



# Final Report

---

**Implementing Efficient Traffic Sign Recognition for Turtlebot3**

---

---

## Declaration of independence

Remerghani

---

---

## Table of Contents

<b>LIST OF ABBREVIATIONS.....</b>	<b>V</b>
<b>LIST OF FIGURES .....</b>	<b>VI</b>
<b>LIST OF TABLES.....</b>	<b>VII</b>
<b>MOTIVATION AND GOAL .....</b>	<b>1</b>
1.1 Introduction .....	1
1.2 Objective.....	1
<b>THEORY &amp; ALGORITHMS .....</b>	<b>2</b>
2.1 Key Task .....	2
2.2 Algorithm Analysis .....	3
2.2.1 YOLO	
2.2.2 CNN	
2.2.3 SIFT	
2.2.4 SURF	
2.2.5 ORB	
2.3 Algorithm to be Implemented: .....	5
<b>IMPROVEMENT .....</b>	<b>6</b>
3.1 Area of Improvement .....	6
3.2 Pathway to Improvement .....	6
3.2.1 Algorithm Change	
3.2.2 Technique Change	
3.3 Implementation of Improvement .....	8
3.3.1 Implementing YOLO	
3.3.2 Implementing ORB	
3.3.3 Adding Frame of Detection	
3.3.4 Combining SIFT and YOLO	
3.3.5 Final Approach	
<b>RESULTS.....</b>	<b>17</b>
4.1 Test-Cases .....	17
4.2 Testing.....	18
4.2.1 On Table Results	
4.2.2 On-Track Results	
4.2.3 Computational Test	
4.3 Results .....	22

<b>SUMMARY .....</b>	<b>25</b>
<b>5.1 Conclusion .....</b>	<b>25</b>
<b>5.2 Discussion .....</b>	<b>26</b>
5.2.1 Problems faced	
5.2.2 Future Possibilities and Thoughts	
<b>REFERENCES .....</b>	<b>28</b>
<b>6.1 References.....</b>	<b>28</b>

## LIST OF ABBREVIATIONS

<b>Abbreviations</b>	<b>Full Forms</b>
CNN	Convolution Neural Networks
SIFT	Scale-Invariant Feature Transform
ORB	Oriented FAST and rotated BRIEF
SURF	Speeded Up Robust Features
YOLO	You Look Only Once
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
ARM	Advanced RISC Machine
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
HSA	Hochschule Anhalt
FLANN	Fast Library for Approximate Nearest Neighbours
DNN	Deep Neural Networks

## LIST OF FIGURES

<b>Figure Number</b>	<b>Names</b>	<b>Page No.</b>
Figure 1	Process of Traffic Sign Recognition .....	2
Figure 2	Traffic Signs.....	2
Figure 3	Devil's Triangle .....	3
Figure 4	Steps to Decide Algorithm .....	6
Figure 5	Steps of Adding Frame of Detection.....	7
Figure 6	Steps of having Two Algorithm.....	7
Figure 7	Annotation Images .....	8
Figure 8	Annotation.....	8
Figure 9	images directory .....	9
Figure 10	labels Directory .....	9
Figure 11	Directories to be Added .....	9
Figure 12	Directory Created Automatically .....	10
Figure 13	Reference Image Directory .....	12
Figure 14	Frame of Detection .....	14
Figure 15	Flowchart of System .....	16
Figure 16	On-Table Set-up.....	18
Figure 17	Reference Images.....	19
Figure 18	Results of SIFT .....	20
Figure 19	On-Track Set-up .....	20
Figure 20	Results .....	24

## LIST OF TABLES

<b>Table Number</b>	<b>Names</b>	<b>Page No.</b>
Table 1	Test-Cases.....	17
Table 2	On Table Test YOLO.....	18
Table 3	On Table Test ORB.....	19
Table 4	On Table Test SIFT.....	19
Table 5	On Table Test YOLO + SIFT.....	20
Table 6	On Track Test YOLO+SIFT .....	21
Table 7	Measurements of Computational Time.....	22
Table 8	Computational Time Test.....	22
Table 9	Accuracy of all Algorithms .....	22
Table 10	Trials of YOLO Model.....	26

## CHAPTER 1

# MOTIVATION AND GOAL

### 1.1 Introduction

Turtlebot3 is having inbuilt SIFT Algorithm for Traffic Sign Recognition [1], however Turtlebot3 is a customizable robot, so we can change its Traffic Sign Recognition as per our convenience which might also improve its accuracy. But question is which technique to use instead of SIFT and is it necessary to change it? So, after doing some deep research and analysis, we can have knowledge of which Algorithm is best among all other techniques. Hence, we can develop our own improvised version of package and this new package should have better accuracy as compared to previous version and it should work a lot better than before. Finally, after developing our own packages we will intercompare both packages and find the best suitable package. Best part of this Project is that we will explore a wide scope of techniques and will come to one final solution. However, only fallback is that we must work on ROS so we need to work with older version of LINUX which might restrain the different algorithms from giving their best performance.

### 1.2 Objective

The primary aim of this is Project enhance the accuracy of the best suitable package by modifying it into a new package. Modifications possible are:

1. Best Recognition Algorithm
2. Best Technique with most reliable results

Further, we will compare the modified package with the expert's package and will try to find the best package among both packages. The criteria for judging would be:

1. Accuracy (Individual and Overall) By: TP, TN, FP, FN
2. Working in all environment - with light and without light
3. By position of signs – Left, Right, Up, Down, Front and Back
4. Detection at different speed of turtlebot
5. Calculation time (Computational performance and Complexity) – How many cores does it use and processing time

Finally, the best package will be selected to the HSA Package for further analysis by different students.

