1. Approach

To make a machine detect signs from a real time video feed, we can either use MATLAB or OpenCV. Since OpenCV with Python is widely used for IoT based applications, we will also be using it as that will enable us to implement other ideas easily. OpenCV provides flexibility to implement it in various day to day IoT appliances. To handle the image processing, complex math operations are required. To cate this need, we use Numpy library and its functions.

Since the objective of the project is to deliver a sign detecting solution, we will only need a system with OpenCV, Python and Numpy installed. Also as there's is not much need of machine learning algorithms for a basic project.

To detect signs, a camera has to process the video frame by frame i.e. we implement still frame processing at a fast rate. The processing involves the following

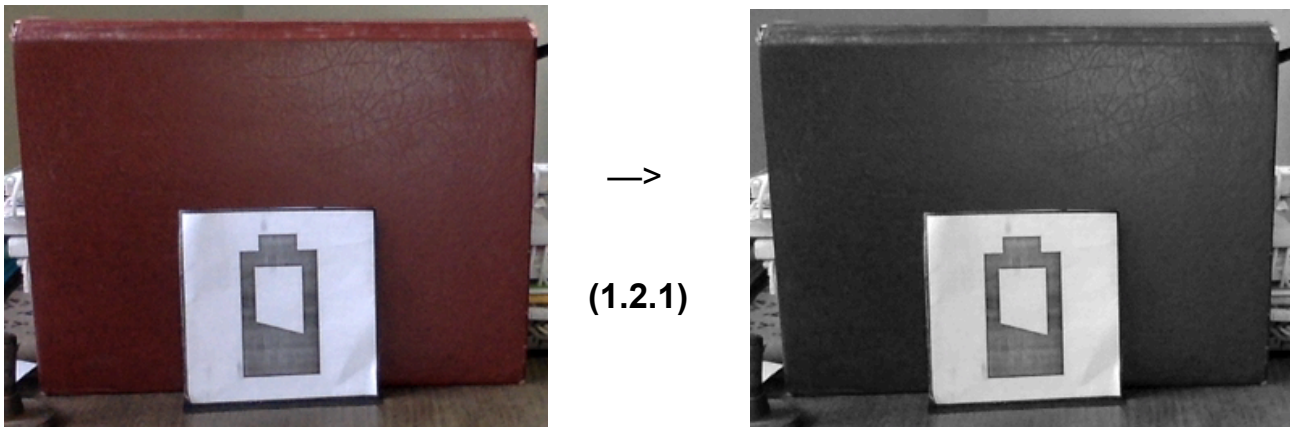
1. Obtaining feed from camera
2. Applying Gray Scale Filter the frame
3. Applying Gaussian Blur to the grey scale image
4. Using canny edge detection on blurred image to find contours in the frame

2. Image Processing:

Gray Scale Transform:

The frame obtained from camera is first transformed from BGR to Gray Scale. Transforming the colours will reduce the amount of data to be processed later in canny edge detection. This gray scale image will then be applied gaussian blur.

<u>The effect of Gray Scale Filter:</u>



—>

(1.2.1)

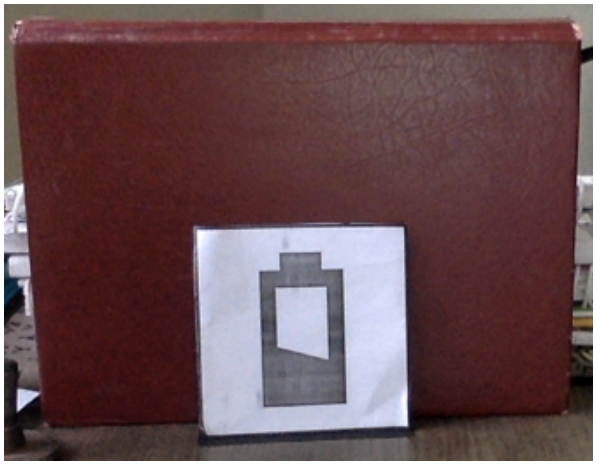
Gaussian Blur:

Gaussian Blur function in OpenCV library is used to blur the image/frame passed to function as argument. Gaussian Blur function is used to reduce the noise in the original frame before passing it to the auto_canny function which is user defined. If we don't apply any filter to the frame, unwanted edges will be detected which will hinder the efficiency of the code and robot.

