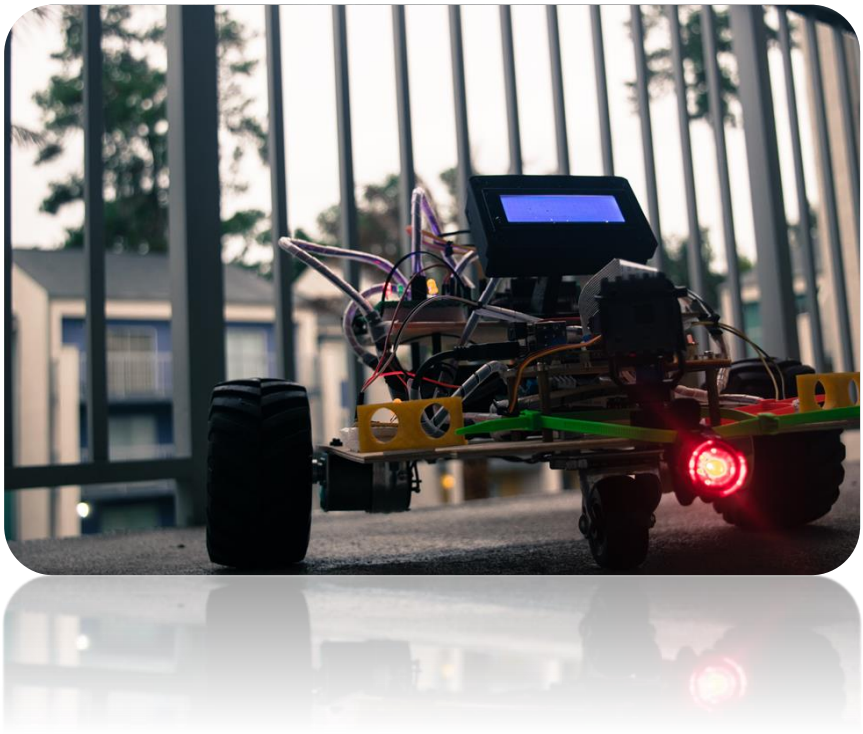
# “SI-FO”

The Universal ‘Signs’ Follower



NOVEMBER 29, 2015

PROJECT REPORT

UF IMDL | FALL 2015

# [Final Report]

# "SI-FO"

## THE UNIVERSAL 'SIGNS' FOLLOWER

Presented by:

IZHAR AMEER SHAIKH

UFID: 1191-4938

**Intelligent Machine Design Lab**

**EEL 4665, Fall 2015**

**Professors: Dr. A. Antonio Arroyo, Dr. Eric M. Schwartz**

**University of Florida**

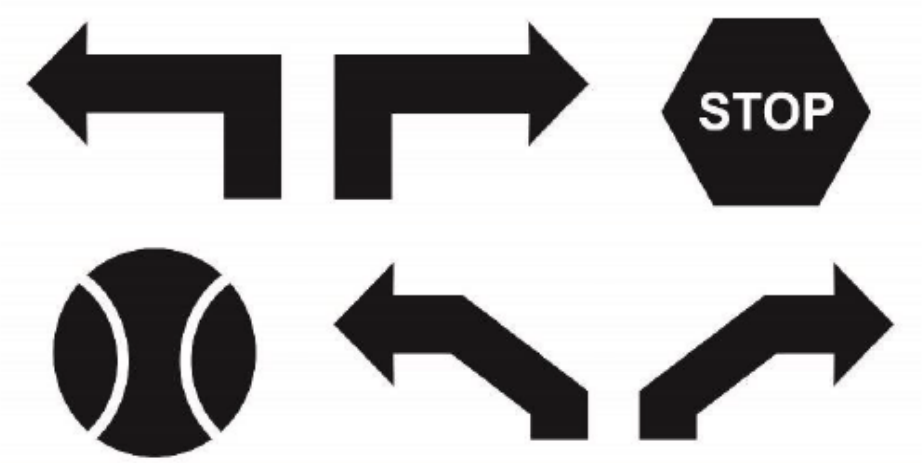
**Department of Electrical and Computer Engineering**

# TABLE OF CONTENTS

## ABSTRACT

The humans have the ability to perceive (look), identify (to know what it actually is), recognize (compare it to something similar in brain) and follow (act accordingly). They not only can do this, but can do it with the most possible efficient manner. I'm planning to implement the same kind of pattern in a robot.

The idea is to design and construct an autonomous robot, which will perceive (look in the real world through a special sensor e.g. camera), identify (know there is an object of interest), recognize (find out the 'actual' meaning of the object by Object Recognition) and follow (plan a movement according to what the object 'means'). In this case, the objects will be signs e.g. a sign which consists of an Arrow which points in the 'left direction' or 'right direction' or 'U turn' etc.




There will be a planned course which will include at least 4 signs and the last sign will be a circular ball. The signs will have fixed places in arena and once the robot begins its navigation, it will recognize the signs and plan it route accordingly. After recognizing the first sign say 'Turn Left', the robot will a 90 degree turn towards left and will move forward looking for other signs. After it finds a new sign, it will track the sign and move towards the sign until the sign is big enough for it to recognize and this process will repeat itself until robot reaches the last sign – a circular ball. After that, robot will move in a quest of finding the circular object – the ball, and once it reaches there, it will sound the buzzer and blink a led indicating the status that it has reached its destination.

The robot will make use of computer vision techniques to find the exact meaning of objects.

# INTRODUCTION

## What is Computer Vision?

According to Wikipedia, "*Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions. A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory. Computer vision has also been described as the enterprise of automating and integrating a wide range of processes and representations for vision perception.*"

## Scope and Objectives

The Intelligent Machine Design Lab class is structured such that system integration and end functionality is valued above subsystem design. With this in mind, almost none of the software or hardware solutions will be designed and implemented from the ground up. For example, the entire OpenCV library is available as an open-source software package and many reliable, tested motor controllers are available for purchase. The scope of the project is to get all of these subsystems to function cohesively together with minimal customization.

The objectives of the project are as follows:

