Analysis and Implementation of an Efficient Traffic Sign Recognition Based on YOLO and SIFT for Turtlebot3 Robot

Stefan Twieg and Ravin Menghani
Department of Electrical, Mechanical and Industrial Engineering, Anhalt University of Applied Sciences, Bernburger Str. 55, Köthen, Germany
stefan.twieg@hs-anhalt.de, ravinvijaybhai.menghani@student.hs-anhalt.de

Keywords: Traffic Sign Recognition, Machine Learning, YOLO, SIFT, Turtlebot3, ROS, Robot Operating System, Convolutional-Neural-Network, CNN.

Abstract: Traffic Sign Recognition (TSR) is one of the key aspects for autonomous driving and it plays a vital role to make autonomous driving successful, but that's only possible if TSR is efficient enough and reliable. This work addresses exploration of simple and fast to implement options for robotic applications. For analysis and implementation, we are focusing on a Turtlebot3 Robot (TB3). Various potential TSR algorithms are evaluated in different test-cases with the goal of developing an optimized TSR with accurate results for German traffic signs. Therefore, the robot was tested on its own Mini-City track. On this Track we started to detect the signs with a simple Scale-Invariant Feature Transform (SIFT). However, the accuracy of SIFT was showing limitations for the use within TSR on mini-city-Track. This approach focuses on educational use where limitations and simple applications of autonomous driving are investigated. A review of state-of-art algorithms was done, to evaluate and improve accuracy. For example, Oriented FAST and Rotated Brief algorithm (ORB), You Only Look Once (YOLO) and SIFT algorithm was tested on TB3 in a way that all important criteria are fulfilled along with system being real-time. Regarding YOLOv8 a custom dataset and training is performed. The YOLO-model achieves 99.5% in terms of mean Average Perception (mAP@0.5) for all classes. In summary, as a powerful alternative to work with, YOLOv8 was identified. Standalone or in combination with SIFT a TSR system is shown which can work impacted by several environmental conditions. Based on evaluation of three algorithms an optimized code was developed in which YOLOv8 and SIFT were used in combination as a well performing TSR algorithm, which has above 95% accuracy for each traffic sign tested.

1 INTRODUCTION

As observed in [1], TurtleBot3 can detect different signs using the SIFT algorithm that compares the source image and camera image. Additionally, as a customizable robot [2], TB3 has the flexibility to modify its functionality including Traffic Sign Recognition to improve its accuracy for autonomous driving task. As a starting point the autonomous driving "Autorace-Package" for Robot Operating System is recommended [1]. This work addresses evaluation as well as identification of alternative techniques to SIFT. After extensive research and analysis, one algorithm was found to be superior to all others. It was possible to develop an improved version of the package, which significantly increases accuracy and performance over the previous version. The work is noteworthy because it explores a variety of techniques and concludes with a solution that has a wide range of applications. The only limitation is that we use a ROS version that requires the use of an older version of Ubuntu, but this is neglected in terms of optimal performance of the algorithms. In summary the main objective of this paper is to improve the accuracy of traffic sign recognition and develop an optimized autonomous driving (so called Autorace) package for TB3. This is achieved by implementing various algorithms such as R-CNN, SIFT, ORB, and YOLO, and determining the algorithm that performs best. Once the best algorithm has been found, a technique that best suits our algorithm criteria can be implemented to improve efficiency.

2 METHODS AND ALGORITHMS

For sign recognition, an algorithm is needed that can detect and recognize the traffic sign from the image received by robot. Furthermore, in order to improve the recognition, it is first necessary to understand how it works and what the flow is. As shown in Figure 1, first the TB3 captures the image and then transmits it in raw and compressed form to the system. After that, the system will pass the image to a certain function and this function will detect whether a traffic sign is present in the image or not. If a traffic sign is present, a frame is formed with the coordinates of the image and the modified image is published. So, if a traffic sign is detected, other nodes in the system will know about it.

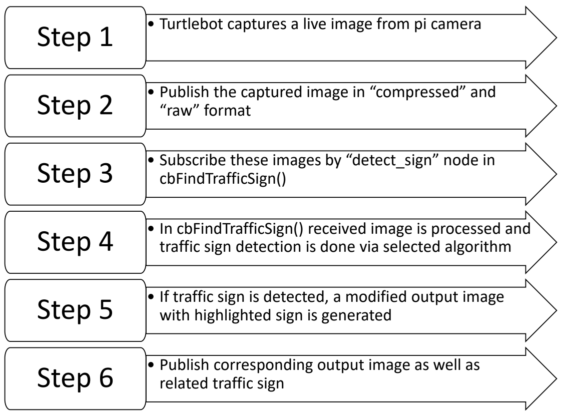

Step 1
• Turtlebot captures a live image from pi camera
Step 2
• Publish the captured image in "compressed" and "raw" format
Step 3
• Subscribe these images by "detect_sign" node in cbFindTrafficSign()
Step 4
• In cbFindTrafficSign() received image is processed and traffic sign detection is done via selected algorithm
Step 5
• If traffic sign is detected, a modified output image with highlighted sign is generated
Step 6
• Publish corresponding output image as well as related traffic sign

Figure 1: Process of traffic sign recognition.

The focus is on "Step 4", where the input image is processed and traffic sign recognition is performed. This is a usual classification task within machine learning. In the autorace-package [1], the SIFT algorithm is used in combination with the FLANN matcher. The SIFT algorithm extracts key points (features) from the reference images of each traffic sign from local data and also finds key points from the received input image [3, 4]. These key points are then compared with FLANN. The final decision is made to publish whether or not a sign was detected [4].

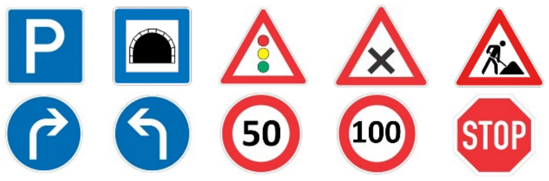


Figure 2: Traffic signs.

Figure 2 shows which traffic signs shall be recognized using the traffic sign recognition system in TB3. As in Figure 3 an artificial testing environment the Mini-City-Track [1] at Anhalt University of Applied Sciences has all of these signs implemented as part of autonomous driving test.