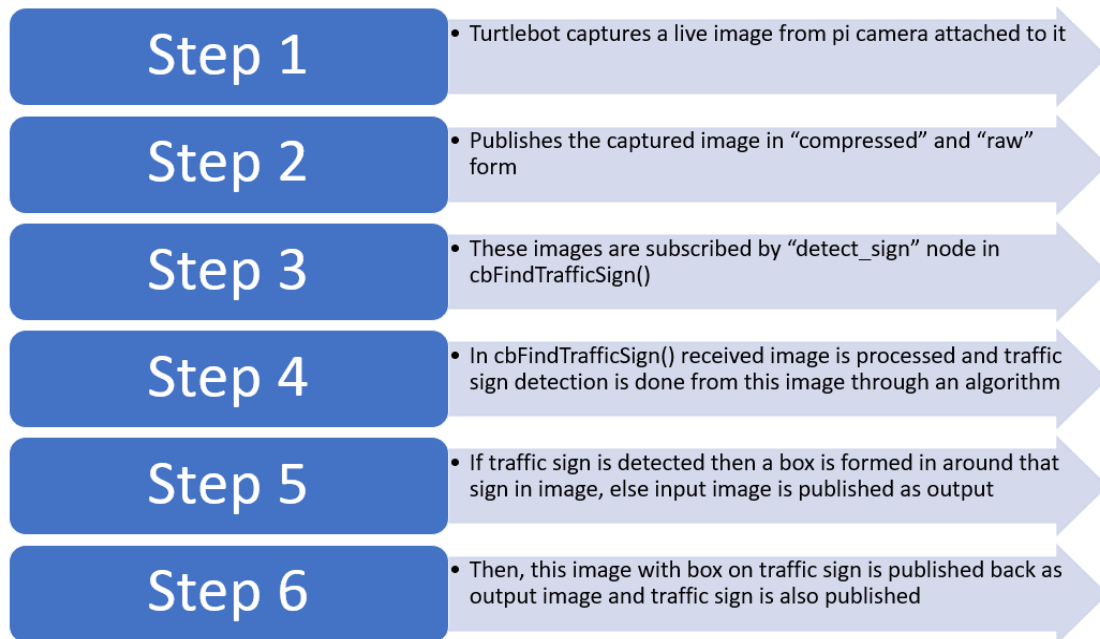


## CHAPTER 2

### THEORY & ALGORITHMS

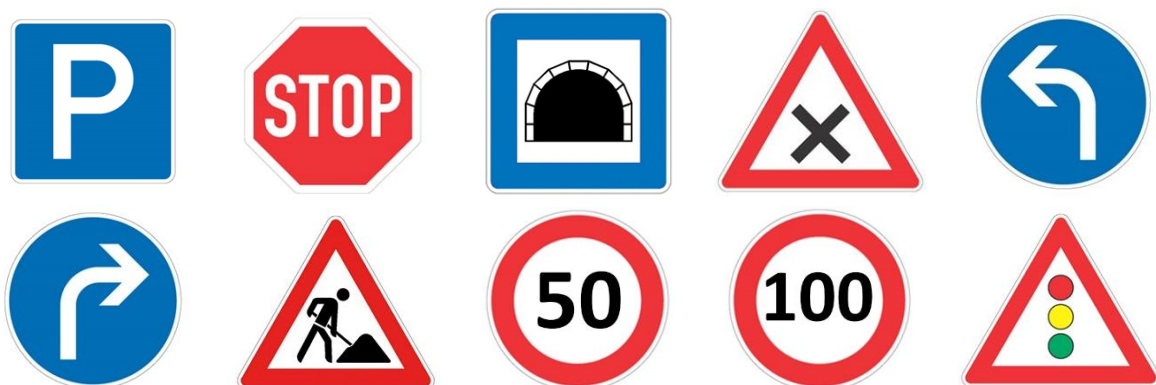
#### 2.1 Key Task

Primary aim of this project is to increase efficiency of traffic sign recognition in Turtlebot3. For detection of signs an algorithm is required which can fetch the traffic sign from image which is received from turtlebot. Further, for improving detection, first it is necessary to understand how it is working and what is the flow.



*Figure 1 Process of Traffic Sign Recognition*

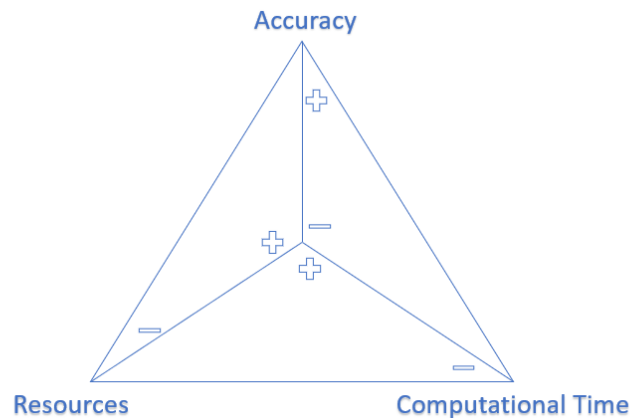
In this project, the key focus is on the “Step 4”, where input image is processed and traffic sign recognition is performed. In default package, SIFT algorithm in combination with FLANN matcher is used [2]. SIFT algorithm, extracts key points (features) from the reference images of each traffic sign from local data and find key points from the input image received too. Further, these fetched key points are matched with the help of FLANN and final decision is made that which sign is detected.



*Figure 2 Traffic Signs*

However, increasing efficiency does not mean having better accuracy. To increase efficiency, other aspects are also taken in consideration such as computational time and resources used, and SIFT is good in these aspects as it requires least computational time but accuracy is not good.

Additionally, it is also observed that accuracy, computational time, and resources used are in devil's triangle relation with each other. If one of them is improved then other one is adversely affected by that.



*Figure 3 Devil's Triangle*

Ideal condition of Devil's Triangle is that there will be a perfect triangle in between with limited resources, manageable computational time, and good accuracy but, that is an ideal state. So, if accuracy is achieved completely a lot of resources are consumed and computational time is also increased, which is not recommended at all. Thus, this project will try to make a traffic sign recognition system which is near to ideal state.

## 2.2 Algorithm Analysis

### 2.2.1 YOLO

Advantages:

It is current state of art, it is having less background error than other algorithms and it is also easy to understand and implement. It is also having different versions to set the complexity from "n" to "xl" which can increase accuracy with better version [3, 4]. It is single stage system so accuracy of detection is less but in real-time systems it is more reliable because of the less time it requires for detection [5].

Limitations:

One of the limitations of this algorithm is that it acquires a lot of memory space, storage space and time while training a custom dataset. However, currently it is said to be the fastest and most accurate algorithm for object detections.

### 2.2.2 CNN

Advantages:

It is one of the most accurate algorithms for object detection, as it deals with multiple layers of an image from input layer to hidden layers all are connected to each other tightly

[6]. There 3 stages for training a CNN model, first convolution then non linearity (ReLU) and then maxpooling, because of these stages there are more features for detection and as it uses maxpooling, it does not take a lot of space in the storage too. Further, even a big image can be trained as it decreases layers after every convolution and maxpooling. There are also various advance versions of CNN to overcome its fallbacks, like R-CNN (Regional-CNN) and Faster R-CNN [6, 7].

Limitations:

Even though there are various versions of CNN it takes long time for training a model it, and as it deals with numerous numbers of features it cannot be implemented in Real-Time systems as it takes time in few seconds for detection.

### **2.2.3 SIFT**

Advantages:

It is also single staged recognition method and it is having a good efficiency [1]. It can extract key points event from the smallest image. Annotations the traffic signs are done automatically in SIFT algorithm [1, 8], it does not require any human or manual support for doing that which is a great advantage as compared to CNN and YOLO. It is mainly not affected by the size of image, amount of light falling on the image or rotating it [9].

Limitations:

The key problem of SIFT with Turtlebot3 is that SIFT needs same quality image as of reference image, and once battery of turtlebot reduces image quality of frame is also not good, so detection is not done properly by SIFT in that case.

### **2.2.4 SURF**

Advantages:

It is an advance version of SIFT and it is having a function Upright which increase the accuracy of the feature extraction in less time as compared to the SIFT [10, 11].

Limitations:

However, these are theoretical findings, practical testing of it was not possible as it is still under patent, so using SURF through Opencv2 is still not possible.

### **2.2.5 ORB**

Advantages:

It is an advance version of FAST key point decider with modification of BREIF for vision tasks [12]. It fast and can detect more key points as compared to SIFT algorithm, in combination with BFMatcher it is used for traffic sign recognition. It is fast and requires less computational time than any algorithm [13].

Limitations:

It works differently in different environment such as it needs same environment as the reference image, it cannot work in a new environment.

### **2.3 Algorithm to be Implemented:**

After complete knowledge of the task and with brief analysis on each algorithm, it is quite easy to decide which algorithm can be suited best for the task. So, as per the information available, YOLO and ORB can be good fit for task and can be tested further along with SIFT. Reason for choosing YOLO is the promising accuracy shown theoretically in with less computational time [4], and ORB can be a better version of SIFT as it is said to be faster than SIFT and more accurate also [12]. But along with YOLO and ORB, SIFT will also be implemented as it is default algorithm so it will be better for comparison.

Even combination of these three algorithms is also possible and might also be implemented for test, such as YOLO with SIFT or YOLO with ORB. ORB with SIFT is not a good idea as but are borderline same and will not provide a productive improvement result for traffic sign recognition.