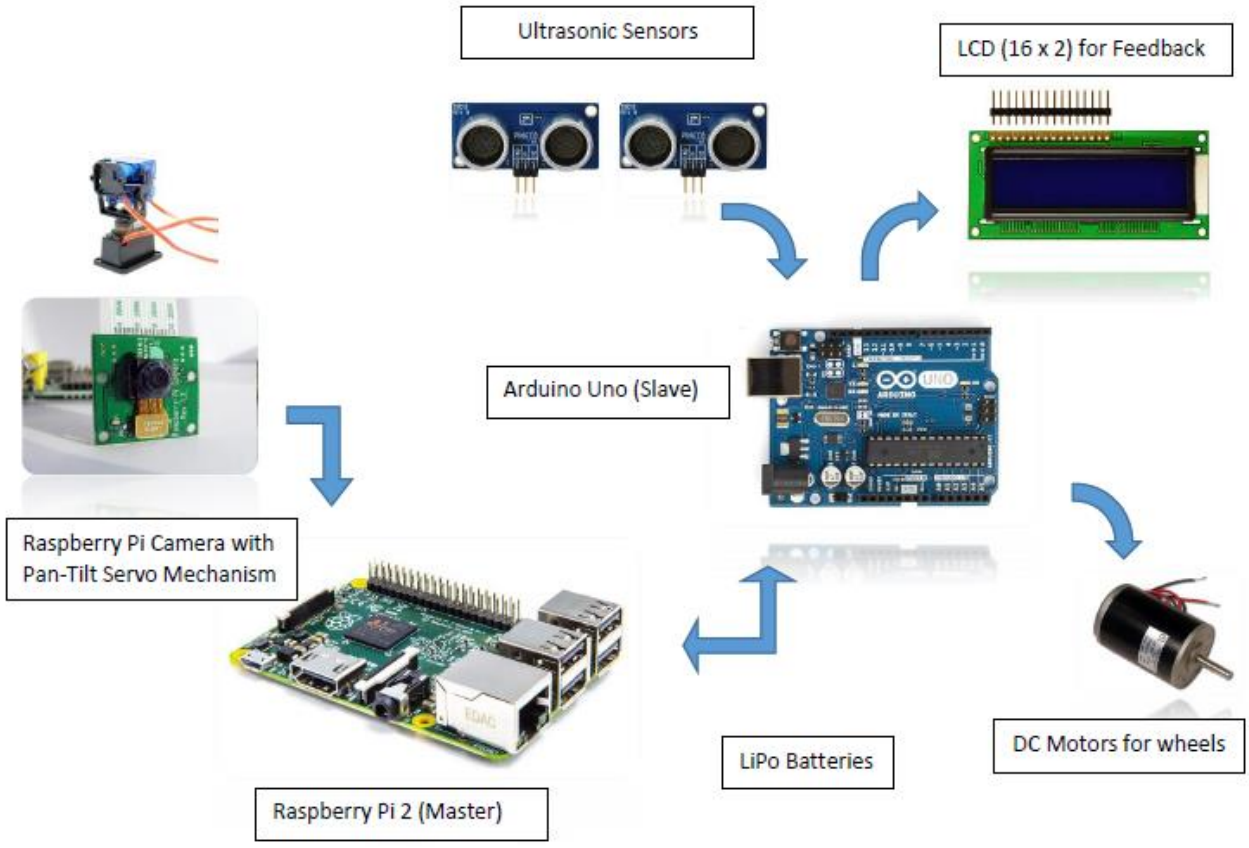
- Use OpenCV with Python to Object (i.e. Sign) Locating & Tracking.
- Tracking the object until it is close enough.
- Compare the object image to reference images and detect the sign from a list of signs.
- Choose the necessary action after recognizing a particular sign (i.e. Turn Left, Right, Backwards or Stop etc.)
- Complete the planned course in efficiently.

## WALK-THROUGH

**The report is broken down into the following nine sections:**

- Integrated System: High-level organization of robot and accessory systems.
- Mobile Platform: Physical robot frame that holds the sensors, batteries, etc.
- Actuation: Components that move or effect the pose of the platform or sensors.
- Sensors: Components that provide information about the surrounding environment.
- Behaviors: Breakdown of tasks and operations to complete while running.
- Experimental Results: System setup and data presentation.
- Conclusion: Summary and lessons learned.
- Documentation: References and bibliography.
- Appendices: Code, circuit diagrams, and supplementary.

## INTEGRATED SYSTEM




The robot will have two main processors on board.

1. The main processor will be a Raspberry Pi 2 (Model B). This board will be responsible for vision processing and behavioral algorithms.

2. The other processor will be an Arduino Uno. It will have a serial communication with Raspberry Pi and will be used to send commands to the drive motors. The Arduino will also read inputs from the ultrasonic sensors and relay that information to the Raspberry Pi.

## MOBILE PLATFORM

Since time is a very real constraint to this project (four months to complete), the goal of the mobile platform is to be as mechanically simple as possible. The lack of sophistication was chosen in order to maximize time spent on developing the software functionality instead of debugging and optimizing mechanical systems. With that in mind, the mobile platform will be defined by the following features:

- Simple wooden rectangular base approximately 25cm wide and 18cm long to house Raspberry Pi, Arduino, and Batteries etc.
- Total height approximately 30cm.
- Two driven wheels and one simple caster wheels to provide pitch stability.
- Wheels driven by two DC Gear motors without encoders.
- Brackets attached to base to provide increased support for motors.
- Sonar sensors fixed to front of wooden base.

## ACTUATION

### Cytron 12V 26RPM 83oz-in Spur Gearmotor

- Output power: 1.1 Watt
- Rated speed: 26RPM
- Rated current: 410mA
- Rated torque: 588mN.m
- Gear ratio: 120:1


Pololu 37D mm Metal Gearmotor Bracket (Pair)

- Twelve M3 screws (six for each bracket)
- Each bracket features seven mounting holes for M3 or #4-size screws (not included)
- Light-weight

## Lynxmotion Off Road Robot Tire - 4.75"D x 2.375"W (pair)

• Proline Masher 2000 (Soft)

• Diameter: 4.75"

• Width: 2.375"

• Weight: 6.40 oz.

• Required Hubs: 6mm Mounting Hub (HUB-12: Lynxmotion Hex Mounting Hub HUB-12 - 6mm (Pair)

• Required Motor: Any 6mm output shaft

## Lynxmotion Hex Mounting Hub HUB-12 - 6mm (Pair)

• High quality RC truck (12mm hex pattern) tire hub

• Works with any motor with a 6mm shaft

• For mounting Lynxmotion Off Road Robot Tire - 4.75"D x 2.375"W (pair)

• Dimensions: 22 mm long x 16 mm diameter

• Weight: 13 g

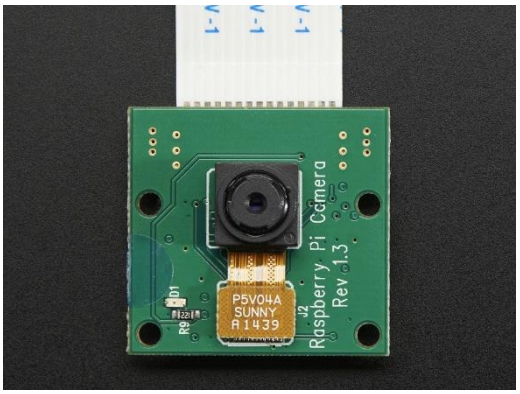
# Sensors

## Raspberry Pi Camera

Small board size: 25mm x 20mm x 9mm

A 5MP (2592×1944 pixels) Omni vision 5647 sensor in a fixed focus module

Support 1080p30, 720p60 and 640x480p60/90 video record




## Ultrasonic Sensors


- Provides precise, non-contact distance measurements within a 2 cm to 3 m range
- Ultrasonic measurements work in any lighting condition, making this a good choice to supplement infrared object detectors
- Simple pulse in/pulse out communication requires just one I/O pin
- Burst indicator LED shows measurement in progress
- 3-pin header makes it easy to connect to a development board, directly or with an extension cable, no soldering required

# Behaviors

**1. Search the sign:**

At first, the robot will start revolving about itself in search of a sign at the location where it started. The Pi camera will check if it sees any sign by looking for the blue rectangle around the sign, every time it pauses revolving for a short time. If it won't find the sign within a certain amount of rotation it will translate some finite distance and again start searching. It will start approaching the sign after detecting it.