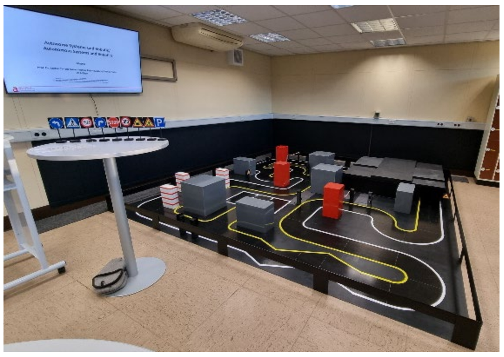


Figure 3: Mini-City-Track at Anhalt University of Applied Sciences.

However, improving efficiency does not mean improving accuracy. In order to increase efficiency, other aspects were also taken into consideration, such as computational time and resources used. As a first impression, SIFT was good in these aspects as it requires the least computational time, but accuracy was limited. It was also observed that accuracy, computation time and resources used were in a devil's triangle relationship with each other, as shown in Figure 4. If one was improved, the other was adversely affected.

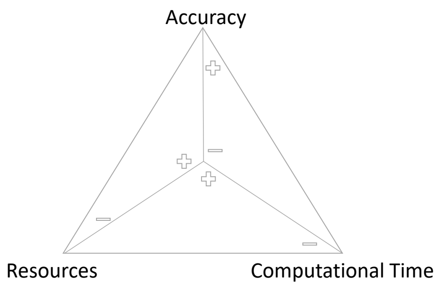


Figure 4: Devil's Triangle.

The ideal state of the Devil's Triangle in Figure 4 is that there is a perfect balance in between with limited resources, manageable computation time and good accuracy, but that is an ideal state. So, usually engineering needs decisions, if accuracy is fully achieved, a lot of resources might be consumed and the computation time probably also increase with complexity, which is not our intention at all. We like to keep it simple like SIFT. This is also important for the educational purpose, where students have limited time to understand and explore functionality. So, in this paper, we present a TSR system that is close to the ideal state.

2.1 Algorithm Analysis

Various algorithms exist for use in terms of object detection, yet the current cutting-edge technique is "Convolutional Neural Networks (CNN)." [5] CNN serves as a foundation for deep learning, wherein one algorithm handles feature extraction and comparison.

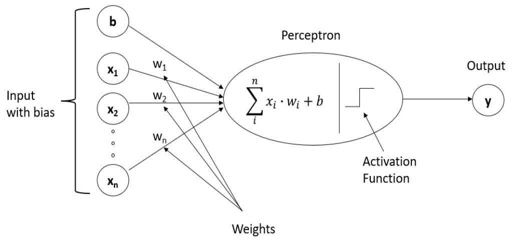


Figure 5: Formula for Neural Networks.

Figure 5 illustrates the basement of Neural Networks – the perceptron is equivalent to a simplified neuron in the human brain. In the perceptron, the inputs are multiplied by their respective weights, and the sum of these value is lead to a (non-linear) output with the assistance of an activation function. Based on this simple representation a network of Neurons consisting of several Neurons in several layers are used where the connection between Neurons can be varied in all directions. Out of that many possible neural network structures can be created, like CNN.

Usually two versions of CNN are used for TSR tasks, which differ based on the number of stages used for detection - One-stage detection, where object classification and bounding boxes are acquired simultaneously, and two-stage detection, where sign area is identified as bounding box within picture first and object classes recognized in a second stage from these. This study compares these deep learning techniques with each other and classical machine learning techniques. The analysis results will determine the most effective algorithm among the currently available techniques for object recognition.

2.1.1 YOLO (You Only Look Once)

According to [6] "YOLOv3: An Incremental Improvement", the YOLO algorithm encodes contextual information about classes and their appearance implicitly during both training and test periods. Hence it can be asserted that YOLO is a single-stage detection system employed by using CNN as the primary principle of detection. YOLO is currently the state-of-the-art technology in terms of having lower background error rates compared to other algorithms. Furthermore, it is easy to comprehend and execute [6]. It also features a range of models to adjust the complexity from "n" to "xl", enhancing accuracy with a superior model [7,8]. As it is a single-stage system, the detection precision is lower, but it is more suitable in real-time systems because it takes less time to detect [8]. However, this algorithm has a drawback - while training a custom dataset, it requires a significant amount of memory and storage space. However, it is currently considered to be the swiftest and most precise algorithm for identifying objects [8].

2.1.2 R-CNN (Region based CNN)

This algorithm operates in two stages, beginning with a proposal of the region of interest where an object may be located. The output of the first stage is then sent to the second stage to classify the object within the designated region of interest. The resulting outcome is a bounding box surrounding the object, along with its classification within that box [9]. It is considered one of the most precise algorithms for detecting objects, as it processes various layers of an image, from the input layer to the hidden layers, all of which are interconnected [10]. The process for training an R-CNN model begins by identifying the region of interest, followed by convolution, non-linearity (using ReLu), and ultimately, maxpooling. This multi-stage approach provides additional features for detection, while the use of maxpooling ensures that it does not consume excessive storage space. Furthermore, a sizeable image can be trained by reducing layers following each convolution and maxpooling. There are also advanced versions of R-CNN available, such as Faster R-CNN, which aim to rectify its shortcomings [10,11]. Despite its various versions, R-CNN is time-consuming to train a model and not suitable for Real-Time systems as it takes a few seconds for detection due to its two-stage detection process [12].

2.1.3 SIFT (Scale-Invariant Feature Transform)

This is a machine learning algorithm used to extract features from a given image. With the help of a feature matcher, the image can then be classified using key points fetched by SIFT. As its name suggests, the algorithm is not affected by the scale or rotation of the image [3]. It can extract key points even from the smallest image [3]. The traffic signs are automatically annotated by the SIFT algorithm, without the need for human or manual intervention. This is advantageous compared to R-CNN and YOLO [13]. The method is not significantly affected by image size, lighting conditions, or rotation. The main issue with TB3 and SIFT is that it requires an image of the same quality as the reference image.