## CHAPTER 3

### IMPROVEMENT

#### 3.1 Area of Improvement

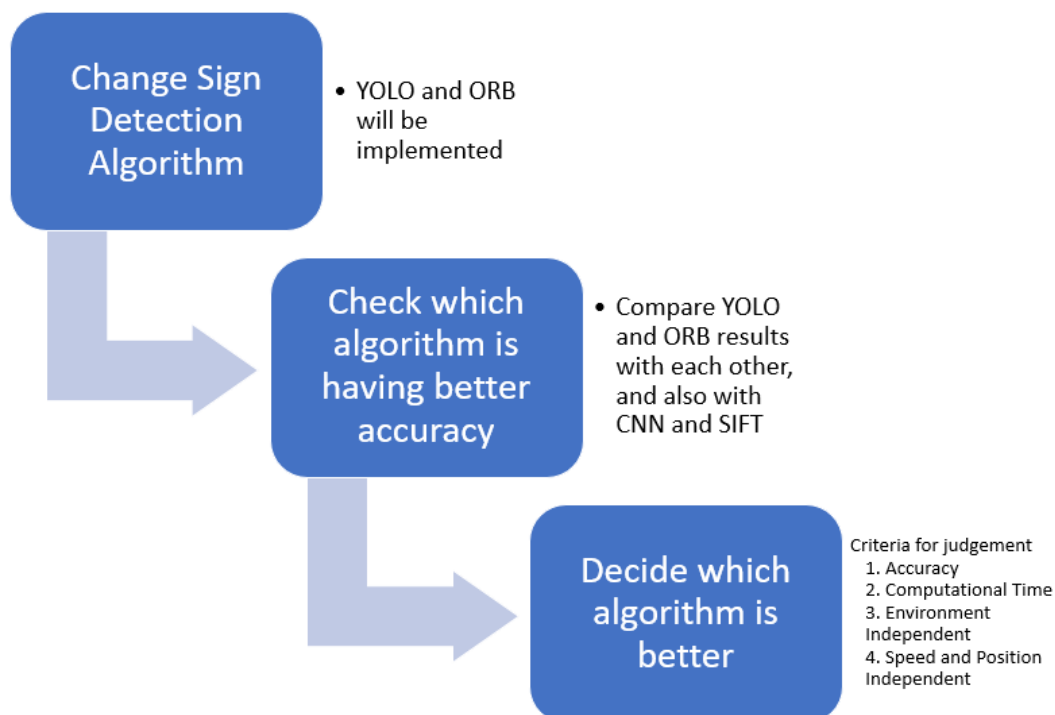
There are two main areas for improvement. First is the algorithm used for detection of traffic sign. Currently the algorithm used is SIFT, which is not accurate at all. It is very much environment dependent, so it cannot be used as a reliable algorithm for sign detection. To overcome this problem, different algorithms can be implemented other than SIFT such as YOLO, ORB and CNN. However, after deeply analysing CNN algorithm, it cannot be used too, as it is accurate but slow. So, other two algorithms will be tested in this project.

Secondly, some techniques can also be changed such as rather than using only one algorithm, there can be two algorithms. One as primary algorithm for traffic sign detection and one more algorithm as secondary algorithm, which will be triggered only for some specific signs. One more technique which can be added is to have a frame of detection. This will not only help to remove “True Negative” error, but can also help us to solve the problem of detection from different distance for different signs.

#### 3.2 Pathway to Improvement

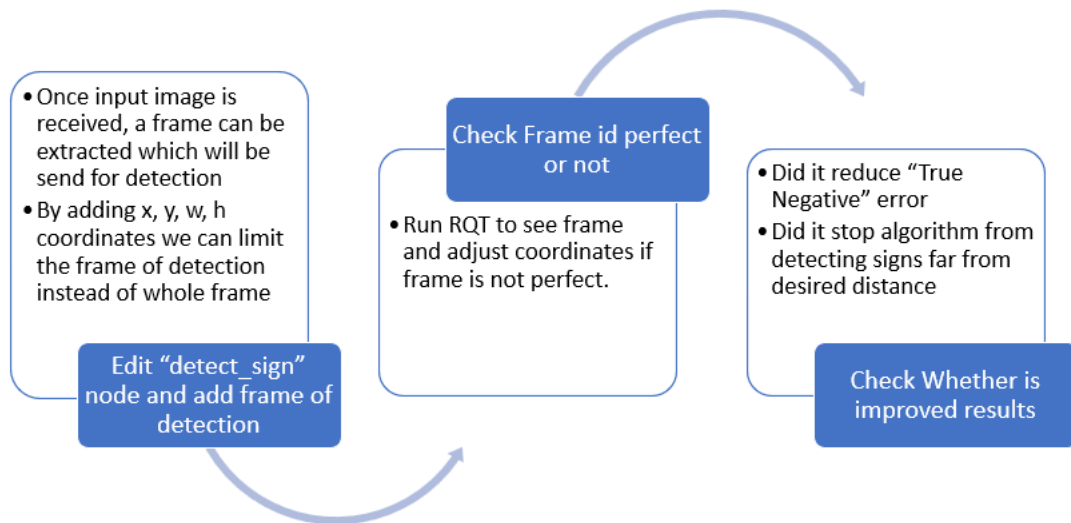
A proper roadmap is required for improvement. How improvements will be, in which segment improvement will be done, how to check whether it is an improvement or not.

##### 3.2.1 Algorithm Change

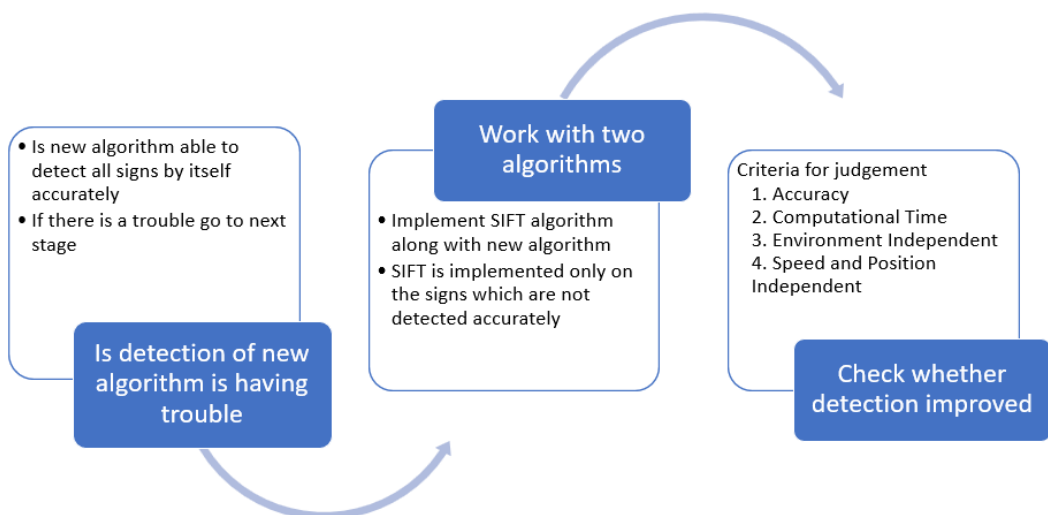


*Figure 4 Steps to Decide Algorithm*

### 3.2.2 Technique Change



*Figure 5 Steps of Adding Frame of Detection*



*Figure 6 Steps of having Two Algorithm*

### **3.3 Implementation of Improvement**

Various modifications are done for improving the traffic sign recognition, and those all are discussed above that what changes will be done. Now, how these changes will be implemented and where it should be implemented will be discussed.

#### **3.3.1 Implementing YOLO**

YOLO is the fastest object detection algorithm and it is also accurate, although implementation of it might not be complex but training your own model on custom dataset is quite a challenging and time-consuming work. For doing it a properly there are certain steps which are needed to be followed.