**2. Approach towards sign:**

Once the robot has found a sign, it needs to approach it. It will do this by first centering the sign in the view. Once it center's the sign, it will drive straight towards it. Every now and then, it will check again to make sure the sign is still at the center of the view and adjust if it isn't. Also, it will constantly check distance to the sign. Once it reaches a distance where the sign is close enough, it will proceed to the next stage of color detection.

**3. Detect shape of the sign and move towards it:**

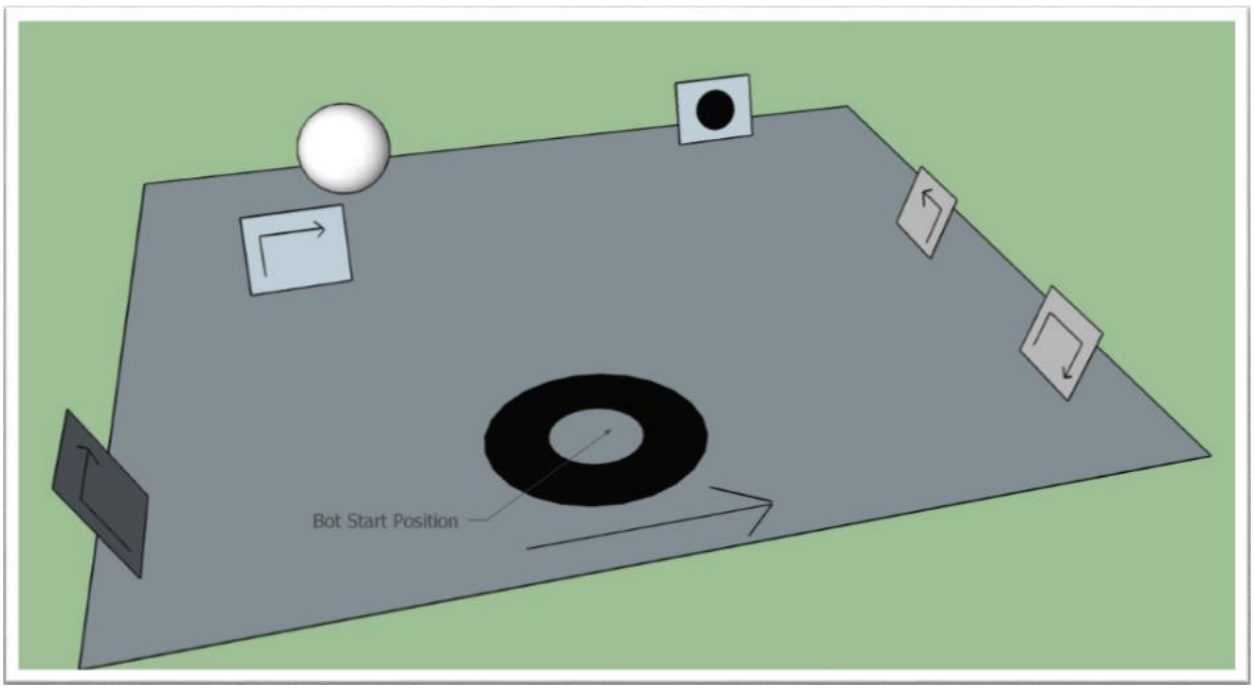
Once the robot is close enough to look at the sign, it will need to figure out the meaning of the sign. It will use image thresholding and contour detection to determine the shape. Once it detects the sign, it will make a move according to the sign.

**4. Search for the new sign and repeat behaviors 2 and 3 until it finds the last sign.**

**5. Do the last part:**

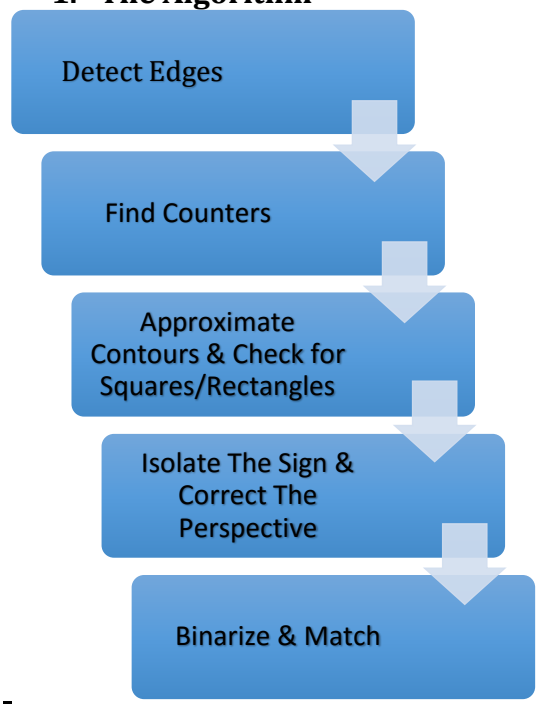
After recognizing the last sign (i.e. a sign of a ball), the bot will move in search of a spherical ball and when the ball is identified, the bot will be go to the ball and play a buzzer at the end indicating the completion of the route.

**OVERVIEW OF THE ARENA:**



## EXPERIMENTAL LAYOUT & RESEARCH (ONLY SPECIAL SENSOR)

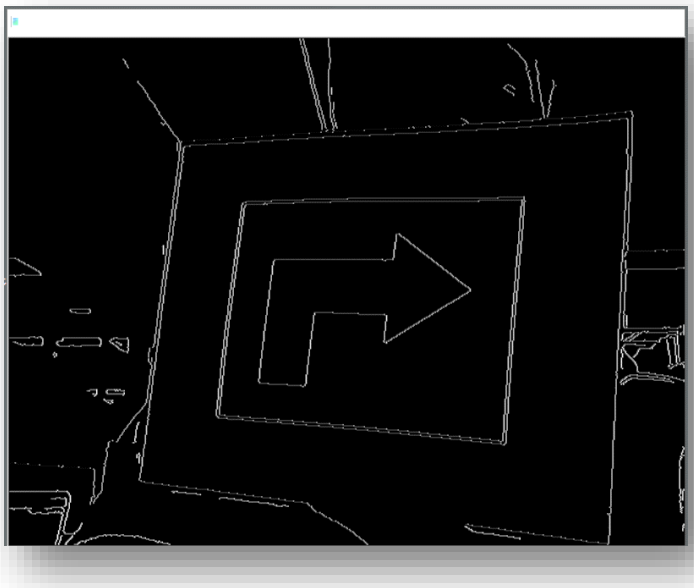
### 1. The Algorithm




When it is close to the sign it performs the next steps to read it:

**1. Find image contours**

Apply **GaussianBlur** to get a smooth image a then use **Canny** to detect edges. Now use this image to find contour using OpenCV method **findContours**. The Canny output looks like this:



