINDOW_AUTOSIZE);
        imshow("Dilate", final_img);

        waitKey(0);
        return 0;
}
```

- canny_filter.cpp : Use library function to show canny filter

```
#include<opencv2/imgproc/imgproc.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<stdlib.h>
#include<stdio.h>
#include<opencv2/core/core.hpp>

using namespace std;
using namespace cv;

Mat src, src_gray;
Mat dst, detected_edges;

int edgeThresh = 1;
int lowThreshold;
int const max_lowThreshold = 100;
int ratio = 3;
int kernel_size = 3;

void CannyThreshold(int, void*){
	// Reduce noise with a kernel 3x3
	blur(src_gray, detected_edges, Size(3, 3));

	// Canny detector
	Canny(detected_edges, detected_edges, lowThreshold, lowThreshold*ratio,
kernel_size);

	// Using Canny's out as a mask, we display our result
	dst = Scalar::all(0);

	src.copyTo(dst, detected_edges);
	imshow("Edge Map", dst);
}

int main(int argc, char **argv){
	// Load an image
	src = imread(argv[1]);

	// Create a matrix of the same type and size as src (for dst)
	dst.create(src.size(), src.type());

	// Convert the image to grayscale
	cvtColor(src, src_gray, CV_BGR2GRAY);

	// Create a window
	namedWindow("Edge Map", CV_WINDOW_AUTOSIZE);

	// Create a Trackbar for user to enter threshold
	createTrackbar("Min Threshold:", "Edge Map", &lowThreshold, max_lowThreshold,
CannyThreshold);

	// Show the image
	CannyThreshold(0, 0);

	waitKey(0);
	return 0;
}
```

Day 4 :

We learned some concepts about Stacks and Queues. We also learned some introductory things about graph theory and about Depth First Search and Breadth First Search. Then we used these concepts in blob detection. We also learned about hough transform for any number of lines. We implemented both of them and also used library functions to do the same. Also learned about different colour spaces, the difference between them and where to use which one to maximize the benefit.

Things taught :-

- ➔ Introductory Graph Theory
- ➔ Stacks and Queues
- ➔ DFS & BFS
- ➔ Single Blob detection
- ➔ Multiple blob detection
- ➔ Own implementation hough transform
- ➔ Library function for hough transform
- ➔ Different Colour Spaces
- ➔ BGR to HSV

- blob_detection.cpp : Detect a single blob using DFS

```
#include<iostream>
#include<stack>
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>

using namespace std;
using namespace cv;

typedef struct{
	int x;
	int y;
} point;

Mat img;
int visited[1000][1000];

void DFS(point p){
	int i, j;
	stack<point> s;
	s.push(p);
	point temp;
	while(!s.empty()){
	point A = s.top();
	s.pop();
	visited[A.x][A.y] = 1;

	// Finding other nodes
	for(i = -1; i < 2; i++){
		for(j = -1; j < 2; j++){
		temp.x = A.x + i;
		temp.y = A.x + j;
		if((img.at<uchar>(temp.x, temp.y) == 0) && (visited[temp.x][temp.y] == 0))
			s.push(temp);
		}
	}
	}
}

int main(int argc, char **argv){
	img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat img1(img.rows, img.cols, CV_8UC1, Scalar(0));
	Mat img2(img.rows, img.cols, CV_8UC1, Scalar(0));
	Mat final_img(img.rows, img.cols, CV_8UC3, Scalar(0, 0, 0));
	cvtColor(img, img1, CV_BGR2GRAY);
	int i, j;

	namedWindow("Original", WINDOW_AUTOSIZE);
	imshow("Original", img);
	namedWindow("Binary", WINDOW_AUTOSIZE);;
	// Binary image
	for(i = 0; i < img1.rows; i++)
	for(j = 0; j < img1.cols; j++)
		if(img1.at<uchar>(i, j) > 127)
		img2.at<uchar>(i, j) = 255;
	imshow("Binary", img2);
```

