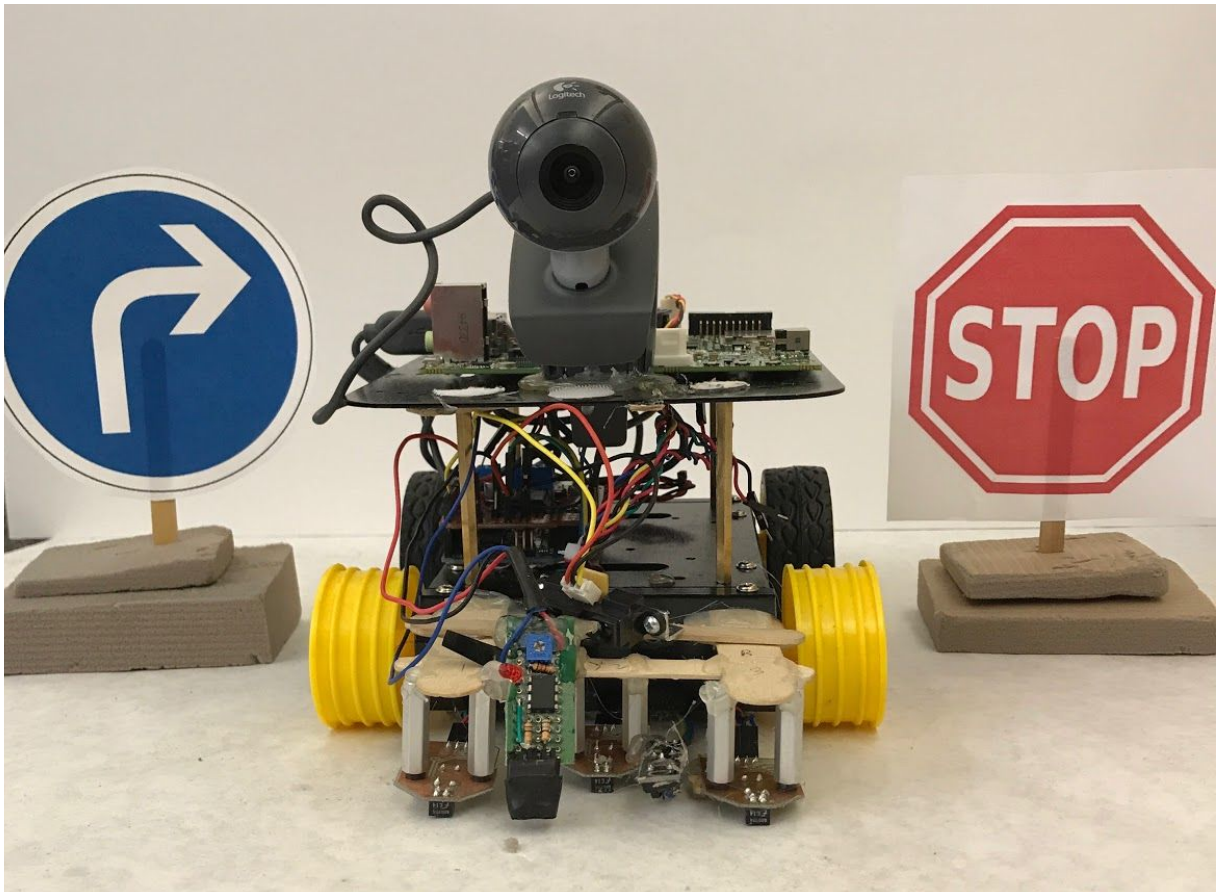
ECEN 5623 - Real Time Embedded Systems

R.E.S.L.A - Real Time TESLA

Autonomous Traffic Sign Detection Bot



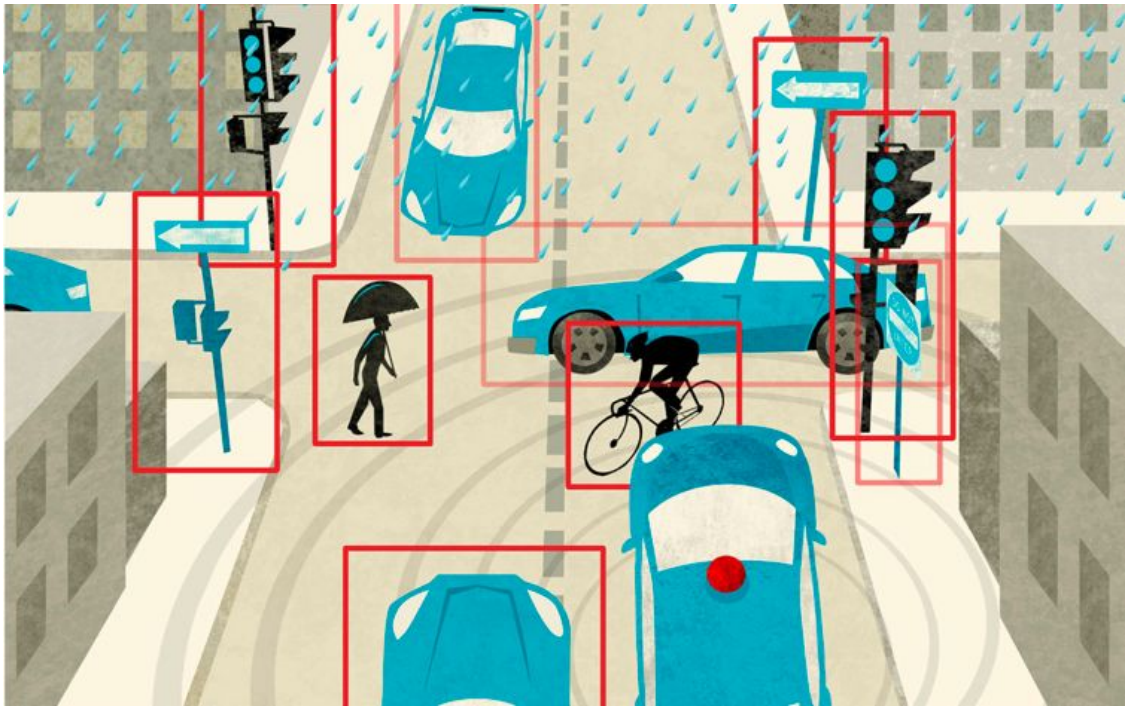
Report by:

Rishabh Berlia
Praveen Gnanasekaran

Contents

Introduction

This is the project for the course ECEN 5623 Real Time Embedded Systems. The aim is to develop a real time system that can run multiple services which run within certain deadlines and verifies whether all of them are feasible and schedulable within the deadlines decided for them. The autonomous car project was undertaken to explore and experiment with critical uses of soft real time systems in life critical systems.



In this age of growing advancements in artificial intelligence, we realize there is a growth in the area for development of autonomous vehicles, which can replace human ability to take decisions in a real time environment.

Our project aims to use the concepts of Real Time Systems and Computer Vision learned in this class and apply it to a real world scenario with the help of Machine Learning.

Functional Requirements

For any system to be truly autonomous, it had to possess some kind of intelligence at least in the least level so it could predict and decide which task to do. This cannot be achieved by using a simple detection method like the hough-elliptical detection or color detection. Instead, we used **machine learning** to classify traffic signs.

The autonomous car detects stop signs in real time and is able to control the navigation and move through waypoints from source to destination without human intervention. This is done by detecting traffic signs on the road using a supervised machine learning algorithm which predicts the stop sign and classifies it based on a trained Sum Vector Machine model (SVM). The system has to be soft real time since the detection should happen within a deadline with an added margin- failure to do so would result in catastrophic consequences such as a crash/collision.

The efficiency of the system is dependent on the machine learning and object/ obstacle detection part. Reliable motor control should also be taken into account since this is the end output which controls the bot. The real time control of motors and wheels and receiving interrupts from the ultrasonic sensor is also critical to this system's success. The camera will capture the image, extract features and try to classify the image to some level of accuracy (if it is a stop sign, left/ right turn or just the lane without any sign) Based on this decision, the NVidia Jetson will control the motors by generating PWM to move the bot forward, left, right or bring it to a stop.

This also has to be in real time as not just the detection of the object is important but the actuation control of the bot is the actual end output result. Because the camera's field of coverage is limited and it cannot detect objects which are outside its field of vision or because an obstacle goes undetected by the camera, an ultrasonic sensor is added to aid in the detection of any obstacle. The ultrasonic sensor has to periodically excited and the distance from this sensor has to be constantly updated.