We can observe results of the function in the following pictures:

The effect of Gaussian blur:

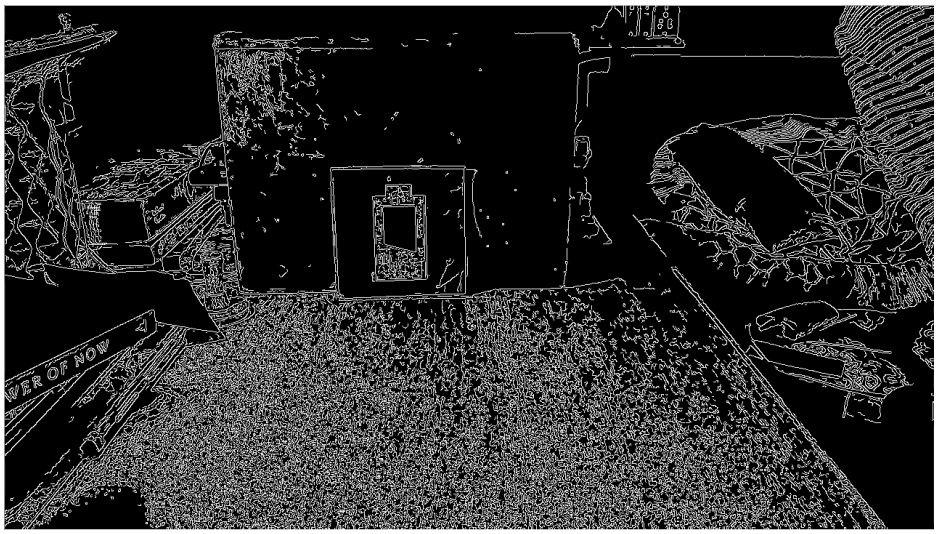
Before **(1.2.2)** After



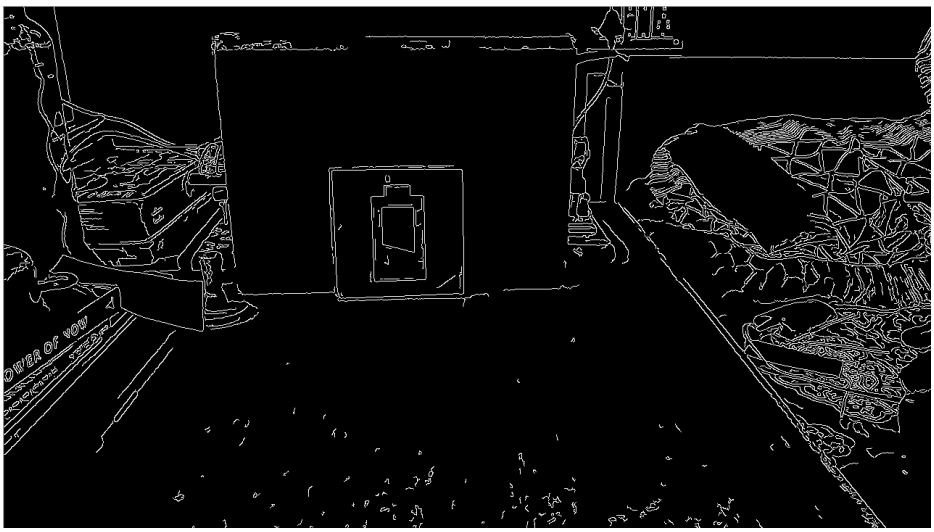
The effect of Gaussian blur on Canny Edge Detection:

Without Blur:

(1.2.3)