2.1.4 ORB (Oriented FAST and Rotated BRIEF)

ORB may be employed as an alternative to the SIFT algorithm, resulting in improved efficiency. ORB is constructed using established FAST key point detection and BRIEF descriptors. Because the FAST method is utilized, key points can be detected more rapidly and efficiently with the aid of the BRIEF descriptor. This modified version of the FAST key point decider is used for vision tasks and has superior key point detection abilities compared to the SIFT algorithm [14]. When combined with the BFMatcher [15], it is suitable for traffic sign recognition. Additionally, it is a quick algorithm that requires less computational time than others. However, it operates differently in various environments (background dependency), best performance is reached by using the same environment as the reference image.

2.2 Algorithm Selection

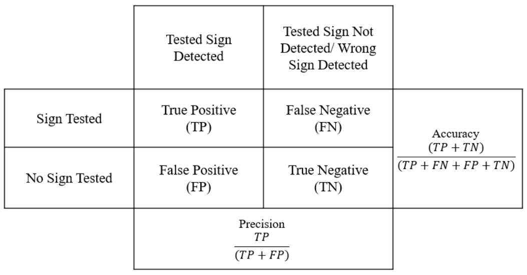
Considering the task and with brief analysis on each algorithm of interest, we decided which algorithm can be suited best for the task. So, YOLO and ORB might be good fit and can be tested/ compared further along with SIFT. Reason for choosing YOLO is the promising accuracy shown in various applications with low computational time [7, 8, 12], and ORB might be a more comparable approach but better version of SIFT as it is said to be faster than SIFT and more accurate also [14]. R-CNN is highly accurate but when working with real-time systems it cannot be used as it is having two stage detection which affects the time used for detection of sign adversely. But along with YOLO and ORB, SIFT will also be implemented as it is the default algorithm for comparison. Furthermore, combinations of these three algorithms are possible and might also be implemented for test, such as YOLO with SIFT or YOLO with ORB.

3 CRITERIA

After carrying out the theoretical analysis, it is necessary to apply the algorithms on TB3 to determine performance and to judge the algorithms. To assess their efficiency, specific criteria must be established:

1) Accuracy for each individual sign;
2) Overall-accuracy for all signs;
3) Calculation/ processing time (Computational performance and Complexity).

Judgement shall be done by TP, TN, FP, FN [16] as with these four categories all the possible outcome can be measured and placed in at least one of these categories, which is helpful to determine the accuracy of the system.



	Tested Sign Detected	Tested Sign Not Detected/ Wrong Sign Detected	
Sign Tested	True Positive (TP)	False Negative (FN)	Accuracy $\frac{(TP+TN)}{(TP+FN+FP+TN)}$
No Sign Tested	False Positive (FP)	True Negative (TN)	
	Precision $\frac{TP}{(TP+FP)}$		

Figure 6: Evaluation Criteria [16].

Figure 6 illustrates. that True Positive is when system detects the sign that was showed, True Negative is that system detects no sign and no sign was shown. False Positive is when system detects a traffic sign even though there is no traffic sign shown, and False Negative occurs when wrong or no traffic sign is detected even though traffic sign is shown. To calculate accuracy and precision two formulas were used from [16] and illustrated in Figure 6. To measure precision, true positive is divided by summation of true positive and false positive, and to calculate accuracy, summation of true positive and true negative is divided by summation of all the four aspects. Furthermore, it is crucial to consider certain impacts when testing the accuracy of each sign individually as well as collectively. Firstly, environmental impacts have to be considered during sign recognition. Light is one of these environmental impacts, for example, the system might be trained in an environment with more or less illumination compared to the environment where the system is utilized. Therefore, testing should be conducted while keeping environmental constraints in mind. Secondly, the positioning of signs is crucial as they are not always in the same place; they can be located on the right, left, top, bottom, near or far. To overcome this problem, testing the signs in different positions is necessary. Furthermore, it is important to note that for real-time systems, computational time alone is not sufficient. Instead, it is necessary to use it to determine how well the system can detect signs whilst in motion and up to which speed it can detect them without any issues.

4 EXPERIMENTAL SETUPS

The algorithms are tested on the TB3 using multiple test-cases, each with its own significance for fulfilling different criteria.

4.1 On-Table and Mini-City-Track Test

In On-Table test TB3 is placed in laboratory on the table to test functionality at defined conditions. During Mini-City test, the TB3 is placed on a track with predefined tasks and signs.

4.1.1 Environment Independency

To test this, the robot is placed on a table to assess its environment independence, mainly focusing on background interferences, as illustrated in Figure 7. The outcome is recorded, analyzed and a decision made accordingly. The traffic sign recognition system is activated after TB3 was placed on table to evaluate the signs presented in front of it.

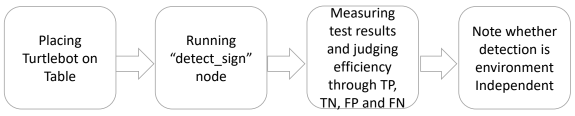

Placing Turtlebot on Table
Running "detect_sign" node
Measuring test results and judging efficiency through TP, TN, FP and FN
Note whether detection is environment Independent

Figure 7: Environment independency test procedure.

4.1.2 Dynamic Positioning Test

There are two dynamic positioning tests: the first involves a moving traffic sign while the TB3 remains stationary, and the second involves the robot in motion on our Mini-City track. For the first test, robot is placed on table with the traffic sign recognition system activated. Different signs are then tested in a continuous manner, going from left to right, up and down, and vice versa. Results are recorded to determine whether the traffic sign recognition system operates independently in positioning. In the second version, TB3 is placed on the track and the traffic sign recognition system is triggered. Additionally, the lane detection function is activated, enabling the robot to move automatically on the signed roads. It shall be determined whether the system can detect signs when the TB3 is in motion or not.

4.1.3 Track Range Test

To determine if the TB3 can effectively detect signs on a track from short or long distances (greater and less than 12cm). This test is used for the evaluation of whether sign detection is affecting the real-time system by means of detecting signs within a required time equivalent to the distance of sign and the related reaction to it from TB3 functionality.