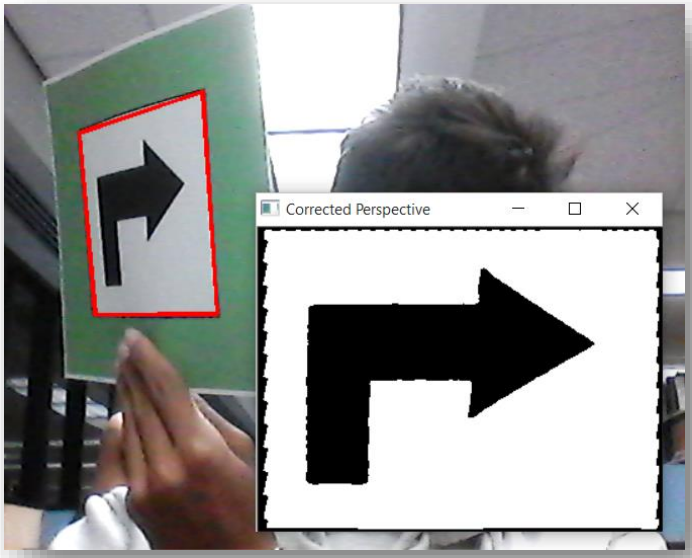
**2. Find approximate rectangular contours**

All the signs have a black rectangle around them so the next step is to find rectangular shaped contours. This is performed using **approxPolyDP** method in OpenCV. At this point found polygons are filtered by number of corners (4 corners) and minimum area.

**3. Isolate sign and correct perspective**

Because the robot moves it will not find perfectly aligned signs. A perspective correction is necessary before try to match the sign with the previous loaded reference images. This is done with the **warp Perspective** method.

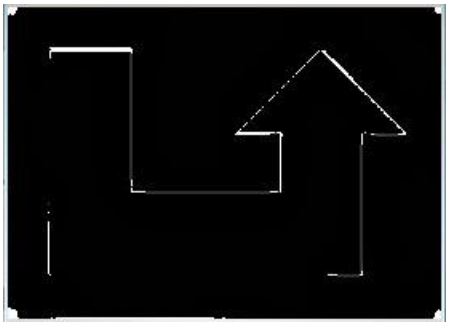
In the next image you can see the result of the process, in "B" is showed the corrected image.



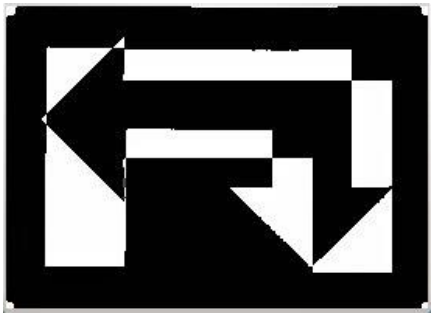

**4. Binarize and match**

After isolate and correct the interest area, the result image is binarized and a comparison is performed with all the reference images to look for a match. At the moment the system has 8 reference images to compare.

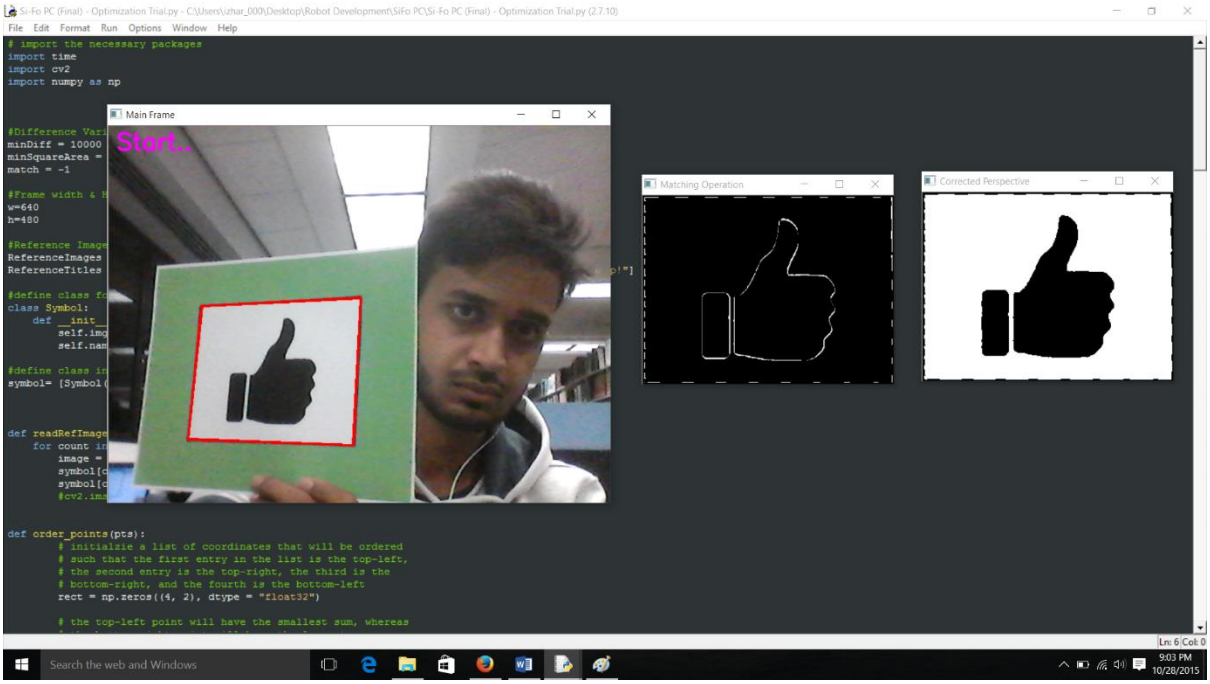
Comparison is performed using a binary XOR function (**bitwise_xor**). In the next images is showed the comparison result for match and not match, using **countNonZero** method is possible to detect the match sign.