```
        // Initialise a array named visited
        for(i = 0; i < img.rows; i++)
        for(j = 0; j < img.cols; j++)
                visited[i][j] = 0;

        // Applying DFS
        point p;
        for(i = 1; i < img2.rows - 1; i++){
        for(j = 1; j < img2.cols - 1; j++){
                p.x = i;
                p.y = j;
                if((img2.at<uchar>(i, j) == 0) && visited[i][j] == 0)
                DFS(p);
        }
        }

        // Marking the blob red
        for(i = 0; i < img2.rows; i++)
        for(j = 0; j < img2.rows; j++)
                if(visited[i][j])
                final_img.at<Vec3b>(i, j)[2] = 255;

        namedWindow("YES", WINDOW_AUTOSIZE);
        imshow("YES", final_img);
        waitKey(0);
        return 0;
}
```

- hough_transform_lib.cpp : Hough Transform

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

using namespace cv;
using namespace std;

void help()
{
 cout << "\nThis program demonstrates line finding with the Hough transform.\n"
       "Usage:\n"
       "./houghlines <image_name>, Default is pic1.jpg\n" << endl;
}

int main(int argc, char** argv)
{
 const char* filename = argc >= 2 ? argv[1] : "pic1.jpg";

 Mat src = imread(filename, 0);
 if(src.empty())
 {
       help();
       cout << "can not open " << filename << endl;
       return -1;
 }

 Mat dst, cdst;
 Canny(src, dst, 50, 200, 3);
 cvtColor(dst, cdst, CV_GRAY2BGR);

 #if 0
  vector<Vec2f> lines;
  HoughLines(dst, lines, 1, CV_PI/180, 100, 0, 0 );

  for( size_t i = 0; i < lines.size(); i++ )
  {
       float rho = lines[i][0], 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 + 1000*(-b));
       pt1.y = cvRound(y0 + 1000*(a));
       pt2.x = cvRound(x0 - 1000*(-b));
       pt2.y = cvRound(y0 - 1000*(a));
       line( cdst, pt1, pt2, Scalar(0,0,255), 3, CV_AA);
  }
 #else
  vector<Vec4i> lines;
  HoughLinesP(dst, lines, 1, CV_PI/180, 50, 50, 10 );
  for( size_t i = 0; i < lines.size(); i++ )
  {
       Vec4i l = lines[i];
       line( cdst, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, CV_AA);
  }
 #endif
 imshow("source", src);
 imshow("detected lines", cdst);
```

```
 waitKey();

 return 0;
}
```

Day 5 :

We learned about video feeds. Video from a file and webcam. We learned to play videos, increase/decrease speed and how to skip frames. We then started working on our PS 1.

bgr_hsv.cpp : Convert from BGR to HSV colour space.

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>

using namespace cv;
using namespace std;

int main(int argc, char **argv){
	Mat img;
	img = imread(argv[1], CV_LOAD_IMAGE_COLOR);
	Mat img_hsv(img.rows, img.cols, CV_8UC3, Scalar(0, 0, 0));
	cvtColor(img, img_hsv, CV_BGR2HSV);

	namedWindow("BGR", WINDOW_AUTOSIZE);
	namedWindow("HSV", WINDOW_AUTOSIZE);
	imshow("BGR", img);
	imshow("HSV", img_hsv);
	waitKey(0);
	return 0;
}
```

- video_feed.cpp : Video feed in OpenCV

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include<iostream>
#include<string>
#include<iomanip>
#include<sstream>

using namespace std;
using namespace cv;

int main(int argc, char **argv){
        VideoCapture cap(argv[1]);

        // Check validity
        if(!cap.isOpened()){
        cout << "Sorry there is some problem with the video feed." << endl;
        return -1;
        }
        int i = 0;
        namedWindow("Video", WINDOW_AUTOSIZE);
        Mat frameReference, frameUnderTest;
        while(1){
        cap >> frameReference;
        cap.set(CV_CAP_PROP_POS_MSEC, 3000 * i);
        imshow("Video", frameReference);
        if(waitKey(50) == 'q')
                break;
        i++;
        }
}
```