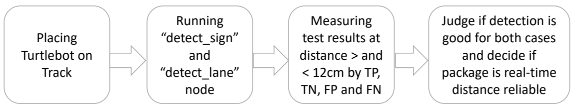

Placing Turtlebot on Track
Running "detect_sign" and "detect_lane" node
Measuring test results at distance > and < 12cm by TP, TN, FP and FN
Judge if detection is good for both cases and decide if package is real-time distance reliable

Figure 8: Range test.

Figure 8 outlines the procedure for conducting the test. Prior to the assessment, markings at precisely 12cm in front of all signs are made. These markings serve to determine the accuracy of the robots sign detection prior to and after crossing the mark. The findings from both criteria are combined to make a final determination.

4.1.4 Computational Power Testing

For our case the best method for measuring computational power is to measure the time required to detect a sign.

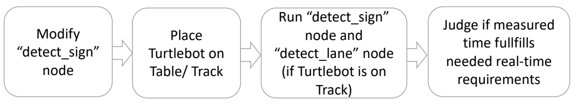

Modify "detect_sign" node
Place Turtlebot on Table/ Track
Run "detect_sign" node and "detect_lane" node (if Turtlebot is on Track)
Judge if measured time fullfills needed real-time requirements

Figure 9: Computational Time Test On-Table/Track.

Figure 9 outlines the method for measuring computational time. Initially, modifications are made to the "detect_sign" node by implementing code for time calculation. The code is inserted after accessing the input sign from the TB3 camera to start the recording process.

```
import time as t
start_time = t.time()
```

Secondly, to measure the time required, following code is added after the node publishes the output image back to the topic it subscribed from.

```
end_time = round(t.time() -
start_time,2) * 1000
print(end_time, "ms")
```

The TB3 is placed on the table or track and the traffic sign recognition system is turned on to detect the sign, by this change the time required for detecting the sign is now also recorded.

5 IMPLEMENTATION AND TESTING

5.1 Implementing Algorithms

The first modification being executed involves altering the sign detection algorithm. As depicted in Figure 10, apart from SIFT, YOLO and ORB algorithms were implemented on the TB3 to assess their effectiveness regarding TSR.

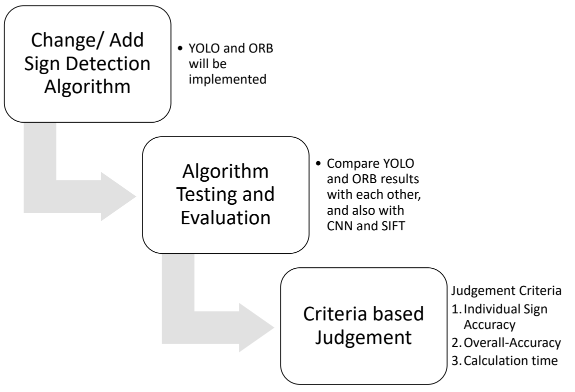

Change/ Add Sign Detection Algorithm
• YOLO and ORB will be implemented
Algorithm Testing and Evaluation
• Compare YOLO and ORB results with each other, and also with CNN and SIFT
Criteria based Judgement
Judgement Criteria
1. Individual Sign Accuracy
2. Overall-Accuracy
3. Calculation time

Figure 10: Steps to decide algorithm.

5.1.1 Implementing YOLO for TSR

YOLO is the fastest object detection algorithm [8], although implementation of YOLO is not complex but training custom model on custom dataset is time-consuming [8]. For doing it a properly there are certain steps which are needed to be followed.

5.1.1.1 Training YOLO on a Custom Dataset

The initial stage involves installing the "Ultralytics" library to use in our case YOLOv8. After installation, import the library into the node for application purposes. To train the model, a custom dataset is necessary. To create a dataset, a multitude of images, showing various backgrounds under different lighting conditions, and in different locations are needed, as shown in Figure 11.

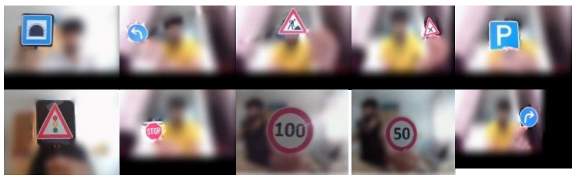


Figure 11: Creating images for TSR dataset.

After creating the image dataset, annotation of dataset is necessary. Using online tools, such as Roboflow [17] and CVAT [18], the images can be annotated. Specifically, for this paper, CVAT was utilized. A new project was created, with all necessary classes/labels specified. Afterwards, individual labels are generated for each class and relevant images are assigned to each label. To summarize, the images are manually annotated by dragging a labelled box to the location of the traffic sign within the image.

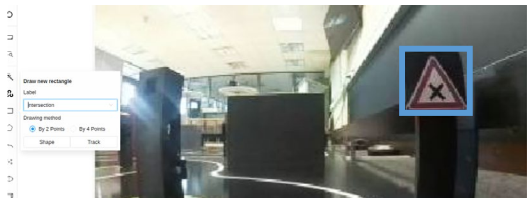


Figure 12: Annotation of sign (image is recorded on Mini-City-Track) with CVAT.

When the images are annotated as displayed in Figure 12, with a box outlining the traffic sign in each image, the data is exported as a dataset, selecting YOLO version to ensure compatibility. Output consists of a label text file corresponding to each image which is located in "labels" directory, while all the images are copied to the "images" directory.

A "config.yaml" file is generated in the same environment. This file specifies the path of the training dataset and defines all the classes/ labels. Within Python [20], which is used for training the YOLOv8n model on our custom dataset, this file is accessed. The model is trained using the "train" function that is pre-defined in the ultralytics library.

```
model = YOLO("yolov8n.pt"
results =
model.train(data="config.yaml",
epochs=65)
```

Finally, a successful training of the model is indicated by a mean Average Perception (mAP@50) near to 1. The "runs" directory, generated during the YOLO training, can then be utilized as a custom model for our custom dataset. This directory contains weights, training results, and output images from various validation datasets.