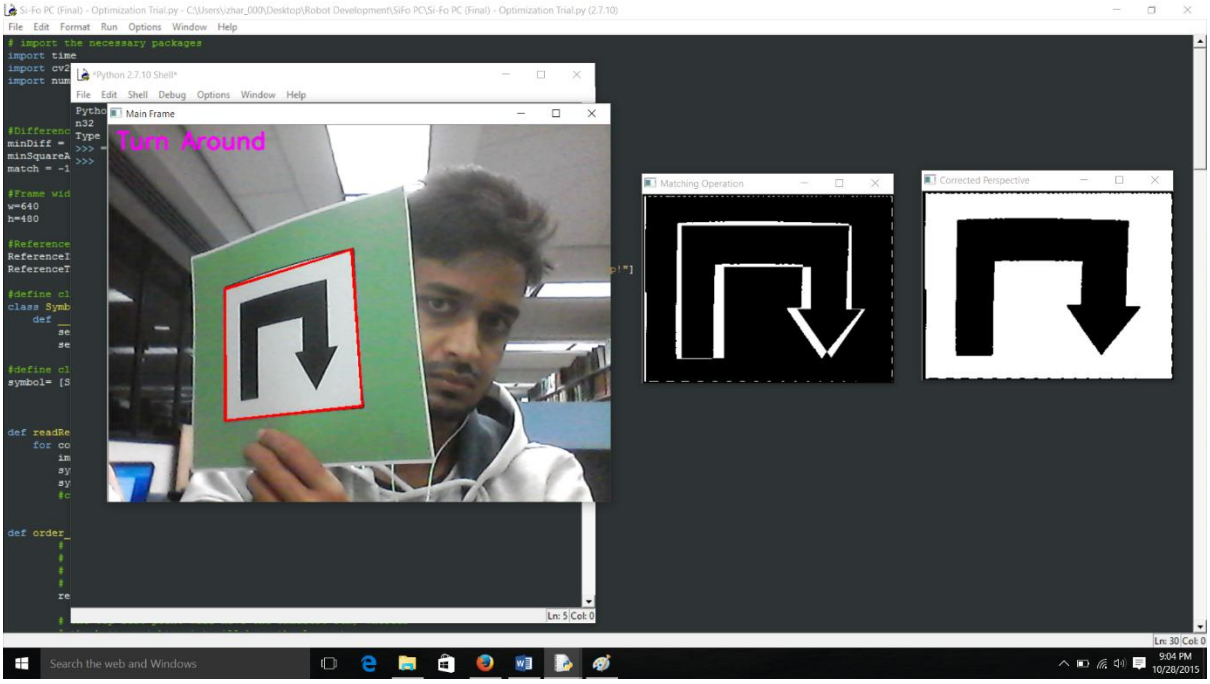


Match image

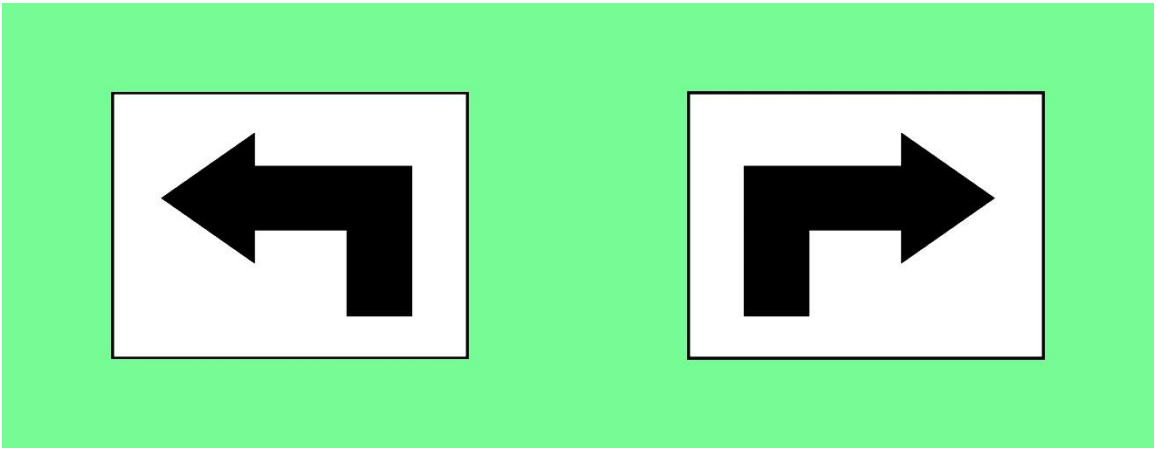


Not match image


Main Frame
Start..
Matching Operation
Corrected Perspective


Main Frame
Turn Around
Matching Operation
Corrected Perspective

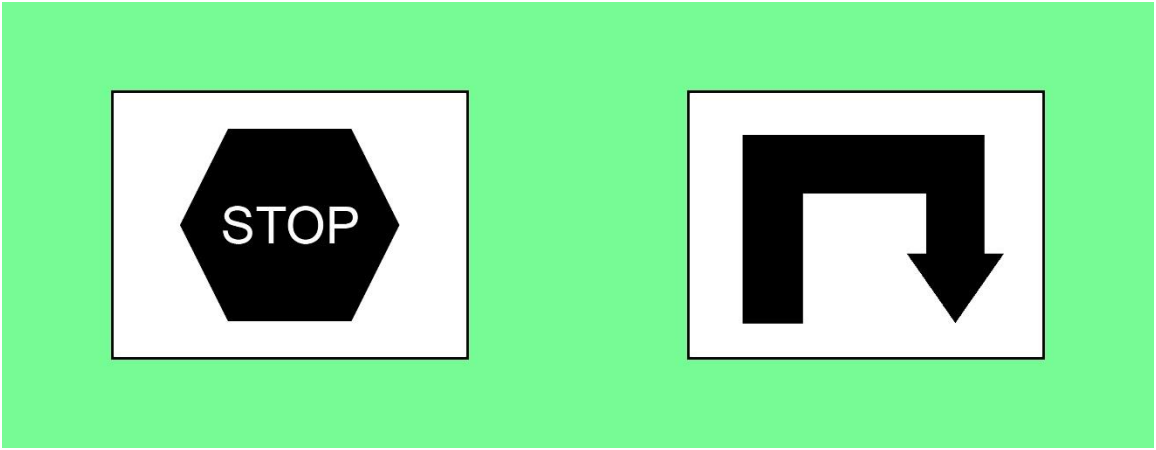
## Final Signs



Turn Right



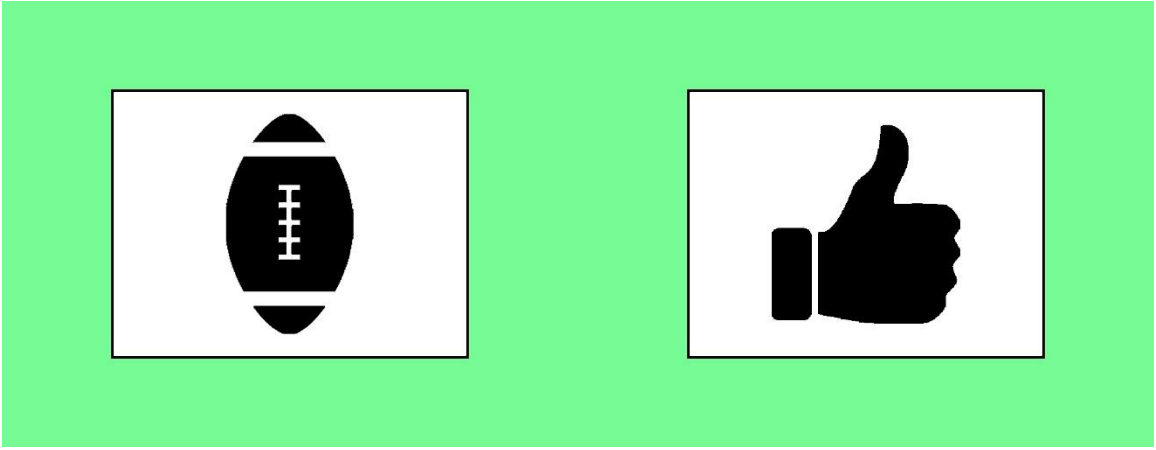
Turn Left




Stop



Turn Around



## Python Code:

```
import cv2
import urllib
import time
import numpy as np
from LoadSymbols import Symbol
from websocket import create_connection

# Width & Height From Camera
width = 640
height = 480

# Difference Variables
minDiff = 8000
minSquareArea = 2000
match = -1

# Middle X
MidX = width/2
MidY = height/2

# Font Type
font = cv2.FONT_HERSHEY_SIMPLEX

# Needed Variables for holding points for perspective correction
rect = np.zeros((4, 2), dtype = "float32")

maxW = width/2
maxH = height/2

dst = np.array([
        [0, 0],
        [maxW - 1, 0],
        [maxW - 1, maxH - 1],
        [0, maxH - 1]], dtype = "float32")

# Instance for Streaming
stream = urllib.urlopen('http://192.168.137.188:8080/?action=stream')
webSocket = create_connection("ws://192.168.137.188:8000")

# Reference Images Display name & Original Name
Reference_Symbols = ["Symbols/ArrowL.jpg",
                     "Symbols/ArrowR.jpg",
                     "Symbols/ArrowT.jpg",
                     "Symbols/Ball.jpg",
                     "Symbols/Go.jpg",
                     "Symbols/Stop.jpg"]

Symbol_Titles = ["Turn Left 90",
                 "Turn Right 90",
                 "Turn Around",
                 "Search for Ball",
                 "Start..",
                 "Stop!"]

Actions = ["Left", "Right", "Back", "Ball", "Go", "Stop"]
```

```
# Define Class Instances for Loading Reference Images (6 objects for 6
different images/symbols)
symbol= [Symbol() for i in range(6)]


def load_symbols():
    for count in range(6):
        symbol[count].read(Reference_Symbols[count], Symbol_Titles[count],
size=(width/2, height/2))
        print "Loading: ", symbol[count].name
    print "All Reference Images Are Successfully Loaded!"


def arduino_string(arduinoList=[], *args):
    return ''.join(['A1:', str(arduinoList[0]), '&2:', str(arduinoList[1])])


def servo_string(servoAngle):
    servoDC = 5./180.*(servoAngle)+5
    return ''.join(['S', str(round(servoDC, 2))])


def generate_windows():
    # Windows to display frames
    cv2.namedWindow("Main Frame", cv2.WINDOW_AUTOSIZE)
    # cv2.namedWindow("Matching Operation", cv2.WINDOW_AUTOSIZE)
    # cv2.namedWindow("Corrected Perspective", cv2.WINDOW_AUTOSIZE)


def get_canny_edge(image, sigma=0.33):
    # compute the median of the single channel pixel intensities
    v = np.median(image)

    # apply automatic Canny edge detection using the computed median
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edged = cv2.Canny(image, lower, upper)

    # return the edged image
    return edged


def correct_perspective(frame, pts):
        # Sort the contour points in Top Left, Top Right, Bottom Left &
Bottom Right Manner
        s = pts.sum(axis = 1)
        rect[0] = pts[np.argmin(s)]
        rect[2] = pts[np.argmax(s)]
        diff = np.diff(pts, axis = 1)
        rect[1] = pts[np.argmin(diff)]
        rect[3] = pts[np.argmax(diff)]

        # Compute the perspective transform matrix and then apply it
        M = cv2.getPerspectiveTransform(rect, dst)
        warped = cv2.warpPerspective(frame, M, (width/2,height/2))
        warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
```

```
        # Calculate the maximum pixel and minimum pixel value & compute
threshold
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(warped)
        threshold = (min_val + max_val)/2

        # Threshold the image
        ret, warped = cv2.threshold(warped, threshold, 255,
cv2.THRESH_BINARY)

        # Return the warped image
        return warped


def main():
    load_symbols()
    generate_windows()
    bytes = ''
    match = -1
    w_save = 0
    motor_speed = 150
    displayText = []
    turn = False
    sign_found = False

    while True:
        bytes += stream.read(2048)

        a = bytes.find('\xff\xd8')
        b = bytes.find('\xff\xd9')

        if a!=-1 and b!=-1:
            jpg = bytes[a:b+2]
            bytes = bytes[b+2:]
            camera_frame = cv2.imdecode(np.fromstring(jpg,
dtype=np.uint8),cv2.IMREAD_COLOR)

            # camera_frame = cv2.flip(camera_frame, 0)
            # camera_frame = cv2.flip(camera_frame, 1)

            # Changing color-space to grayscale & Blurring the frames to
reduce the noise
            gray = cv2.cvtColor(camera_frame, cv2.COLOR_BGR2GRAY)
            blurred = cv2.GaussianBlur(gray,(3,3),0)

            # Detecting Edges
            edges = get_canny_edge(blurred)

            # Contour Detection & checking for squares based on the square
area
            cnt_frame, contours, hierarchy =
cv2.findContours(edges,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

            # Sorting contours; Taking the largest, neglecting others
            contours = sorted(contours, key=cv2.contourArea, reverse=True)
[:1]
```

```
            for cnt in contours:
                    approx =
cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)

                    if len(approx) == 4:
                            area = cv2.contourArea(approx)

                            if area > minSquareArea:
                                warped = correct_perspective(camera_frame,
approx.reshape(4, 2))

                                for i in range(6):
                                        diffImg = cv2.bitwise_xor(warped,
symbol[i].img)
                                        diff = cv2.countNonZero(diffImg)

                                        if diff < minDiff:
                                            match = i
                                            #diff = minDiff
                                            cnt = approx.reshape(4,2)
                                            displayText = tuple(cnt[0])
                                            sign_found = True