1. Training YOLO on custom dng your



## Directory 1: images (copy all annotated images in this directory)

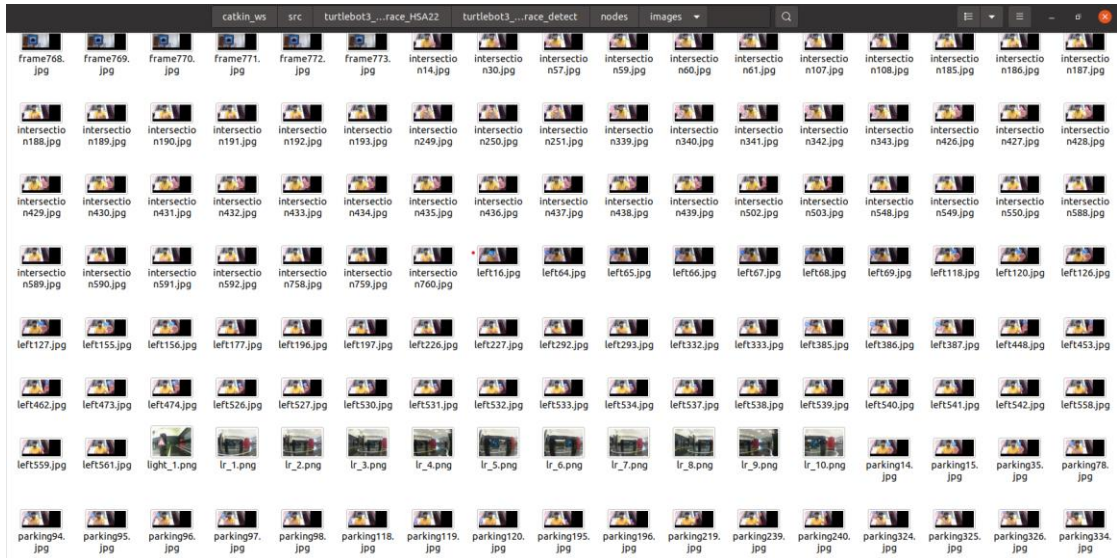


Figure 9 images directory

## Directory 2: labels (copy all labels from dataset exported in this directory)



Figure 10 labels Directory



Figure 11 Directories to be Added



Step 6: Create a “config.yaml” file

```
path:
/home/ravin/catkin_ws/src/turtlebot3_autorace_HSA22/turtlebot3_autorace_detect/no
des # dataset root dir
train: images # train images (relative to 'path')
val: images # val images (relative to 'path')

# Classes
names:
0: tunnel
1: Speedlimit_50
2: Speedlimit_100
3: intersection
4: parking
5: right
6: left
7: construction
8: stop
9: traffic_light
```

Step 7: Create and run a “train.py” file

```
from ultralytics import YOLO

model = YOLO("yolov8n.pt") # build a new model from scratch

results = model.train(data="config.yaml", epochs=50) # train the model
```

Step 8: Once training is completed a "runs" directory is created in working directory. This contains report of the training and weights of your trained model.

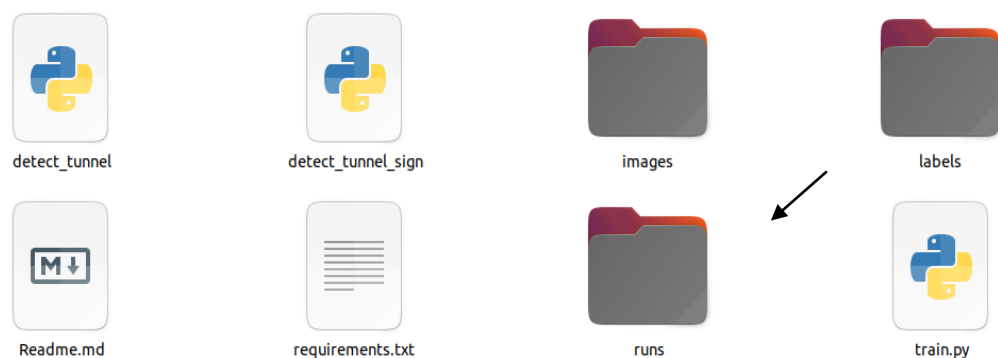


Figure 12 Directory Created Automatically

## 2. Using pre-trained custom YOLO model:

Step 1: Modify “detect\_sign” node to implement YOLO

Libraries added:

```
from ultralytics import YOLO
```

Load Model in code:

```
# path of model from current working directory of the code
model_path = os.path.join('.', 'runs', 'detect', 'train4', 'weights', 'best.pt')

#load model
model = YOLO(model_path)
```

Detection done through YOLO:

```
results = model.predict(cv_image_input)
result = results[0]
```

Fetching data about detected sign:

```
box = result.bboxes[0]
cords = box.xyxy[0].tolist() # Coordinates of Box
sign_ID = box.cls[0].item() # Class of the Box
confidence = box.conf[0].item() # Confidence
xmin,ymin,xmax,ymax = int(cords[0]), int(cords[1]), int(cords[2]), int(cords[3])
```

Checking which traffic sign is detected:

```
if sign_ID == 0 and confidence > 0.60:
    #tunnel
    msg_sign = UInt8()
    self.pub_traffic_sign.publish(msg_sign)
    msg_sign.data = self.TrafficSign.tunnel.value
    rospy.loginfo("Traffic Sign Detected: Tunnel")

    detected_image = 0
```

Repeat same for other signs

Transmit output back to Turtlebot:

```
elif detected_image == 0:

    self.output_Img = cv2.rectangle(cv_image_input, (xmin,ymin) , (xmax,ymax), color
    = (0,255,0), thickness= 2)

    final0 =
cv2.drawMatches(cv_image_input,None,self.img0,None,None,outImg=self.output_I
mg)

    if self.pub_image_type == "compressed":
# publishes traffic sign image in compressed type
        self.pub_image_traffic_sign.publish(self.cvBridge.cv2_to_compressed_imgmsg(f
inal0, "jpg"))

    elif self.pub_image_type == "raw":
```

```
# publishes traffic sign image in raw type
self.pub_image_traffic_sign.publish(self.cvBridge.cv2_to_imgmsg(final0,
"bgr8"))
```

Repeat it for other signs

Step 2: Place pre-trained model ('runs' directory) in working directory

In this case, 'runs' directory is supposed to place in '.ros' directory at home

Step 3: Run "detect\_sign" node with default steps of turtlebot for traffic sign recognition

Later, in topic [4.2.1 "On Table Results"](#) table 2 "On Table Test YOLO" can be referred for deciding whether YOLO should be used or not.

### 3.3.2 Implementing ORB

ORB is a better form of SIFT algorithm, it almost works in similar pattern as SIFT. It is having reference images for each sign and these images are used to find base key points for each sign and later these key points are matched with the help of BFMatcher to the key points of the input image [13]. It uses less computational time than YOLO as feature extraction of all reference images and initialization of ORB and BFMatcher is done before input image is subscribed. Steps for implementation of ORB are:

Step 1: Import Library OpenCV2

Step 2: Initiate ORB

```
self.orb = cv2.ORB_create(nfeatures=2000)
```

Step 3: Place all reference images in "image" directory

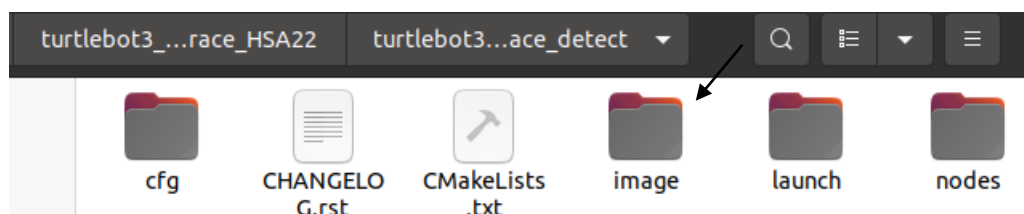


Figure 13 Reference Image Directory

Step 4: Read all referenced images in "detect\_sign" node

```
self.img2 = cv2.imread(dir_path + 'stop.png',0) # trainImage1
```