This exciting and measuring has to be in a task within real time requirements as well. Based on the distance measured, the bot will slow down or complete stop depending on the distance of the object detected. To summarize, the functional requirements are, to be able to capture images, extract features and get a classification for the image based on an existing trained model. Use this information to control the bot in real time to be able to turn right, go forward or come to a complete stop based on the image in the frame and if an obstacle is detected.

The following are the **capability requirements** of our system:

1. The system shall operate with input from the camera, distance sensor and actuate the motors. This system (referred to as a bot) should act autonomously with these inputs.
2. The images captured by the camera shall be processed by the Jetson TK1 and information(features) shall be extracted from captured frames. This should happen within a deadline so that the bot stays constantly updated on its position.
3. Real time motor control response based on the processed image classification.
4. The system should try to be on track and not colliding into obstacles.
5. The system must be able to navigate autonomously according to traffic signs.

Real Time Requirements

Before our system can start, we need to run a one-time computation called for SVM Training, which uses an existing dataset of classified images and extracted HOG (Histogram of oriented gradients) features and train the feature on an SVM (Sum Vector Machine) which generates the model. This trained model is used by the Image Classifier Service to determine the Traffic Sign detected. The real time system built for this project has a total of four real time services, including the following:

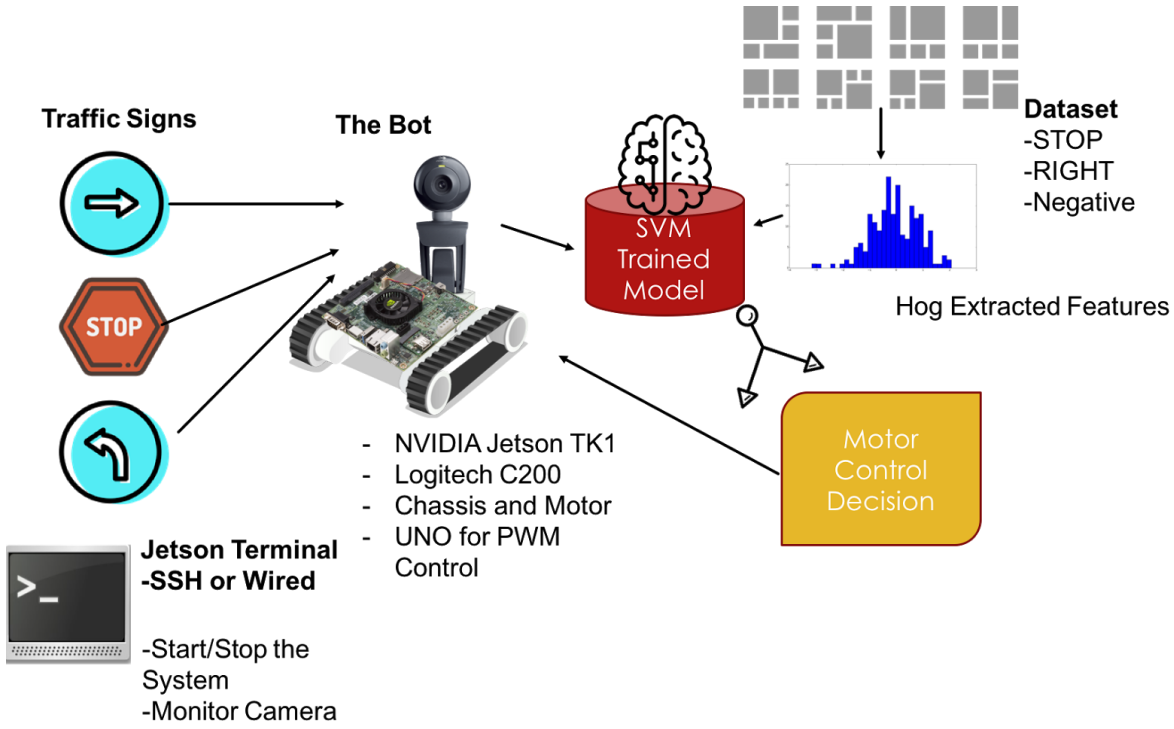
1. **Image Feature Extraction Service :** This service captures any new frame from the camera, extract the HOG features and saves it.
2. **Image Classifier Serivce :** The Image Classifier service reads the extracted features and tries to find a match with the existing model, to predict the class in which the frame would belong.
3. **Drive System Control Service :** This service actuates the bot according the predicted response. This services uses UART to communicate (bi-directionally) to an Arduino UNO which controls the motors. It send out a message indicating the direction of drive, and receives an acknowledgment from the system.
4. **Obstacle/Crash Avoidance Service :** This service is used get the data from the Ultrasonic Sensor, which detects if there is a obstacle in front of the bot and also provide the distance from it.