                                            break

            if sign_found == True:

                sign_found = False
                x,y,w,h = cv2.boundingRect(cnt)
                centroid_x = x + (w/2)
                centroid_y = y + (h/2)
                # Real width * focal length = 16175.288
                camera_range = round((16175.288 / w),0)
                # Draw the contours around sign & bounding box around sign &
put the sign title
                cv2.drawContours(camera_frame, [cnt], 0, (255, 0, 0), 2)
                cv2.rectangle(camera_frame, (x, y), (x + w, y + h), (0, 255,
0), 2)
                cv2.putText(camera_frame, symbol[match].name, displayText,
font, 1, (255,0,255), 2, cv2.LINE_AA)

                if w < 420:
                    if camera_range == 0 or camera_range >= 60:
                        speed_adjust = 1.0 *(((centroid_x - 320) / 380.) *
80)
                        speed_adjust = round (speed_adjust)
                        motor_l = motor_speed + speed_adjust
                        motor_r = motor_speed - speed_adjust
                        webSocket.send(arduino_string([motor_l, motor_r]))
                        turn = False
                    else:
                        if w*h >= 70000 and turn == False:
                            if Actions[match] == "Back":
                                webSocket.send(arduino_string([230,230]))
#Back
                                match = -1
                                print "Turn Back"
                                turn = True
```

```
                                elif Actions[match] == "Left":
                                    webSocket.send(arduino_string([235,235]))
#Left
                                    match = -1
                                    print "Left Turn"
                                    turn = True
                                elif Actions[match] == "Right":
                                    webSocket.send(arduino_string([232,232]))
#Right
                                    match = -1
                                    print "Right Turn"
                                    turn = True
                                elif Actions[match] == "Stop":
                                    webSocket.send(arduino_string([237,237]))
#Stop
                                    match = -1
                                    print "Stop"
                                    turn = True
                                    exit(0)
            else:
                centroid_x = 0
                centroid_y = 0
                camera_range = 0
                motor_r = 0
                motor_l = 0
                webSocket.send(arduino_string([motor_l, motor_r]))

            # Displaying co-ordinates & camera range
            central = 'Centre:  X:%d : Y:%d ' % (centroid_x, centroid_y)
            distance = 'Range:   %d cm' % (camera_range)
            cv2.putText(camera_frame, central,(20, 25), font, 0.65,
(255,255,255), 1, cv2.LINE_AA)
            cv2.putText(camera_frame, distance,(20, 45), font, 0.65,
(255,255,255), 1 , cv2.LINE_AA)

            # Displaying Frames
            cv2.imshow('Main Frame', camera_frame)

            if cv2.waitKey(1) == 27:
                break

    webSocket.close()
    stream.close()
    cv2.destroyAllWindows()


if __name__ == "__main__":
    main()
```

```
#include <Wire.h>
#include <Thread.h>
#include <ThreadController.h>
#include <LiquidCrystal_I2C.h>
#include "Support.h"

const int DIR1 = 4; //left Motor
const int EN1 = 5;
const int EN2 = 6;  //right Motor
const int DIR2 = 7;
const int pace = 140; //Speed

//Declaring Instances of Classes
//UltrasonicSensor  Center(9, 8), Left(10, 11), Right(3, 2);
Motor             L(DIR1, EN1), R(DIR2, EN2);
//LiquidCrystal_I2C LCD(0x27, 20, 4);
Thread            *blinkAllLEDs = new Thread();
ThreadController  *threadController = new ThreadController();

// Calculate based on max input size expected for one command
#define INPUT_SIZE 15
char input[INPUT_SIZE + 1];

//Variables for Ultrasonic Sensor
boolean left=false;
boolean center=false;
boolean right=false;

boolean turn = false;

//LED Pins
uint8_t ledArray[4] = {28, 29, 30, 31};
//uint8_t ledArray[4] = {13,13,13,13};

//Data to be displayed on LCD
const char *LCDData[] = {"         GO         ",
                         "      LEFT TURN     ",
                         "     RIGHT TURN     ",
                         "      TURN BACK     ",
                         "        BALL        ",
                         "        STOP        "};

void setup()
{
  Serial.begin(9600);
  Serial.setTimeout(20);

//  LCD.init();
//  LCD.backlight();
//
//  LCD.setCursor(0,0);
//  LCD.print("     #- SiFo -#     ");
//  LCD.setCursor(0,1);
//  LCD.print("  Last Sign Found   ");

  for (uint8_t pin = 0; pin < 4; pin++)
  {
```

```
    pinMode(ledArray[pin], OUTPUT);
    digitalWrite(ledArray[pin], HIGH);
  }

  ledAllOff();

//  Left.begin();
//  Center.begin();
//  Right.begin();

  L.begin(pace);
  R.begin(pace);

  blinkAllLEDs->onRun(blinkAllCallback);
  blinkAllLEDs->setInterval(5000);

  threadController->add(blinkAllLEDs);

//  for(int j=0; j<5; j++)
//  {
//    left = Left.getStatus();
//    center = Center.getStatus();
//    right = Right.getStatus();
//  }
}

void loop()
{
  //threadController->run();

  byte packetSize = Serial.readBytes(input, INPUT_SIZE);

  // Add the final 0 to end the C string
  input[packetSize] = 0;

  // Read each command pair
  char* command = strtok(input, "&");

  while (command != 0)
  {
    // Split the command in two values
    char* separator = strchr(command, ':');
    if (separator != 0)
    {
        // Actually split the string in 2: replace ':' with 0
        *separator = 0;
        int ID = atoi(command);
        ++separator;
        int value = atoi(separator);

        //if (value < -pace)
          //value = -pace;

        //if (value > pace)
          //value = pace;

        if (value == 235)    //Left
```

```
        {
          L.backward(120);
          R.forward(120);
          delay(1000);
          L.stop();
          R.stop();
          turn = true;
        }