Repeat this step for all signs

Step 5: Extract features using ORB

```
self.kp2, self.des2 = self.orb.detectAndCompute(self.img2, None)
```

Repeat this step for all signs

Step 6: Initiate BFMatcher

```
self.bf = cv2.BFMatcher()
```

Step 7: Find key points from input image and match it with every reference images

```
kp1, des1 = self.ORB.detectAndCompute(cv_image_input, None)
if len(des1) != 0:
    matches2 = self.bf.knnMatch(des1, self.des2, k=2)
```

Repeat matching step for all signs

Step 8: Check which sign it matches and publish its sign detected

```
good2 = []
for m, n in matches2:
    if m.distance < 0.7 * n.distance:
        good2.append(m)

if len(good2) > MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good2 ]).reshape(-1, 1, 2)
    dst_pts = np.float32([ self.kp2[m.trainIdx].pt for m in good2 ]).reshape(-1, 1, 2)

    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    matchesMask2 = mask.ravel().tolist()

    mse = self.fnCalcMSE(src_pts, dst_pts)
    if mse < MIN_MSE_DECISION:
        msg_sign = UInt8()
        msg_sign.data = self.TrafficSign.stop.value
        self.pub_traffic_sign.publish(msg_sign)
        rospy.loginfo("TrafficSign 2: stop")

        image_out_num = 2

else:
    matchesMask2 = None
```

Repeat this for all signs

Step 9: Publish the image of sign detected with feature matching

```
elif image_out_num == 2:
    draw_params2 = dict(matchColor = (0, 0, 255),
                        singlePointColor = None,
                        matchesMask = matchesMask2,
                        flags = 2)

    final2 =
cv2.drawMatches(cv_image_input, kp1, self.img2, self.kp2, good2, None, **draw_params2)

    if self.pub_image_type == "compressed":
```

```
# publishes traffic sign image in compressed type
self.pub_image_traffic_sign.publish(self.cvBridge.cv2_to_compressed_imgmsg(final2, "jpg"))

elif self.pub_image_type == "raw":
    # publishes traffic sign image in raw type
    self.pub_image_traffic_sign.publish(self.cvBridge.cv2_to_imgmsg(final2, "bgr8"))
```

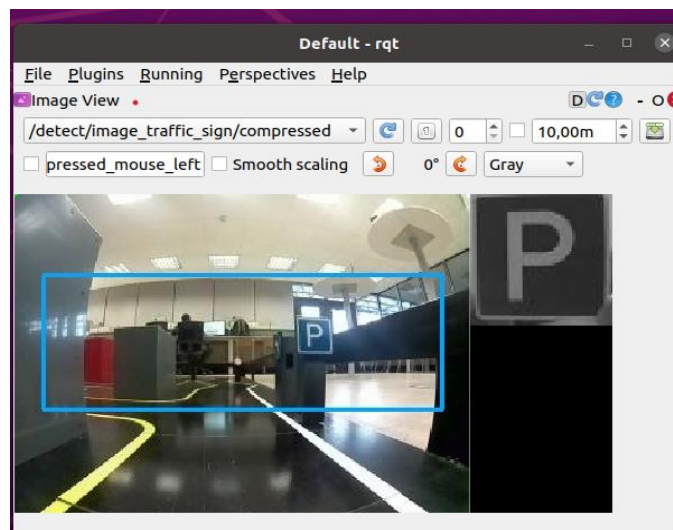
Finally repeat this line of code for all signs.

Step 10: Run “detect\_sign” node with default steps of turtlebot for traffic sign recognition

For better understanding of how accurate ORB is topic [4.2.1 “On Table Results”](#) in that table 3 “On Table Test ORB.”

### 3.3.3 Adding Frame of Detection

Frame can be useful for decreasing TN error and might help in stopping detection of signs at different distance. Frame can be placed on the image which is fetched and feature extraction or comparison is done on cropped image. Mainly line of codes will be added after input image is fetched.



*Figure 14 Frame of Detection*

Step 1: Adding coordinates for cropping input image

```
x,y,w,h = 20,60,280,100 # Coordinates to limit frame
```

Step 2: Providing restricted input to algorithms

ORB:

```
kp1, des1 = self.sift.detectAndCompute(cv_image_input[y:y+h,
x:x+w], None)
```

YOLO:

```
results = model.predict(cv_image_input[y:y+h, x:x+w])
```

Step 3: Make a rectangle in the frame

```
cv2.rectangle(cv_image_input, (x,y), (x+w,y+h), (255,165,0),2) #  
Showing Rectangle frame formed
```

### 3.3.4 Combining SIFT and YOLO

For making detection more accurate with less computational time a best way is to use another algorithm only for the signs which are not detected accurately by YOLO. So, in this case, SIFT algorithm is used in combination with YOLO only for some signs.

Line of code is same as for YOLO detection with SIFT detection only for two signs not all.

Step 1: Importing libraries for both YOLO and SIFT

Step 2: Loading pre-trained model of YOLO for detection

Step 3: Initiating SIFT algorithm

Step 4: Reading all reference images and extracting features only for limited signs

Step 5: Initiating FLANN matcher

Step 6: Detecting sign using YOLO and publishing its value, if it is one of the directions, then going for SIFT feature matching and publishing value if sign is matched

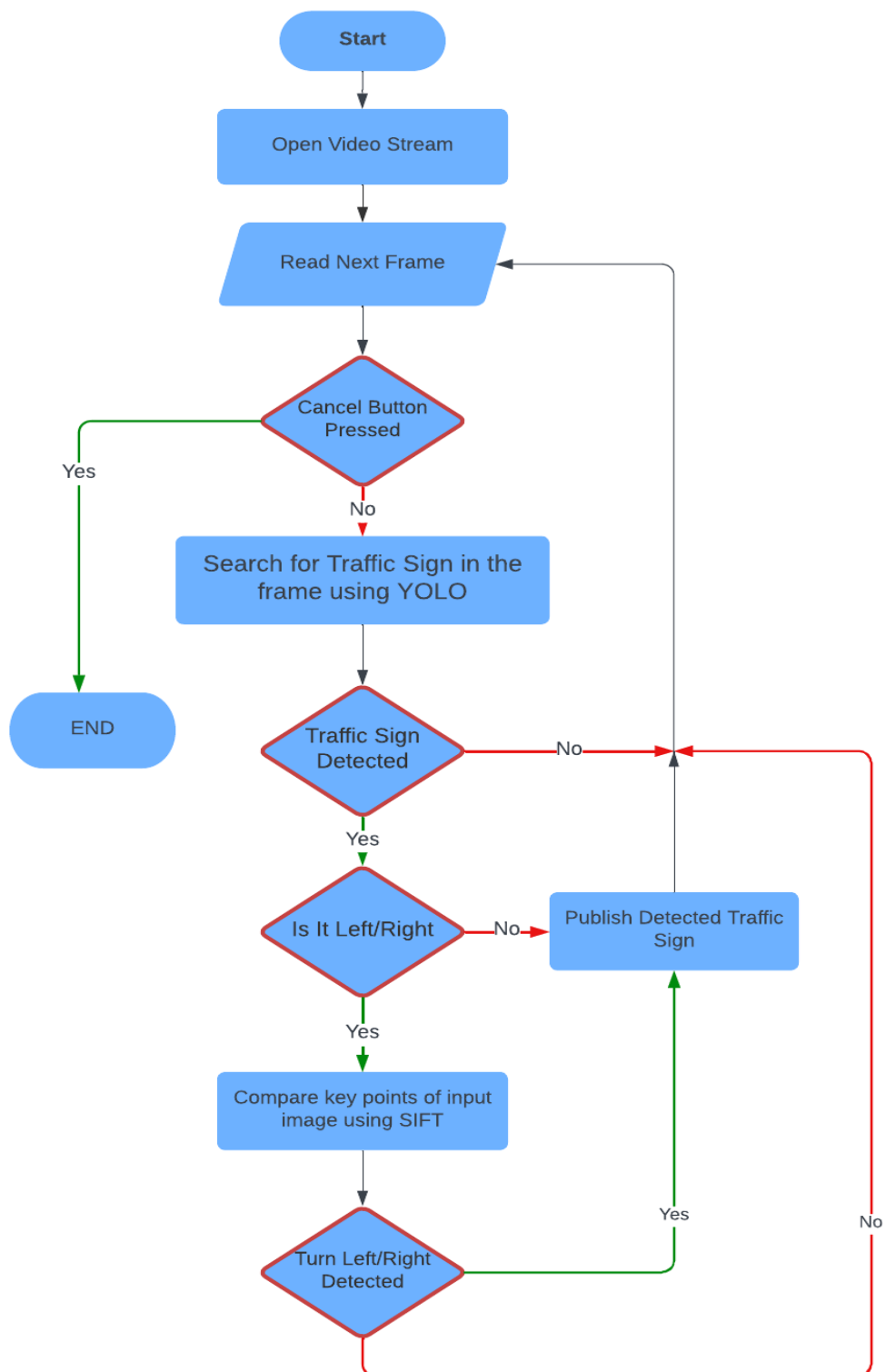
Step 7: Publishing output image with traffic sign detected and box on the traffic sign in input image