- who_wins.cpp : Code for PS 1

```
#include<opencv2\opencv2.hpp>
#include<stdafx.h>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include<iostream>
#include<math.h>

using namespace std;
using namespace cv;

int range(int x, int y, int d){
    if(x >= y - d && x <= y + d)
        return 1;
    else if(y >= x - d && y <= x + d)
        return 1;
    else
        return 0;
}

vector<Point> corrector(vector<Point> v){
    vector<Point> l;
    int i, j, flag;
    for(i = 0; i < v.size(); i++){
        flag = 1;
        for(j = 0; j < l.size(); j++){
            if(range(l[j].x, v[i].x, 5))
                flag = 0;
        }
        if(flag)
            l.push_back(v[i]);
    }


    return l;
}

int main(){
    Mat src1, src2;
    Mat src1_gray, src2_gray;
    Mat canny1, canny2;
    vector<vector<Point> > contours1, contours2;
    vector<Vec4i> hierarchy1, hierarchy2;
    VideoCapture cap("video.avi");
    float t1, t2;
    t1 = 900;
    t2 = 1500;

    if(!cap.isOpened()){
        cout << "Sorry there was some problem playing the video" << endl;
        return -1;
    }

    // Store various instances of frames of video
    cap.set(CV_CAP_PROP_POS_MSEC, t1);
    cap >> src1;
    cap.set(CV_CAP_PROP_POS_MSEC, t2);
```

```
    cap >> src2;

    // Convert image to gray and blur it
    cvtColor(src1, src1_gray, CV_BGR2GRAY);
    cvtColor(src2, src2_gray, CV_BGR2GRAY);
    blur(src1_gray, src1_gray, Size(3, 3));
    blur(src2_gray, src2_gray, Size(3, 3));

    // Applying Canny for edge Detection
    Canny(src1_gray, canny1, 10, 30, 3);
    Canny(src2_gray, canny2, 10, 30, 3);

    // Find contours
    findContours(canny1, contours1, hierarchy1, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE,
Point(0, 0));
    findContours(canny2, contours2, hierarchy2, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE,
Point(0, 0));

    // Get centres of contours
    int i, j;
    int tempx, tempy;
    vector<Point> centres1_, centres2_, centres1, centres2;
    for(i = 0; i < contours1.size(); i++){
        tempx = tempy = 0;
        for(j = 0; j < contours1[i].size(); j++){
            tempx += contours1[i][j].x;
            tempy += contours1[i][j].y;
        }
        centres1_.push_back(Point(tempx/j, tempy/j));
    }
    for(i = 0; i < contours2.size(); i++){
        tempx = tempy = 0;
        for(j = 0; j < contours2[i].size(); j++){
            tempx += contours2[i][j].x;
            tempy += contours2[i][j].y;
        }
        centres2_.push_back(Point(tempx/j, tempy/j));
    }

    centres1 = corrector(centres1_);
    centres2 = corrector(centres2_);

    vector<float> a;
    vector<float> u;
    float y1, y2;
    y1 = y2 = 0;
    for(i = 0; i < centres1.size(); i++){
        y1 = centres1[i].y;
        y2 = centres2[i].y;
        u.push_back((y2*t1*t1 - y1*t2*t2)/t1/t2/(t1-t2));
        a.push_back((y1*t2 - y2*t1)/t1/t2/(t2-t1));
    }

    vector<float> position;
    float t_final = 8000;
    for(i = 0; i < a.size(); i++)
        position.push_back(u[i]*t_final + t_final*t_final*0.5*a[i]);

    for(i = 0; i < a.size(); i++)
        cout << "x = " << position[i] << endl;
```

```
    for(i = 0; i < centres1.size(); i++)
        cout << centres1[i] << endl;









    imshow("src1", canny1);
    imshow("src2", canny2);

    if(waitKey(0) == 'q')
        return 0;
}
```

Day 6 :

We were taught about arduino and how to send commands to arduino. We also made the PS 2.