        else if (value == 230)   //Turn Back
        {
          L.backward(130);
          R.backward(130);
          delay(1000);
          L.stop();
          R.stop();
          L.forward(150);
          R.backward(150);
          delay(1500);
          L.stop();
          R.stop();
          turn = true;
        }
        else
        {
          turn = false;
        }



    if (turn == false)
    {

        // Do something with servoId and position
        switch(ID)
        {
          case 1:
            (value >= 0)? ((value > 0)? L.forward(value) : L.stop()) :
L.backward(abs(value));
            break;

          case 2:
            (value >= 0)? ((value > 0)? R.forward(value) : R.stop()) :
R.backward(abs(value));
            break;
        }

     }
    }
    // Find the next command in input string
    command = strtok(0, "&");
  }
}


//################### LCD Update ###################################

//void updateLCD(int code)
```

```
//{
// switch(code)
// {
//   case 1:
//     L.backward(pace);
//     R.forward(pace);
//     delay(3000);
//     L.stop();
//     R.stop();
//     break;
// }
////  LCD.setCursor(0,2);
////  LCD.print(LCDData[code]);
//}


//#################### Thread Callbacks ############################

void blinkAllCallback()
{
  static bool ledStatus = false;
  ledStatus = !ledStatus;

  //for (int pinNumber = 0; pinNumber < 4; pinNumber++)
    digitalWrite(ledArray[3], ledStatus);
}


//#####################LED Routines###################################

void ledAllOn()
{
  for (int pinNumber = 0; pinNumber < 4; pinNumber++)
    digitalWrite(ledArray[pinNumber], HIGH);
}

void ledAllOff()
{
  for (int pinNumber = 0; pinNumber < 4; pinNumber++)
    digitalWrite(ledArray[pinNumber], LOW);
}


void ledRoll(int times)
{
  for (int count = 1; count <= times * 2; count++)
  {
    if (count % 2 == 0)
    {
      for (int pinNumber = 0; pinNumber < 8; pinNumber++)
      {
        if (pinNumber < 4)
          digitalWrite(ledArray[pinNumber], HIGH);
        else
          digitalWrite(ledArray[pinNumber - 4], LOW);
        delay(50);
      }
```

```
    }
    else
    {
      for (int pinNumber = 7; pinNumber >= 0; pinNumber--)
      {
        if (pinNumber > 3)
          digitalWrite(ledArray[pinNumber - 4], HIGH);
        else
          digitalWrite(ledArray[pinNumber], LOW);
        delay(50);
      }
    }
  }
}


void blinkAll(int times)
{
  //Blink
  for (int count = 1; count <= times; count++)
  {
    for (int pinNumber = 0; pinNumber < 4; pinNumber++)
      digitalWrite(ledArray[pinNumber], HIGH);
    delay(50);
    for (int pinNumber = 0; pinNumber < 4; pinNumber++)
      digitalWrite(ledArray[pinNumber], LOW);
    delay(50);
  }
}

void ledAlternate(int times)
{
  for (int count = 1; count <= times; count++)
  {
    for (int pinNumber = 0; pinNumber < 4; pinNumber++)
    {
      if (pinNumber % 2 == 0)
        digitalWrite(ledArray[pinNumber], HIGH);
      else
        digitalWrite(ledArray[pinNumber], LOW);
    }
    delay(250);
    for (int pinNumber = 0; pinNumber < 4; pinNumber++)
    {
      if (pinNumber % 2 != 0)
        digitalWrite(ledArray[pinNumber], HIGH);
      else
        digitalWrite(ledArray[pinNumber], LOW);
    }
    delay(250);
  }
}


void ledOC(int times)
{
  for (int count = 1; count <= times; count++)
```

```
  {
    for (int pinNumber = 0; pinNumber < 4; pinNumber++)
    {
      digitalWrite(ledArray[pinNumber], HIGH);
      digitalWrite(ledArray[3 - pinNumber], HIGH);
      delay(100);
      digitalWrite(ledArray[pinNumber], LOW);
      digitalWrite(ledArray[3 - pinNumber], LOW);
      delay(100);
    }
  }
}


void blinkEach(int times)
{
  for (int pinNumber = 0; pinNumber < 4; pinNumber++)
  {
    for (int count = 0; count <= times; count++)
    {
      digitalWrite(ledArray[pinNumber], HIGH);
      delay(75);
      digitalWrite(ledArray[pinNumber], LOW);
      delay(75);
    }
  }
}



#include <Arduino.h>

#define MAX_DISTANCE 20

class UltrasonicSensor
{
  private:
    uint8_t trigPin;
    uint8_t echoPin;

  public:

    UltrasonicSensor(uint8_t Tr, uint8_t Ec);
    void begin();
    int getDistance();
    boolean getStatus();
};


class Motor
{
  private:
    uint8_t DIR;
    uint8_t EN;
    uint8_t PWM;

  public:
```

```
    Motor(uint8_t Dir, uint8_t Enb);
    void begin(uint8_t pace);
    void forward(uint8_t pace);
    void backward(uint8_t pace);
    void stop();
};


#include "Support.h"

//#####################Ultrasonic Sensors#########################

    UltrasonicSensor :: UltrasonicSensor(uint8_t Tr, uint8_t Ec)
    {
      trigPin = Tr;
      echoPin = Ec;
    }

    void UltrasonicSensor :: begin()
    {
      pinMode(trigPin, OUTPUT);
      pinMode(echoPin, INPUT);
    }


    int UltrasonicSensor :: getDistance()
    {
      long duration;

      digitalWrite(trigPin, LOW);
      delayMicroseconds(2);
      digitalWrite(trigPin, HIGH);
      delayMicroseconds(5);
      digitalWrite(trigPin, LOW);

      duration = pulseIn(echoPin, HIGH, 30000);

      return int(duration/29/2);        //Convert to cm
    }


    boolean UltrasonicSensor :: getStatus()
    {
      int d = getDistance();
      return ((d > 0) && (d < MAX_DISTANCE))? true : false;
    }


//#######################Motors#################################

Motor :: Motor(uint8_t Dr, uint8_t Enb)
{
  DIR = Dr;
  EN = Enb;
};

void Motor :: begin(uint8_t pace)
```

```
{
  PWM = pace;
  pinMode(DIR, OUTPUT);
  pinMode(EN, OUTPUT);
  digitalWrite(EN, LOW);
};

void Motor :: forward(uint8_t pace)
{
  analogWrite(EN, pace);
  digitalWrite(DIR, HIGH);
};

void Motor :: backward(uint8_t pace)
{
  analogWrite(EN, pace);
  digitalWrite(DIR, LOW);
};

void Motor :: stop()
{
  digitalWrite(EN, LOW);
};


//##########################Robot Motion Control####################
/*
void backMotion()
{
  allstop();
  Lbackward();
  Rbackward();
  delay(500);
  allstop();
}

void backFromRight()
{
  allstop();
  Lstop();
  Rbackward();
  delay(2000);
  allstop();
}

void backFromLeft()
{
  allstop();
  Rstop();
  Lbackward();
  delay(2000);
  allstop();
}

void sharpLeft()
{
  allstop();
```