Results of using this approach are shown further in topic [4.2.1 “On-Table Results”](#) table 4 “On Table Test YOLO+SIFT” and in topic [4.2.2 “On-Track Results”](#) table 5 “On Track Test YOLO+SIFT.”

### 3.3.5 Final Approach

Based on all the approaches shown above and in combination of all techniques suggested, a final way of traffic sign recognition system is decided, which is having YOLO algorithm as the primary algorithm for detection of all signs and SIFT as secondary algorithm which is used only for verification in case of “Left” and “Right” signs. Also, ORB was not accurate at all and is also environment dependent, so it is omitted completely from the system.

Further, no frame is used for cropping input image, as in YOLO there are already no TN error and secondly, YOLO itself is a heavy algorithm using it with intrinsic and extrinsic camera already causes a lot load on processors. In that having one more strategy of cropping images was increasing computational time and power, which is not at all recommended in a real-time system.



*Figure 15 Flowchart of System*

## CHAPTER 4

### RESULTS

#### 4.1 Test-Cases

Test-cases are good for having complete knowledge of how good a system is and how efficient it is. However, there are various criteria to be kept in mind before have test-cases for a system. Particularly, in this project key criteria are pre-defined, those are:

1. Traffic sign recognition system should have near to perfect accuracy under following circumstances (Accuracy):
  - a. It should detect signs in all environment
  - b. It should be able to detect sign even it is placed at different position, other than where it will be on mini-city track
  - c. Turtlebot is supposed to be a prototype for an auto-pilot car, so it is moving at a certain speed and traffic sign should be detected at any speed.
2. System should not take much of a computational time for detection of the sign, as it is mentioned earlier turtlebot is supposed to be like a car and if it takes a lot of time then that can even cause an accident in worst case (Computational Time).

So, to fulfil all these criteria there are certain test-cases which are developed, such as complete examination of the system can be done.

Test	Criteria Fulfilled	Description
On – Table	Accuracy	<p>To test accuracy of turtlebot through TP, TN, FP, and FN</p> <ol style="list-style-type: none"> <li>1. Turtlebot is placed away from the track on a table which is completely new to it, so environment dependency is tested</li> <li>2. Turtlebot is not moving but the sign shown to turtlebot will be moving and at different positions in frame like left, right, up and down.</li> </ol>
On – Track	Accuracy	<p>To test accuracy of turtlebot through TP, TN, FP, and FN</p> <ol style="list-style-type: none"> <li>1. Turtlebot is placed in the mini-city track and it will perform sign detection along with lane detection, so motion test is done.</li> <li>2. Because it is moving signs can be detected from long and short both ranges</li> </ol>
Time Test	Computational Time	This test is done along with On-Table and On-Track, by adding few line of codes in detect_sign node

*Table 1 Test-Cases*



[illegible]

<b>FP</b>	2	1	1	3	2	2	1	1	0	2
<b>FN</b>	8	9	7	10	10	8	8	10	2	8
<b>Accuracy</b>	20%	10%	30%	0%	0%	20%	20%	0%	80%	20%

*Table 3 On Table Test ORB*

Observation: It is clear from these results that ORB is not at all accurate and it cannot be considered as a sign detection algorithm in the final system.

SIFT: It uses reference images for detection of key points and further matches those referenced key points with input image's key points.



*Figure 17 Reference Images*

	<b>Parking</b>	<b>Tunnel</b>	<b>Stop</b>	<b>Intersection</b>	<b>Construction</b>	<b>50</b>	<b>100</b>	<b>Light</b>	<b>Left</b>	<b>Right</b>
<b>TP</b>	1	7	10	8	9	10	8	5	7	10
<b>TN</b>	0	0	0	0	0	0	0	0	2	0
<b>FP</b>	1	1	2	2	2	2	1	5	3	2
<b>FN</b>	9	3	0	2	1	0	2	0	3	0
<b>Accuracy</b>	10%	70%	100%	80%	90%	100%	80%	50%	70%	100%

*Table 4 On Table Test SIFT*

*Figure 18 Results of SIFT*

Observation: It is observed that SIFT is better than ORB and in case of detection of “Left” and “Right” it is better than both YOLO and ORB.

YOLO with SIFT: SIFT algorithm was applied only on “Left” and “Right”

	Parking	Tunnel	Stop	Intersection	Construction	50	100	Light	Left	Right
<b>TP</b>	10	10	10	10	9	10	9	9	8	8
<b>TN</b>	0	0	0	0	0	0	0	0	0	0
<b>FP</b>	2	1	1	3	2	2	1	1	3	2
<b>FN</b>	0	0	0	0	1	0	1	1	2	2
<b>Accuracy</b>	100%	100%	100%	100%	90%	100%	90%	90%	80%	80%

*Table 5 On Table Test YOLO + SIFT*

Observation:

	Parking	Tunnel	Stop	Intersection	Construction	50	100	Light	Left	Right
<b>TP</b>	10	10	10	10	9	10	10	10	9	10
<b>TN</b>	0	0	0	0	0	0	0	0	0	0
<b>FP</b>	4	2	3	2	1	2	3	1	2	2
<b>FN</b>	0	0	0	0	1	0	0	0	1	0
<b>Accuracy</b>	100%	100%	100%	100%	90%	100%	100%	100%	90%	100%

*Table 6 On Track Test YOLO+SIFT*

Observation: Results clearly state that YOLO with SIFT shows same accuracy as it was on table.

#### 4.2.3 Computational Test

If YOLO algorithm is used for d00886469 0 595.2 841.92a(i)38(o)-19(n)20(:)] T,ETQq0.0000

Algorithm	On-Table (ms)	On-Track (ms)
YOLO+SIFT	196.9, 133.8, 135.1, 143.4, 119.9, 142.8, 143.1, 115, 151.6, 110.7	219, 144.1, 195, 167, 245.8, 178.2, 170, 212.7, 191.2, 159
YOLO	175.8, 150.2, 117.2, 115.5, 106.3, 145, 100.8, 105.9, 111.5, 120	140.5, 210.8, 191.8, 189.6, 188.3, 220.9, 210.4, 155.6, 170.3, 175.6
ORB	15.5, 16.2, 16, 19.3, 20.5, 11.2, 30.7, 25.6, 22.9, 13.6	50.9, 43.6, 48.2, 35.7, 30.2, 40, 28.9, 26.3, 37.4, 33

*Table 7 Measurements of Computational Time*

Algorithm	Average Time Taken	Standard Deviation
YOLO+SIFT	163.7ms	36.94
YOLO	155.1ms	39.36
ORB	28.25ms	11.69

*Table 8 Computational Time Test*

Observation: Results clearly state that YOLO with SIFT requires the most amount of computational time and ORB requires least computational time.