5.1.1.2 Using Pre-Trained YOLO Model

Once the model has been trained, it can then be utilized to recognize traffic signs in images obtained from the TB3. To initiate sign recognition, modification is required to the "detect_sign"-node of autorace-package. The pre-trained YOLO model is imported from the "ultralytics" library and subsequently loaded as the model.

```
from ultralytics import YOLO
model_path = os.path.join('.',
'runs', 'detect', 'train',
'weights', 'best.pt',)
model = YOLO(model_path)
```

Afterwards, the YOLO model's "predict()"-function is utilized on the input image to detect signs, and the resulting tensor is analyzed to determine whether any signs were detected. If a traffic sign is detected, the coordinates for bounding box, the classification of the box and its associated confidence are extracted from the tensor. If the confidence exceeds a predetermined threshold, the output image, along with the derived information, is transmitted back to the TB3.

```
results =
model.predict(cv_image_input)
result = results[0]
box = result.boxes[0]
cords = box.xyxy[0].tolist()
sign_ID = box.cls[0].item()
confidence = box.conf[0].item()
xmin,ymin,xmax,ymax =
int(cords[0]), int(cords[1]),
int(cords[2]), int(cords[3])
```

5.1.2 Implementing ORB

ORB is a better form of SIFT algorithm, it almost works in similar pattern as SIFT [14]. 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 [15]. It uses less computational time than YOLO as feature extraction of all reference images and initialization of ORB and BFMatcher was done just once before input image is subscribed. Once, one reference image for each sign is taken, "OpenCV2" [19] library is imported in the node because it is used as our base library for machine learning feature extraction. First, all the reference images are loaded in the node and afterwards, key points are fetched from each reference image with the help of ORB.

```
self.orb =
cv2.ORB_create(nfeatures=2000)
self.img2 = cv2.imread(dir_path +
'stop.png',0)
self.kp2, self.des2 =
self.orb.detectAndCompute(self.img
2,None)
```

Once, the key points are fetched from the reference images, the input image subscribed from the TB3 is processed. Therefore, the feature matcher is initiated, in this case "BFMatcher" is used to compare key points of reference images to current image. If a sign is detected a bounding box is drawn on the input image along with which sign it is. Finally, the output image with the bounding box on it is published.

```
self.bf = cv2.BFMatcher(
kp1, des1 =
self.orb.detectAndCompute(cv_image
_input,None)
matches2 =
self.flann.knnMatch(des1,self.des2
,k=2)
```

6 EVALUATION OF PERFORMANCE

Test-cases were formed in a way that all criteria are given for judging efficiency of the system regarding defined task of TSR for autonomous driving on Mini-City track. Within following work each traffic sign (as shown in Figure 2) together with Accuracy and Precision criteria gets a specific abbreviation:

P – Parking;	T – Tunnel;	S – Stop;
I – Intersection;	C – Construction;	50 – Speed 50;
100 – Speed 100;	Li – Traffic Light;	L – Left;
R – Right;	A% – Accuracy;	P% - Precision.

Three test-cases have been selected which focus on both, accuracy and computational time required. All needed changes have been embedded in the autorace-package code. "On-Table test" was conducted on ORB, SIFT, YOLO and YOLO with SIFT. Further, the one with the best results was taken in consideration for "On-Track test" to get to a final decision.

Table 1: Test-Cases.

Test	Description
On-Table: Accuracy	To test accuracy of TB3 through TP, TN, FP, and FN 1) TB3 is placed on a table, so environment/ background independency is tested. 2) TB3 is not moving but the sign shown to robot is moved to different positions like left, right, up, and down.
On-Track: Accuracy	To test accuracy of TB3 through TP, TN, FP, and FN 1) TB3 is placed on the Mini-City track and it performs sign detection along with lane detection, by that motion test is done. 2) Due to moving, signs can be detected from long and short distances. Range is defined as long range for above 12cm and short range for less than 12 cm.
Processing: Time	This test is done during On-Table and On-Track testing, by determining computational time in detect_sign node

6.1 On-Table Results

This test was mainly focused on accuracy in limited environmental conditions as name suggests. This test is conducted on YOLO, ORB and SIFT and the one with the best accuracy will be considered for further testing.

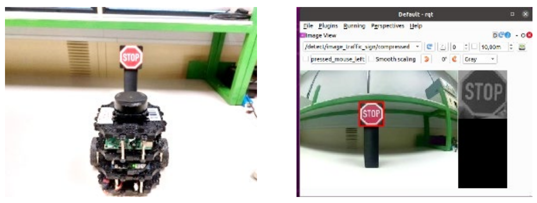


Figure 13: TurtleBot3 during On-Table test.

Figure 13 represents the setup of On-Table test, in which the TB3 is placed on the table in the laboratory and one after the other sign is placed in front of it as shown in the image on the left. Figure 13 right side shows in a ROS rqt-view what TB3 recognizes while On-Table test.

6.1.1 YOLO

Results from Table 2 clearly state that YOLO is having 100% accuracy for five signs out of ten, and 90% or above accuracy for remaining signs except traffic sign "Left" and "Right."

Table 2: On-Table test YOLO.

	P	T	S	I	C	50	100	Li	L	R
TP	10	10	10	10	9	10	9	9	7	5
FP	0	0	0	0	0	0	0	0	0	0
TN	2	1	1	3	2	2	1	1	3	2
FN	0	0	0	0	1	0	1	1	3	5
A%	100	100	100	100	92	100	91	91	77	58
P%	100	100	100	100	100	10	100	100	100	100

6.1.2 ORB

It is clear from the results shown in Table 3 that ORB is having less than 40% accuracy for all signs except "Left" sign and it cannot be considered as a sign detection algorithm in the final system.

Table 3: On-Table test ORB.

	P	T	S	I	C	50	100	Li	L	R
TP	2	1	3	0	0	2	2	0	8	2
FP	0	0	0	0	0	0	0	0	4	0
TN	2	1	1	2	2	2	1	1	0	2
FN	8	9	7	10	10	8	8	10	2	8
A%	33	18	36	16	16	33	27	9	57	33
P%	100	100	100	100	100	100	100	100	66	100

6.1.3 SIFT

As shown in Figure 14 ORB as well as SIFT uses reference images.