Each of these real time services were critical for the correct functioning of the entire system as failure of any one of them would result in a complete loss of the system which can lead to a disaster. The deadlines allotted to these services included margins for the frame jitter from the camera, variation in the classification based on images with different colors and

lighting conditions and the sensor readings from the ultrasonic sensor and delay in transmission times through USB from Arduino to the Jetson and vice-versa.

The feature extractor service requires a computation time of 7ms when using a camera with **30 fps**. This service extracts the HOG descriptor features for any image captured by the camera. The WCET for this service comes around 10ms. The image classifier service utilizes around 60ms, the obstacle avoidance service takes 25ms and the motor actuation service requires 15ms for completion. It was necessary that the image classifier service ran only after the feature extractor service since there weren't any lost frames in the process. The obstacle avoidance service determined on the ultrasonic sensors value if any obstacle was detected and only if there were no obstacles, the motor actuation would happen.

Functional Design Overview:



As it can be seen above, the functional analysis of our system includes the camera, and motor control on the bots. The entire system is run on the NVidia Jetson on a Linux platform OS. The components that were used in the entire project includes:

1. An Nvidia Jetson running Linux Ubuntu
2. A Logitech camera
3. 2 Geared Brushed DC Motors
4. Stop sign and Right turn sign
5. Image dataset to train the SVM algorithm
6. A moveable bot with a chassis and motors attached
7. Arduomoto Dual H Bridge Motor Shield (Drive System)
8. Arduino to generate PWM to drive the motors and measure ultrasonic sensor
9. An environment resembling a road with signs that the bot can detect
10. Battery packs to drive the motor and to power the Jetson

The Diagrams including - System Flow Diagram, Sequence Flow, Hardware and Software Block Diagrams and State Diagram are included below.

Real Time Analysis and Design

The real time service sets were first run and their execution times were observed and noted. This was done numerous number of times to get a worst case execution time figure.

Drawing from the references of studies and research in the real world autonomous driving system, we determined and scaled our system down while keeping relatives deadlines due to the reduced complexities in our autonomous environment.
Capacity for 'Image based' services were found out by running the code numerous number of times and capturing different types images while changing the lighting and environment conditions.
The Capacity for Obstacle/Crash Avoidance Service was calculated over a broad of distances for different types of sizes of objects. This WCET was found to be 25 ms.
For the Drive system service. The time is calculated by giving the directional command multiple times, and receiving an acknowledgement after the drive system rotates for a fixed amount of time. WCET for this service was found to be 15ms

Once the WCET analysis was completed, the services were scheduled according to our application. The services aren't independent but are dependent on inputs from other services.

Ci and WCET specification

Service Name	**Worst Case Execution Time (ms)**
S1: Image Feature Extraction Service	**C1:** 10 ms
S2: Image Classifier Serivce	**C2:** 60 ms
S3: Obstacle/Crash Avoidance Service	**C3:** 25 ms
S4: Drive System Control Service	**C4:** 15 ms

	T1	100	C1	10	U1	0.1	LCM =	600					
	T2	120	C2	60	U2	0.5							
	T3	150	C3	25	U3	0.1666667							
	T4	300	C4	15	U4	0.05	Utot =	0.8166667					
RM Schedule	10	20	30	40	50	60	70	80	90	95	100	110	120
S1													
S2													
S3													
S4													

Rate Monotonic Scheduling

The above image shows the RM Schedule for our system, we can calculate the system utilisation.