### 4.3 Results

Previously, the computational time for each algorithm was recorded also accuracy of On-Table and On-Track test for different algorithms for all signs was also measured. However, an average accuracy is also needed for comparing all the algorithms with each other.

Algorithm	Accuracy	
YOLO	89% (On-Table)	
SIFT	75% (On-Table)	
ORB	20% (On-Table)	
YOLO+SIFT	93% (On-Table)	98% (On-Track)

*Table 9 Accuracy of all Algorithms*

1. ORB:

It showed that ORB is with average 28.25ms processing time the fastest system in this situation for traffic sign detection, but the accuracy of ORB is 20% which proved that it cannot be trusted as primary algorithm for traffic sign detection.

2. YOLO:

It is having 89% accuracy which is better than ORB but it is still not up to the mark, there is still room for improvement in this case also. Even though computational time 155.1ms but because of good accuracy improvement can be done in this direction only.

3. SIFT:

It is having 75% accuracy which is better than ORB but not as good as YOLO, so going with default algorithm of turtlebot is not a good idea. However, SIFT is not completely removed from consideration as it is having better accuracy than YOLO in case of “Left” and “Right,” so it can be considered in combination of YOLO only for these two signs.

4. YOLO with SIFT:

It is having 93% accuracy in On-Table test and 98% in On-Track test, it covered the problem faced in using only YOLO, which was detection of left and right. However, it is having high computational time which is 163.7ms but by reducing frame per seconds it can be managed.

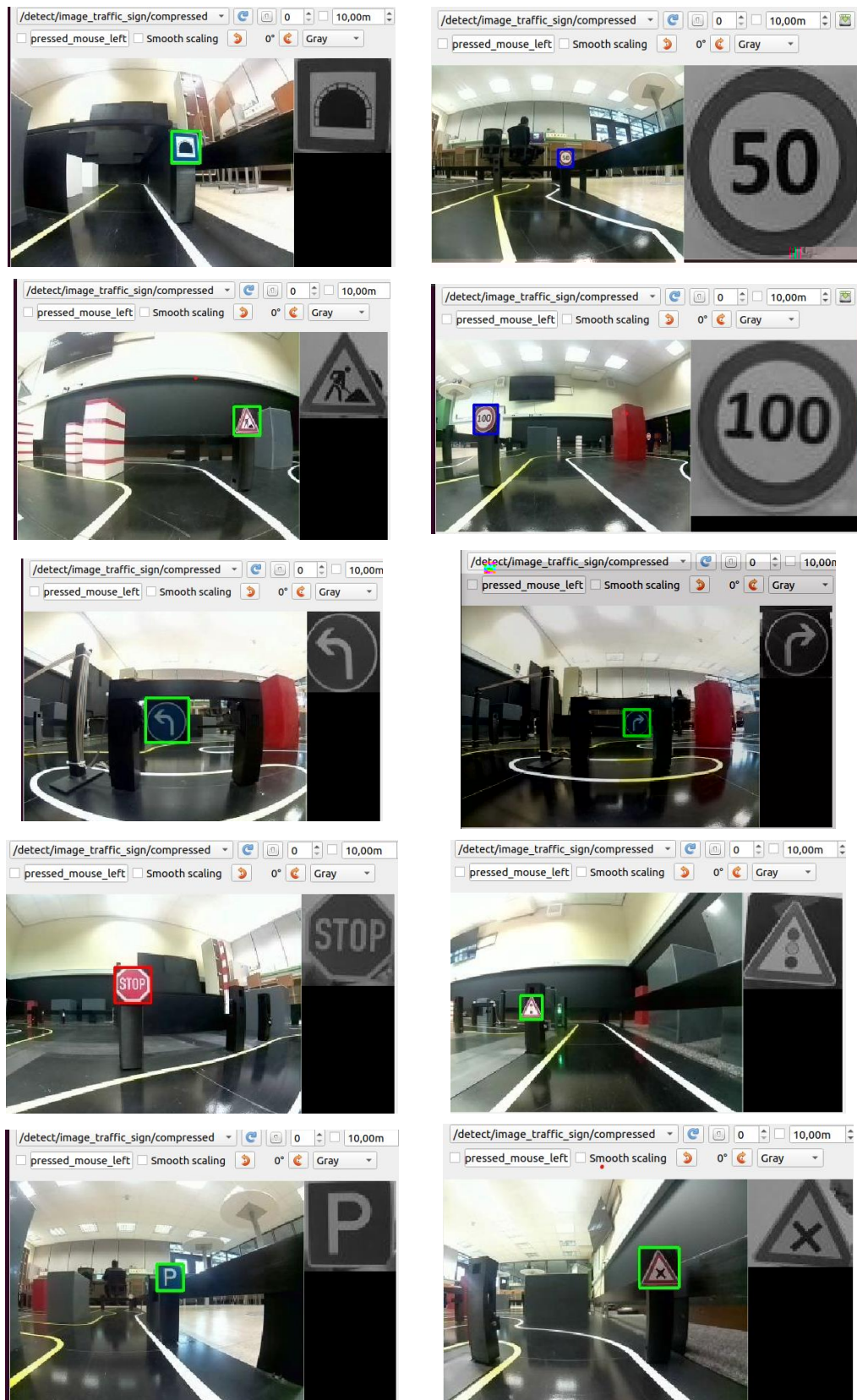


Figure 20 Results



## CHAPTER 5

### SUMMARY

#### 5.1 Conclusion

The primary aim of this project was to have an optimized system which can perform traffic sign recognition in a robust form with minimum errors and should also be real-time. In order to have a better efficiency different algorithm such as CNN, YOLO, ORB, SURF and SIFT were taken in consideration. Theoretical analysis of all these algorithms was done by listing out advantages and disadvantages of each algorithm, and only few of them were decided to be tested on turtlebot as it might increase accuracy of detection. From these algorithms, YOLO, ORB and SIFT were considered for testing on turtlebot3. However, using right algorithm is not the only thing required to increase efficiency, so various techniques were also implemented for example, have a frame of detection or combination of two algorithm. Moreover, these techniques did increase the accuracy but it also affected the computational time and to test all these things various test-cases were implemented.

Test-cases showed how good the systems are and which technique should be adapted, for getting this judgement two tests were considered, “On-Table Test” which dealt with environment independency, floating traffic sign and on computational power criteria and “On-Track Test” which gave accuracy based on range independency and mobility independency, along with accuracy efficiency was also tested by adding computational time test in this. The key findings of these tests were:

1. Between ORB and YOLO, YOLO had better accuracy than ORB as ORB was good but only if it tested in the same environment as its reference image, but YOLO works in any environment with any background.
2. Computational time is least in ORB and with YOLO it quite high if proper resources are not made available. This is because YOLO does most of its work on recognition after the image is accessed but ORB just does key points matching after input image is fetched.
3. YOLO itself is also not accurate for all signs, so to make it work on all signs SIFT is introduced in combination with YOLO, but only on signs which are not detected accurately not on all the images.

After all those findings and measuring computational time of all algorithms and techniques a final approach was decided which was to work with YOLO and SIFT, in which YOLO works on all the traffic signs and if YOLO detects traffic sign “Left” or “Right” it will not publish the results directly, but it will cross verify it with SIFT algorithm and if it is detected in SIFT too, then it will be uploaded.