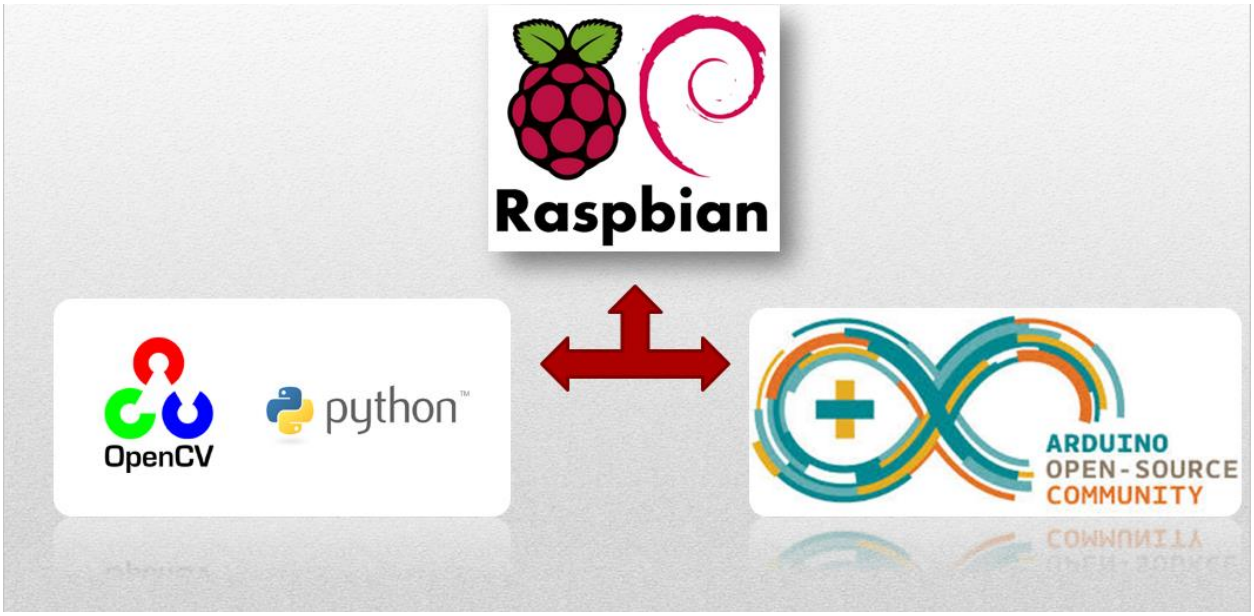
```
  Lforward();
  Rbackward();
  delay(1350);
  allstop();
}

void sharpRight()
{
  allstop();
  Lbackward();
  Rforward();
  delay(1350);
  allstop();
}

void tiltRight()
{
  allstop();
  Rbackward();
  Lforward();
  delay(350);
  allstop();
}

void tiltLeft()
{
  allstop();
  Lbackward();
  Rforward();
  delay(350);
  allstop();
}
*/
```

**SOFTWARE:**



**CONCLUSION:**

Si-Fo was not only a challenging task was for me, but it was a great experience. I have been introduced to so many new things like real-time image processing, WebSockets, network communications and protocols, communication between integrated circuits and real-time constraints. The project is moderately complex and deals with many different fields of engineering.

There is still a scope of improvement and upgradation. Overall, it was a great learning experience to implement this and I'm happy about it.