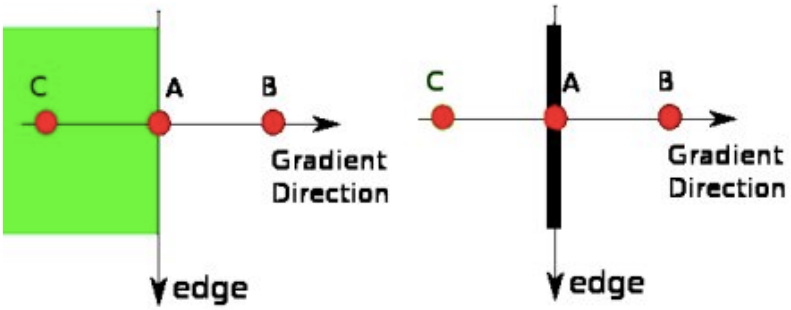
With Blur :



(1.2.4)

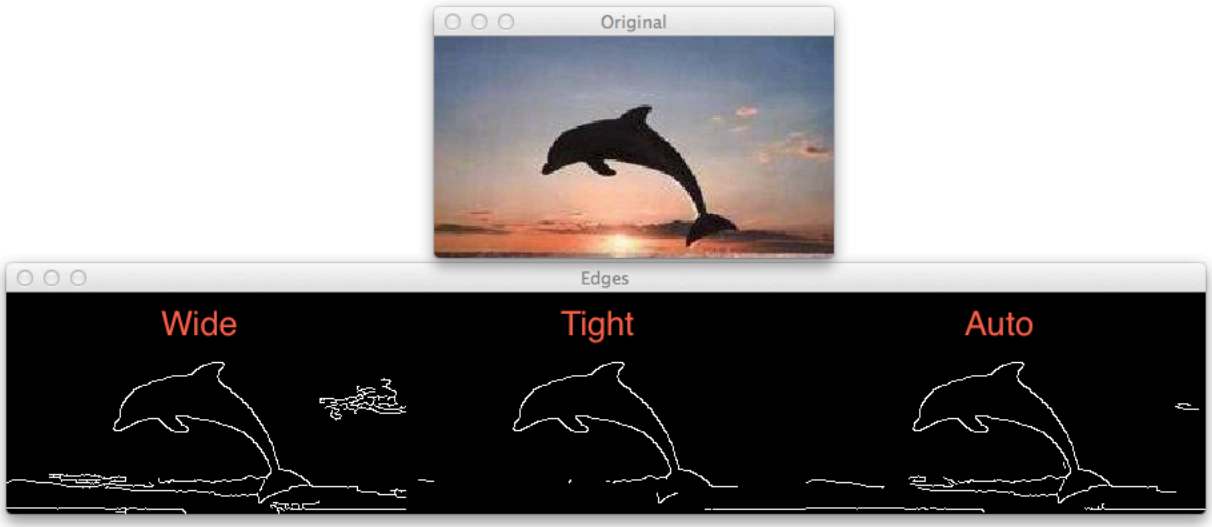
Auto Canny Edge Function:

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. The canny edge detector function detects edges on the basis of changing gradient in the images.



(3.2.5)

In auto canny edge detection, we use the median of the image array and perform a set of operations to produce the lower and upper bounds for the canny edge function.