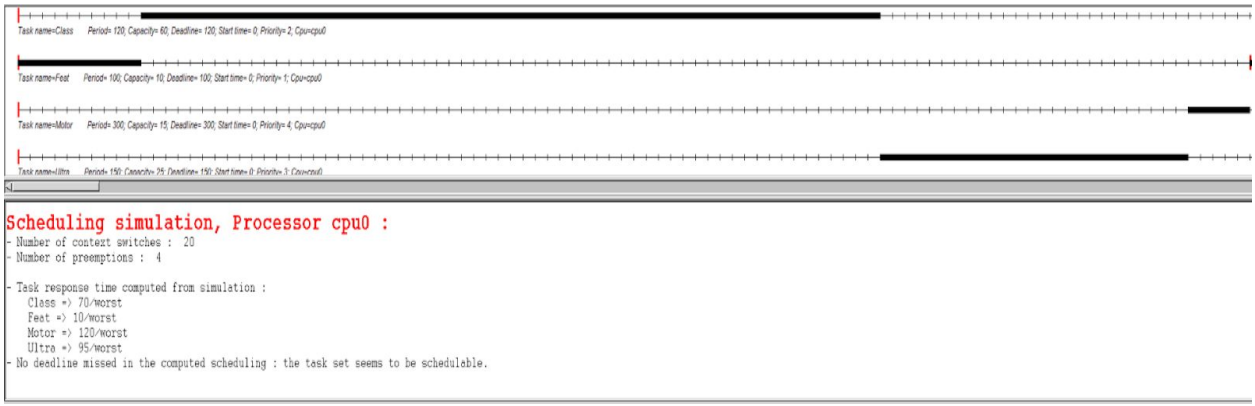
$$U = \sum_{i=1}^{m} Ci/Ti \leq m\,(2^{1/m} - 1)$$

Therefore, calculating utilization for our system.

C1/T1 + C2/T2 + C3/T3 + C4/T4 = 0.1 + 0.5 + 0.167 + 0.05 = .816

For RM LUB, the utilization U is m(2^(1/m)-1) where m is the no. of services. i.e. **4(2^(1/4)-1) = .74**

We can see our scheduling does not satisfy RM LUB, but it doesn't necessarily mean it's not schedulable.



Cheddar Analysis for the Service Set

The utility of the service set was found out to be 0.82 or 82%. The RM LUB for 4 service sets is 74%. Clearly, the service set's utility does not pass the RM LUB test.
To check the feasibility of the service set, the feasibility analysis was done by drawing timing diagrams by hand. Cheddar analysis was also done to verify the hand drawn timing diagrams. The service set was found to be feasible and implemented.

```
root@praveen-Latitude-E5430-non-vPro:~/Downloads/Berlia_Gnanasekaran_Exercise2/Question4/OriginalFeasibilityTest_All# ./feasibility_tests
******** Completion Test Feasibility Example
Ex-0 U=0.82 (C1=10, C2=60, C3=25, C4=15 T1=100, T2=120, T3=150, T4=300; T=D): FEASIBLE


******** Scheduling Point Feasibility Example
Ex-0 U=0.82 (C1=10, C2=60, C3=25, C4=15 T1=100, T2=120, T3=150, T4=300; T=D): FEASIBLE
root@praveen-Latitude-E5430-non-vPro:~/Downloads/Berlia_Gnanasekaran_Exercise2/Question4/OriginalFeasibilityTest_All#
```

Completion and Scheduling Point Tests

We also carried out Completion and Scheduling point tests for our system for a sanity check. They also prove that this set of services would be feasible for RM Scheduling.

The request frequencies for the services were calculated for each service and the average request frequency and release times were observed

Feature Extractor Service:

Request frequency (Hz)	Release time (ms)
25.703	38.905
24.991	40.014
25.349	39.449
25.590	39.077
25. 163	39.74

Image Classifier Service:

Request frequency (Hz)	Release time (ms)
25.703	47.805
24.991	46.429
25.349	45.612
25.590	47.487
25. 163	46.461

Obstacle Avoidance Service:

Request frequency (Hz)	Release time (ms)
17.412	57.431
17.538	57.019
17.724	56.42
17.568	56.921
17.343	57.66

Drive System Control Service :

Request frequency (Hz)	Release time (ms)
20.703	48.302
19.991	50.022
20.349	49.142
20.459	48.878
20.233	49.424

From the above data, the average request frequency and release times for the services were calculated as shown below

Service Name	Request Frequency (Hz)	Release Time (ms)
Feature Extractor	25	40
Image Classifier	21	47.61
Read ultrasonic	17	58.82
Motor control	20	50

During our initial analysis, we had calculated our service deadlines to be 100ms, 120ms, 150ms and 400ms. This gave us enough margin to meet all of our deadlines.

Technical Description

The scope of our project is to do the following things:

- Gather a data set of Positive Images (Actual Traffic Signs) and Negative Images (Things other than the Object of Interest).
- Extract HOG features of the dataset and then train a SVM (supervised learning) model on it to apply classification.
- Classify between at least 2 different SIGNS (STOP, RIGHT) and then actuate our bot accordingly within the deadlines specified.