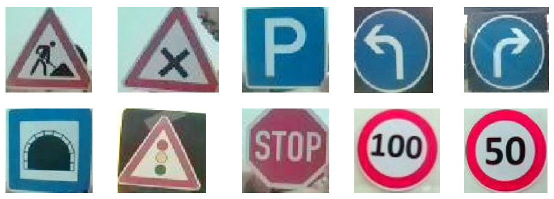


Figure 14: Reference images used for SIFT.

SIFT detects key points in each reference image and further matches those with the current input image's key points. It is observed in Table 4 that SIFT is having better accuracy than ORB in all signs and in case of "Right" it is having better accuracy than YOLO and for "Left" equivalent to YOLO.

Table 4: On-Table test SIFT.

	P	T	S	I	C	50	100	Li	L	R
TP	1	7	10	8	9	10	8	5	8	10
FP	0	0	0	0	0	0	0	0	2	0
TN	1	1	2	2	2	2	1	5	3	2
FN	9	3	0	2	1	0	2	5	2	0
A%	18	73	100	83	91	100	82	66	73	100
P%	100	100	100	100	100	100	100	100	80	100

However, Figure 15 left side shows also clearly, that partly not all the key points are set to the sign itself, in the shown case one was set to the bottom of the stand.

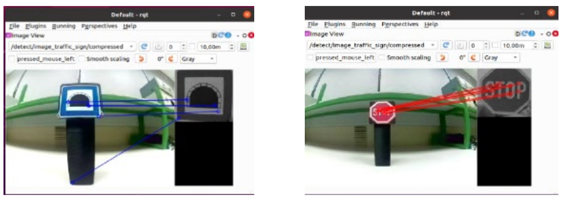


Figure 15: Results of SIFT.

6.2 Optimized Approach

Based on all the gathered experience above and in combination of all techniques suggested, a final way of traffic sign recognition system was designed, which is using YOLO as the primary algorithm for detection of all signs and SIFT as secondary instance which is used only for verification of “Left” and “Right” traffic signs. This way the weakness of YOLO for these two signs shall be vanished. Due to that ORB is not accurate for our use case at all as well as it shows strong environment dependency, it was not further considered. Figure 16 shows the implementation of optimized TSR system. First, the TB3 will start its camera for publishing its video stream. Then, TSR will read frames until and unless system is stopped using “CTRL+C” from terminal. The TSR system starts to process these frames The object detection done by YOLO is detecting trained signs, if traffic sign is detected it checks if identified object is “Left/Right” sign or any other sign, else it will publish no sign detected. For the case of “Left/Right” sign it will cross verify it with the help of SIFT, else it will directly publish the detected sign without using this instance. If SIFT also confirms the same direction like YOLO, the system publishes that sign, else system will publish no sign detected.

6.2.1 Implementing YOLO with SIFT

Quickly, both libraries “Ultralytics” and “OpenCV2”, supporting YOLO and SIFT, have been added to the node. The pre-trained model is loaded for YOLO and SIFT is also initiated. Afterwards, for SIFT “Left” and “Right” sign’s reference images are loaded and key points are fetched from those two images. FLANN matcher is initiated and the input image is handled by YOLO model before it is optionally forwarded to the SIFT for cross verification. If it is any sign other than direction signs it will be published directly after YOLO detection is finished.

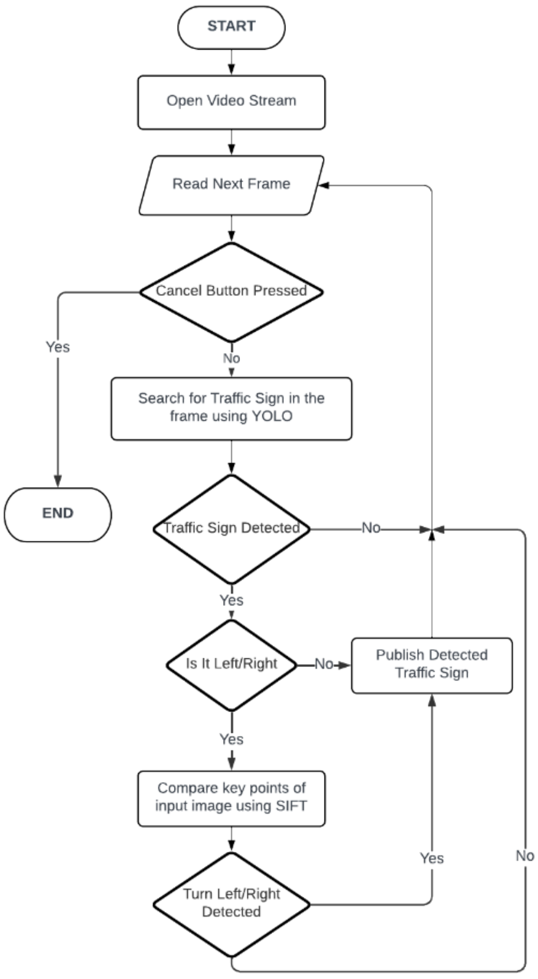


Figure 16: Processing flow of Optimized-TSR based on YOLO combined with SIFT.

6.2.2 On-Table Results YOLO with SIFT

Results from Table 5 clearly state that YOLO with SIFT is having 100% accuracy for five out of ten signs, while for others it is higher than 80%.

Table 5: On-Table test YOLO+SIFT.

	P	T	S	I	C	50	100	Li	L	R
TP	10	10	10	10	9	10	9	9	8	8
FP	0	0	0	0	0	0	0	0	0	0
TN	2	1	1	3	2	2	1	1	3	2
FN	0	0	0	0	1	0	1	1	2	2
A%	100	100	100	100	92	100	91	91	85	83
P%	100	100	100	100	100	100	100	100	100	100

Hence, it can be said that it is best result reached for On-Table test and that is why it is taken for continuing with On-Track test.

6.3 On Track Results

The On-Track test exclusively evaluates the YOLO with SIFT implementation due to that it has demonstrated the best results among all algorithms in the On-Table test. The TB3 is positioned on the Mini-City track during this test to judge dynamic positioning and range dependencies, as well as the computational time when capturing images during motion. The main result of this test is to determine if the implementation is feasible to handle the given real-time task where it has to follow the lane with the aid of a lane detection node while detecting the signs at fixed positions. Obviously, detection must be finished in front of a sign so that other nodes can be called and processed in time. Earlier detection results in more time for processing other tasks during autonomous driving. However, if signs are detected to early, functions might be called that are not yet relevant. Figure 17 illustrates the setup of the On-Track test, where the TB3 moves and upon approaching signs, it detects and notifies the system.