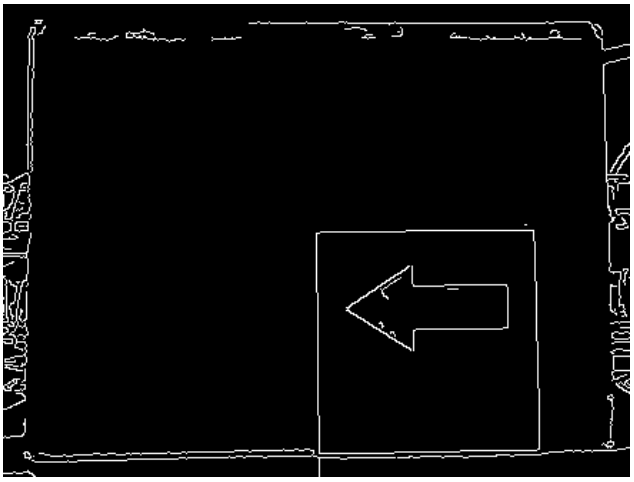


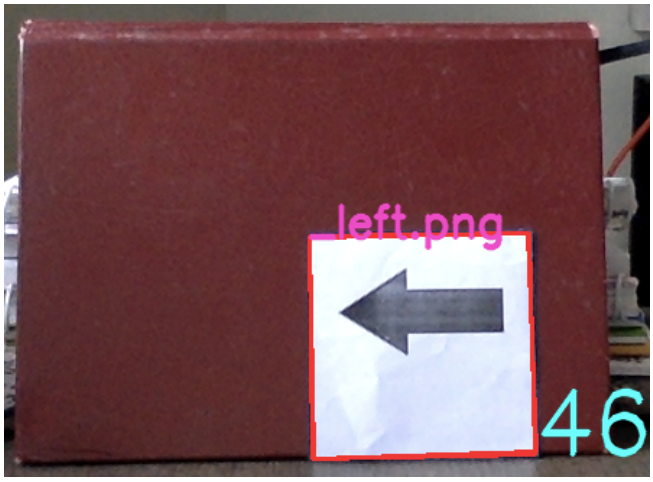
(1.2.6)

A function auto_canny is defined that takes an image as an argument and processes it using the above discussed algorithm and returns an image which has a black background and detected edges.

Run time Example:

Edges Detected **(1.2.7)** Original Frame

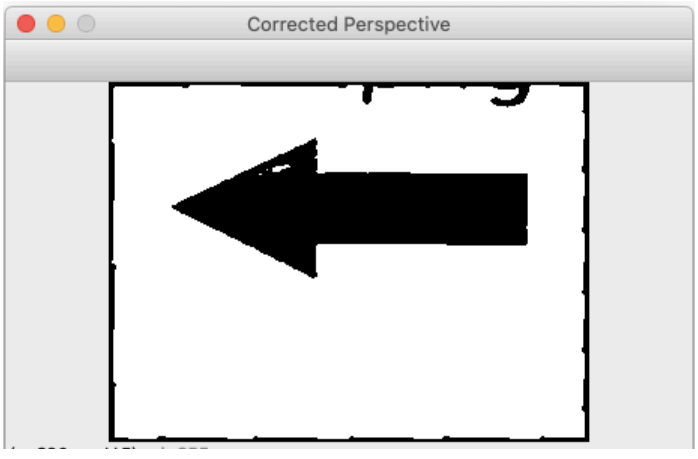




Perspective Transform:

The matching operation would face difficulties if the orientation of frame is not perpendicular. To fix this problem, we use perspective transformation on the detected polygon contour. This is achieved by using four point perspective transform algorithm.

Run Time Results



(1.2.8)

Operations Post Image Processing:

After we detect the sign in the frame, the co-ordinates of the polygon contour of sign are used to find the position of the sign with respect to the camera. To give the position, we define two parameters: 1. Direction and 2. Distance. The distance and direction calculations are accomplished using the numpy array of the polygon contour.

The name of the sign is displayed after the calculation of position on the window that shows real time output of the program. The direction and distance is also printed on the output window.

Code And Algorithm:

The first step to solve any task is to devise the algorithm or logic of the solution.

Decoding Logic:

Any camera capturing a video feed is basically capturing many still image frames finite times every second. Hence, video processing is simply still mage processing.

The logic can be explained in the following manner. We capture a snapshot and then apply certain filters to it such that it becomes easy to detect edges. Later we find contours and check for rectangular contours. If a rectangular contour is detected, the contour is cropped and its perspective is normalised. To find a match with the existing sign library, we perform bitwise xor on the warped image and stored signs in the directory. The sign with which the bitwise operator obtains a value below certain threshold limit is said to be a match.

Thus we can detect signs from a real time video feed using still frame image processing.