Hence, it can be concluded that theoretical results of YOLO were not only promising but also till some extend and after doing certain modification and training custom model with precise annotation, YOLO with SIFT can give near to perfect traffic sign detection which is real-time and reliable.



## 5.2 Discussion

### 5.2.1 Problems faced

As good as the results are from this project, it required a lot of patience and numerous problems were faced to get a fruitful outcome. Certain problems were solved, while for some a solution was not found and a different approach was selected to overcome it.

First problem faced was that to work with YOLO and CNN algorithm, it requires good amount of knowledge of these algorithms along with literature research of them. This process demanded plenty of time and efforts, which eventually delayed the whole plan and deadline had to be extended. However, after working dedicatedly on this step a good result was achieved which helped to filter algorithms based on theoretical knowledge gained.

Second, specifically for YOLO algorithm, it requires time not only on the theoretical stage but it also takes time in training and validation stage, to have accurate results through YOLO nearly 500 images were captured of 10 traffic signs, on an average 50 images for each sign. Afterwards, these 500 images were supposed to be annotated manually, as annotation requires precision and for that couple of days were consumed. Once, custom dataset was ready then, it went for first training and in YOLO minimum of 45 EPOCHS are needed for accurate results and it took nearly 3 hours to train. If, results are not satisfying then new images are captured, again annotation is done, more number or EPOCHS are tried and again model is trained, about 12 times models were trained for getting this perfect result.

No. of Trails	No. of Epochs	No. of Images	Box Loss	Class Loss
2	65	500, 452	0.48, 0.54	0.37, 0.45
8	50	420, 400, 416, 452, 477, 490, 500, 520,	0.67, 0.66, 0.63, 0.50, 0.55, 0.59, 0.52, 0.51	0.55, 0.52, 0.53, 0.51, 0.49, 0.44, 0.45, 0.50
1	45	452	0.57	0.66
1	25	452	0.65	0.78

*Table 10 Trials of YOLO Model*

Third difficulty faced did not had any solution, but a compromise was done to achieve the target, the difficulty was the computational time taken by YOLO for recognition as YOLO itself is the fastest and accurate and it does not require much time but when it is worked on turtlebot along with intrinsic and extrinsic cameras it had latency in working with higher frames per second.

```
# drop the frame to 1/3 because of the processing speed.
if self.counter % 3 != 0:
    self.counter += 1
    return
else:
    self.counter = 1
```

This made whole system less realistic and no solution of this problem was found as YOLO works in tight knit algorithm and modifications in the structure of YOLO is very difficult. So, frames per seconds were reduced which made whole system real-time and even though it works on low frame per seconds it works perfectly with speed of turtlebot.

```
# drop the frame to 1/15 because of the processing speed.
if self.counter % 15 != 0:
    self.counter += 1
    return
else:
    self.counter = 1
```

### 5.2.2 Future Possibilities and Thoughts

Currently for getting accuracy of all traffic signs combination of two algorithm is used but this can also be solved.

Accuracy of YOLO detection can be improved in a way that whole system works only of YOLO and it will also help to reduce complexity of the system. But to do that a lot of time and wide amount of knowledge in YOLO is required. General solution which can be adapted is that all images are also augmented and then all those original images and augmented images are annotated manually with more precise boxes and even number of images can be increased. All this might help in overcoming the problem being faced while detection of few signs.

Second improvement is based more on computational side, YOLO needs a lot of computational power which makes whole system slow, so to solve this problem what can be done is that rather than taking YOLO predict function for detection of traffic sign, own function can be developed with the help of basis YOLO uses for detection which is DNN [15]. But again, this also requires proper knowledge of machine learning and neural networks and to learn everything is not possible to be done in a short span of time. Hence, I was not able to follow the possible solution as it required adequate training in this field and good amount of reading which would have not been possible in the provided deadline or in one term. To build own function which is same like YOLO is a bigger task than the whole project and working on it might have diverted the whole aim of the project.


Nevertheless, both these possibilities are what I think might be a possible solution to improve YOLO detection and reducing computational time rather than decreasing frame per second, but there is no surety this will be a right way to improve everything.

Finally, I would like to conclude the project with enormous experience I have gained in the field of Machine Learning and Autonomous Systems, which not only helped me to have growth in the field of automation but also gained my interest in a way that how there are still so many things to be discovered and how working on new technologies can increase the knowledge of coding and also changes the whole mindset of developing a code.

## CHAPTER 6

### REFERENCES

#### 6.1 References

- [1] E. Karami, M. Shehata, and A. Smith, “Image Identification Using SIFT Algorithm: Performance Analysis against Different Image Deformations,” Oct. 2017. [Online]. Available: <https://arxiv.org/pdf/1710.02728>
- [2] V. Vijayan and P. Kp. “FLANN Based Matching with SIFT Descriptors for Drowsy Features Extraction.” (accessed Aug. 22, 2023).
- [3] GitHub. “GitHub - ultralytics/ultralytics: NEW - YOLOv8  in PyTorch > ONNX > OpenVINO > CoreML > TFLite.” <https://github.com/ultralytics/ultralytics> (accessed Aug. 10, 2023).
- [4] M. Hussain, “YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection,” *Machines*, vol. 11, no. 7, p. 677, 2023, doi: 10.3390/machines11070677.
- [5] J. Terven and D. Cordova-Esparza, “A Comprehensive Review of YOLO: From YOLOv1 and Beyond,” Apr. 2023. [Online]. Available: <https://arxiv.org/pdf/2304.00501>
- [6] O. Hmidani and E. M. Ismaili Alaoui, “A comprehensive survey of the R-CNN family for object detection,” in *2022 5th International Conference on Advanced Communication Technologies and Networking (CommNet)*, Marrakech, Morocco, uuuu-uuuu, pp. 1–6, doi: 10.1109/CommNet56067.2022.9993862.
- [7] “MIT 6.S191: Convolutional Neural Networks - YouTube.” [https://www.youtube.com/watch?v=NmLK\\_WQBxB4&t=199s](https://www.youtube.com/watch?v=NmLK_WQBxB4&t=199s) (accessed Aug. 10, 2023).
- [8] A. M. Khan, “What is SIFT?,” *Educative*, 16 Jun., 2022. <https://www.educative.io/answers/what-is-sift> (accessed: Jul. 6, 2023).
- [9] YouTube. “How does the SIFT algorithm work? | 3D Forensics.” [https://www.youtube.com/watch?v=m\\_XtEl-HUIg](https://www.youtube.com/watch?v=m_XtEl-HUIg) (accessed Jul. 6, 2023).
- [10] H. Bay, T. Tuytelaars, and L. van Gool, “SURF: Speeded Up Robust Features,” in 2006, pp. 404–417, doi: 10.1007/11744023\_32.
- [11] Daliyah S. Aljutaili, Redna A. Almutlaq, Suha A. Alharbi, and Dina M. Ibrahim, “A Speeded Up Robust Scale-Invariant Feature Transform Currency Recognition Algorithm,” 2018, doi: 10.5281/ZENODO.1316790.
- [12] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. “ORB: An efficient alternative to SIFT or SURF.” (accessed Aug. 22, 2023).
- [13] “OpenCV: ORB (Oriented FAST and Rotated BRIEF).” [https://docs.opencv.org/3.4/d1/d89/tutorial\\_py\\_orb.html](https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html) (accessed Aug. 10, 2023).
- [14] “Train Yolov8 object detection on a custom dataset | Step by step guide | Computer vision tutorial - YouTube.” <https://www.youtube.com/watch?v=m9fH9OWn8YM> (accessed Aug. 10, 2023).
- [15] “OpenCV: YOLO DNNs.” [https://docs.opencv.org/4.x/da/d9d/tutorial\\_dnn\\_yolo.html](https://docs.opencv.org/4.x/da/d9d/tutorial_dnn_yolo.html) (accessed Aug. 10, 2023).