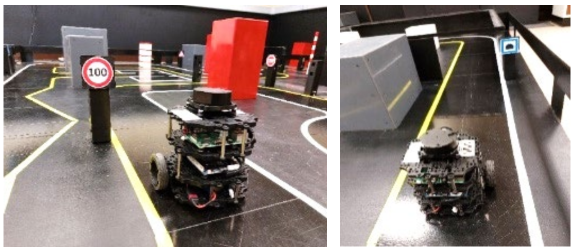


Figure 17: TurtleBot3 during On-Track test.

From Table 6 it is observed that YOLO with SIFT is having 100% accuracy for eight out of ten signs and for remaining two signs it is having above 90% accuracy. If compared to Table 5 "On-Table test YOLO+SIFT" shows that YOLO with SIFT shows better result in artificial-world conditions of Mini-City.

Table 6: On-Track test YOLO+SIFT.

	P	T	S	I	C	50	100	Li	L	R
TP	10	10	10	10	9	10	10	10	9	10
FP	0	0	0	0	0	0	0	0	0	0
TN	2	1	1	3	1	2	1	1	2	3
FN	0	0	0	0	1	0	0	0	1	0
A%	100	100	100	100	91	100	100	100	92	100
P%	100	100	100	100	100	100	100	100	100	100

6.4 Analysis of Algorithm Performance

As explained in Figure 8 "Computation Time Test On-Track/Table" in Chapter 4 "Experimental Setup" within "detect_sign" node the computational time is recorded.

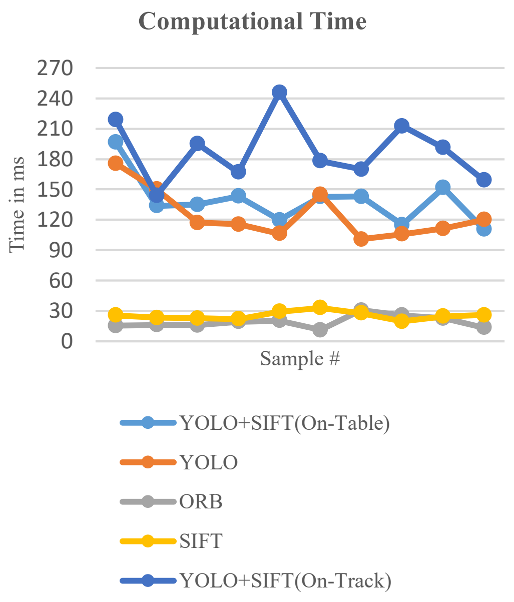
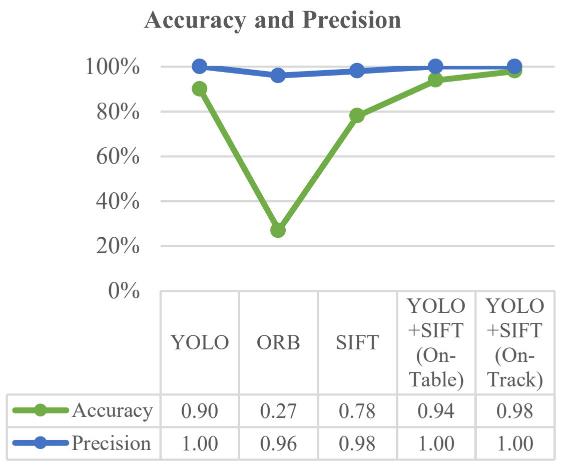


Figure 18: Computational Time Test samples.

At Figure 18, it is clearly visible that YOLO+SIFT is the most time-consuming method for detecting signs and publishing the result. The average time taken for YOLO+SIFT (combining data from both On-Table and On-Track tests) is 163.7ms with a standard deviation of 36.94, highlighting the different effects of using two algorithms in sign detection. YOLO+SIFT takes longer for sign recognition, with an average time of 124.82ms and a standard deviation of 24.20. ORB and SIFT take nearly the same amount of time for traffic sign detection. ORB is a bit quicker with an average time of 19.15ms compared to SIFT's average time of 25.37ms. However, in terms of stability, SIFT is superior with 3.95, whereas for ORB the standard deviation is 5.93. Therefore, it can be concluded that ORB is the fastest algorithm among them, while SIFT yields more consistent outcomes.
In addition, the accuracy and precision of On-Table and On-Track test for different algorithms for all signs was also measured with the help of formula shown in Figure 5 "Evaluation Criteria".

Average accuracy and precision in Figure 19 demonstrates that the decision to use YOLO for traffic sign detection was justified, as it improved TB3's traffic sign recognition accuracy by 12% to

90% compared to the default SIFT algorithm. Although the ORB algorithm was also implemented, where Figure 17 demonstrates that ORB has the shortest computational time at 19.15ms but with a low performing average accuracy of only 27% and a 4% loss in precision too. Table 2 reveals a lack of accurate detection for traffic signs such as "Left" and "Right" for YOLO.



	YOLO	ORB	SIFT	YOLO +SIFT (On-Table)	YOLO +SIFT (On-Track)
Accuracy	0.90	0.27	0.78	0.94	0.98
Precision	1.00	0.96	0.98	1.00	1.00

Figure 19: Comparison of average accuracy and precision for each algorithm.

The combination of YOLO +SIFT achieved the highest accuracy of 98% on the track, again with perfect precision for traffic sign detection with the effort of a long computational time of 124.82ms. The optimized integration of YOLO with SIFT resulted in 94% accuracy for table and 98% accuracy for track. Whereas most probably the fixed position in comparison to table tests pushed the average 4% up. This approach successfully detected all signs both on and off track with a cost of computational time, which can be managed by reducing frames per second.