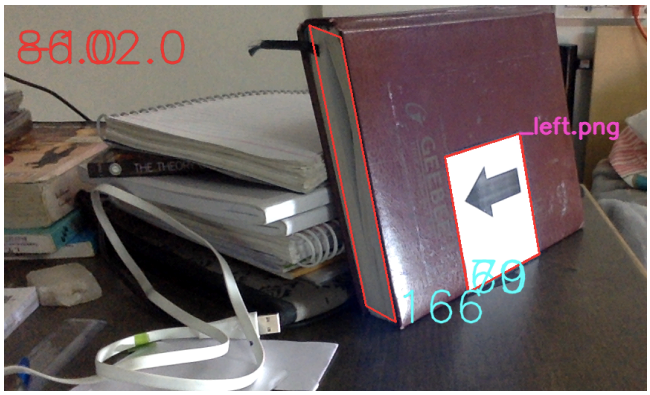
Detailed Algorithm:

```
Step 1: START
Step 2: image = image read by camera
Step 3: Gray = GRAY_SCALE_IMAGE(image)
Step 4: Blur = Gaussian_Blur(Gray)
Step 5: Edge_frame = Canny_Edge(Blur)
Step 6: Contours = Find_Contour(Edge_Frame)
Step 7: FOR cnt in Contours
        7.1     Approx = Apprx_Polygon(cnt)
        7.2     IF length(Approx) == 4

                7.2.1     Area = Contour_Area(Approx)
                7.2.2     IF Area(Contour) > 5000

                          7.2.2.1     Warped=Perspective_Transform(image)
                          7.2.2.2     FOR i in range(6)
                                      sym=symbol[i].img
                                      Diff=Bitwise_XOR(sym,image)
                                      W=countNonZero(Diff)
                                      IF W<Mindiff
                                          Match found
                                          DISPLAY matching sign name
      7.3   IF Exit_protocol==True
      7.4   GOTO Step 8
Step 8: End
```

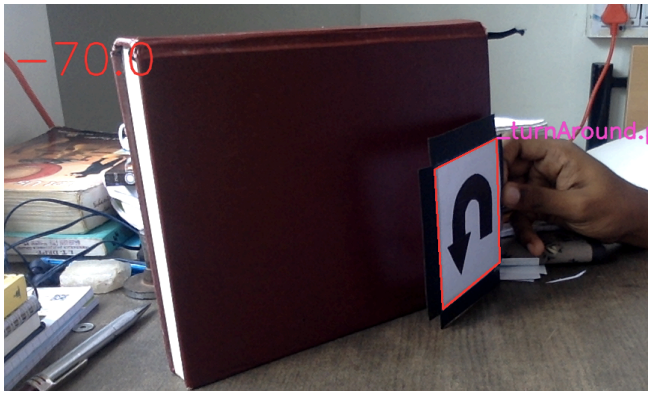
2. Results and Conclusion:

1. Detection Test:

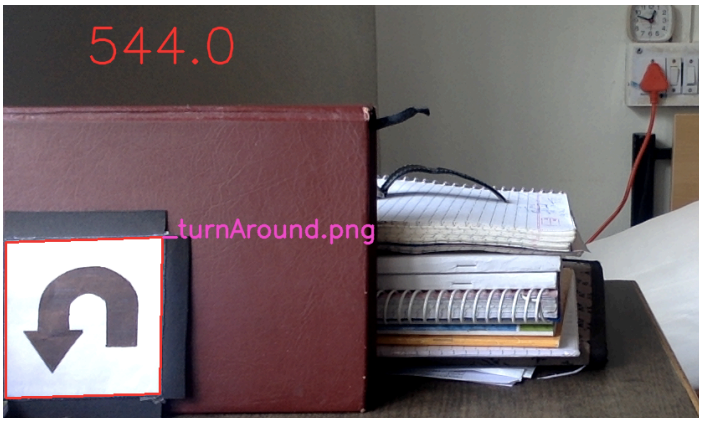
(2.1.1)



2. Perspective Test:**(2.2.1)**



3. Max Direction Test:



(2.3.1)

Thus, signs inclined almost less than 45 degrees with respect to horizontal and also having a steep perspective can be detected.