7 CONCLUSIONS

The main objective of this paper is to develop an efficient, easy to train and understandable system for traffic sign recognition in educational use. The system should be robust, with minimal errors and operate in real-time. To improve efficiency, several algorithms such as R-CNN, YOLO, ORB, SURF, and SIFT were considered. The advantages and disadvantages of all algorithms, including SIFT, were analyzed through theoretical examination. YOLO, ORB and SIFT were selected for testing on TB3 within an artificial environment, the Mini-City. However, selecting the appropriate algorithm alone does not suffice to boost efficiency; various techniques were also utilized. For instance, a detection frame and a fusion of two algorithms were employed. While these measures enhanced accuracy, they also impacted computational time, necessitating the implementation of numerous test cases.

Two tests - the "On-Table Test," which focused on environment independence, floating traffic signs, and computational power criteria, and the "On-Track Test," which evaluated accuracy based on range and dynamic positioning - were conducted and efficiency was additionally tested via computational time. Test cases have demonstrated the effectiveness of the systems and identified the most suitable technique.

- YOLO exhibited superior accuracy and perfect precision compared to ORB and SIFT in various environments, while the last two were only reliable if tested in the same environmental setting as their reference image. In contrast, YOLO can effectively operate environment independent with any background.
- ORB requires the least computational time, whereas YOLO demands extensive resources to function optimally. This is due to YOLO performing recognition tasks after image access, whereas ORB merely matches key points once the input image is retrieved.
- YOLO itself is not entirely accurate for all signs; therefore, SIFT is introduced in combination with YOLO, but solely on signs that are not detected accurately, rather than all images.

One of the primary issues encountered was the high computational power demand of YOLO, leading to latency in the system and preventing it from being real-time. However, YOLO performed complete detection while intrinsic and extrinsic camera threads were working in the background. Simplest way to cover is to decrease the frames per second within autorace-package for TB3.

```
if self.counter % 10 != 0:
        self.counter += 1
        return
else:
        self.counter = 1
```

After considering the findings and analyzing the computational time of various algorithms and techniques, it was decided that the optimal approach would be to utilize YOLO and SIFT. Of course, this approach might not be needed for a simple TSR exploration within educational use, however, it demonstrates a usual way of implementing assistant systems where for example in addition to a camera-based object detection digital-map data is used to validate detected objects or situations. As an alternative the YOLO training process could be made more complex to reach better accuracy for Left/ Right sign.

REFERENCES

[1] "ROBOTIS e-Manual," Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/autonomous_driving/#traffic-sign-detection, [Accessed Sep. 7, 2023].

[2] R. Amsters and P. Slaets, "Turtlebot 3 as a Robotics Education Platform," in Robotics in Education (Advances in Intelligent Systems and Computing), M. Merdan, W. Lepuschitz, G. Koppensteiner, R. Balogh, and D. Obdržálek, Eds., Cham: Springer International Publishing, 2020, pp. 170-181.

[3] B. Zhong and Y. Li. "Image Feature Point Matching Based on Improved SIFT Algorithm," [Accessed Aug. 22, 2023].

[4] V. Vijayan and P. Kp. "FLANN Based Matching with SIFT Descriptors for Drowsy Features Extraction," [Accessed Aug. 22, 2023].

[5] M. A. A. Babiker, M. A. O. Elawad, and A. H. M. Ahmed, "Convolutional Neural Network for a Self-Driving Car in a Virtual Environment," 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), Khartoum, Sudan, 2019, pp. 1-6, doi: 10.1109/ICCCEEE46830.2019.9070826.

[6] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," Apr. 2018, [Online]. Available: https://arxiv.org/pdf/1804.02767.

[7] 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.

[8] 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.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, Jun. 2014 - Jun. 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81.

[10] H. Yanagisawa, T. Yamashita, and H. Watanabe, "A study on object detection method from manga images using CNN," in 2018 International Workshop on Advanced Image Technology (IWAIT), 2018, pp. 1-4, doi: 10.1109/IWAIT.2018.8369633.

[11] O. Hmidani and E. M. Ismaili Alaoui, "A comprehensive survey of the R-CNN family for object detection," 2022 5th International Conference on Advanced Communication Technologies and Networking (CommNet), Marrakech, Morocco, 2022, pp. 1-6, doi: 10.1109/CommNet56067.2022.9993862.

[12] J. Du, "Understanding of Object Detection Based on CNN Family and YOLO," J. Phys.: Conf. Ser., vol. 1004, no. 1, p. 12029, 2018, doi: 10.1088/1742-6596/1004/1/012029.

[13] 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.

[14] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. "ORB: An efficient alternative to SIFT or SURF," [Accessed Aug. 22, 2023].

[15] F. K. Noble, "Comparison of OpenCV's feature detectors and feature matchers," in The proceedings of 23rd International Conference on Mechatronics and Machine Vision in Practice: M2VIP 2016 : Nov. 28-30, 2016, Nanjing, Jiangsu, China, Nanjing, China, J. Potgieter, P. Xu, Z.-S. Zhang, X.-S. Wang, H. Yi, and I. C. o. M. a. M. V. i. Practice, Eds., 2016, pp. 1-6, doi: 10.1109/M2VIP.2016.7827292.

[16] Sh. Nimmisha, "Classification of stages of Diabetic Retinopathy using Deep Learning," 2020, doi: 10.13140/RG.2.2.10503.62883.

[17] "Quickstart - Ultralytics YOLOv8 Docs," [Online]. Available: https://docs.ultralytics.com/quickstart/#use-ultralytics-with-cli, [Accessed Sep. 12, 2023].

[18] S. Ola, Th. Bjørsum-Meyer, A. Histace, G. Baatrup, and A. Koulaouzidis, "Annotation Tools in Gastrointestinal Polyp Annotation" Diagnostics 12, no. 10: 2324, 2022, [Online]. Available: https://doi.org/10.3390/ diagnostics12102324

[19] M. Shoeb, M. Akram Ali, M. Shadeel, and M. Abdul Bari, "Self-Driving Car: Using Opencv2 and Machine Learning," The International journal of analytical and experimental modal analysis (IJAEMA), ISSN 0886-9367.

[20] G. Rossum and F.L. Drake, "Python 3 Reference Manual", Scotts Valley, CA: CreateSpace, 2009.