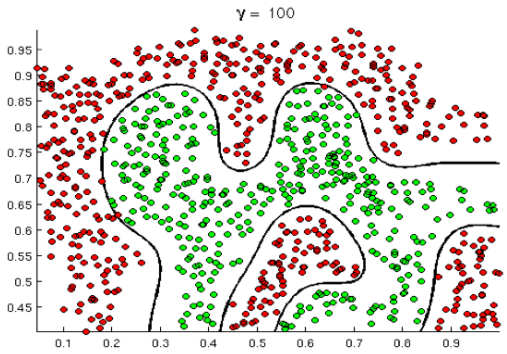
Use a sensor input (Distance Sensor) to detect nearby objects and stop if the distance is too close.

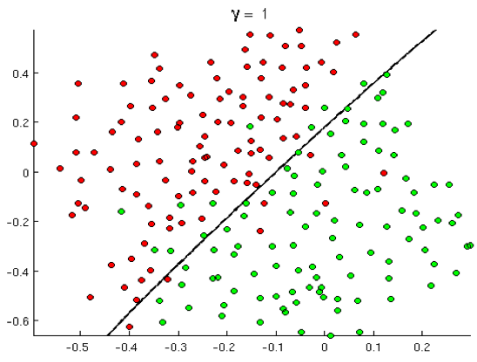
Supervised Learning

- It is a Machine Learning task of inferring a function from a training data and makes prediction based on it.
- Training Data Set : Input Data and Response Values
- This approach seeks to build a model that can make predictions of the response values on a test data set with maximum accuracy.
- Classification: for categorical response values, where the data can be separated into specific “classes”

SVM Support Vector Machine

- Type of a common Supervised Learning Algorithm
- Uses concept of separating hyperplane, to classify the difference between data sets.
- Uses the concept of mathematical Vectors in this hyper plane.
- It can be used to classify complex hyperplanes easily.





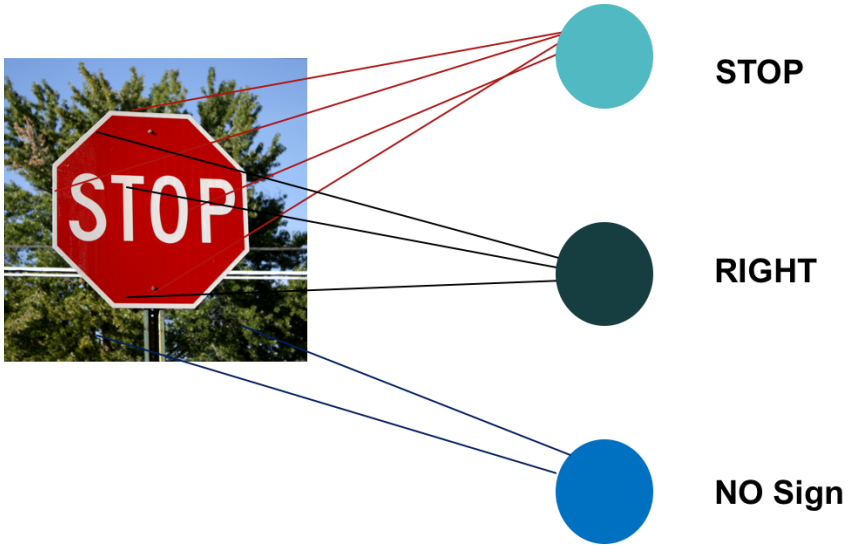
HOG - Histogram of Oriented Gradients

- It is a feature descriptor used in computer vision.
- Main use is for object detection.

The main idea behind it is :

  - The Local Object's appearance and shape can be described by distribution of intensity gradients.
  - The image is divided into cells and then the histogram of gradients is calculated for each cell.
  - The HOG descriptor is the concatenation of these histograms.

The Model



Our Classifier has 3 main classes, that means the output of a given input image can only be classified within these 3 possibilities. The No. of lines indicates how closely it matches the model for the given class. The more the weights for one class, the more is the probability of that class being the classified one.

The Data Set

Our initial training data-set consisted of images from the German Traffic Sign Database of different signs with varying conditions like angle of view, color variations, lighting variations etc.

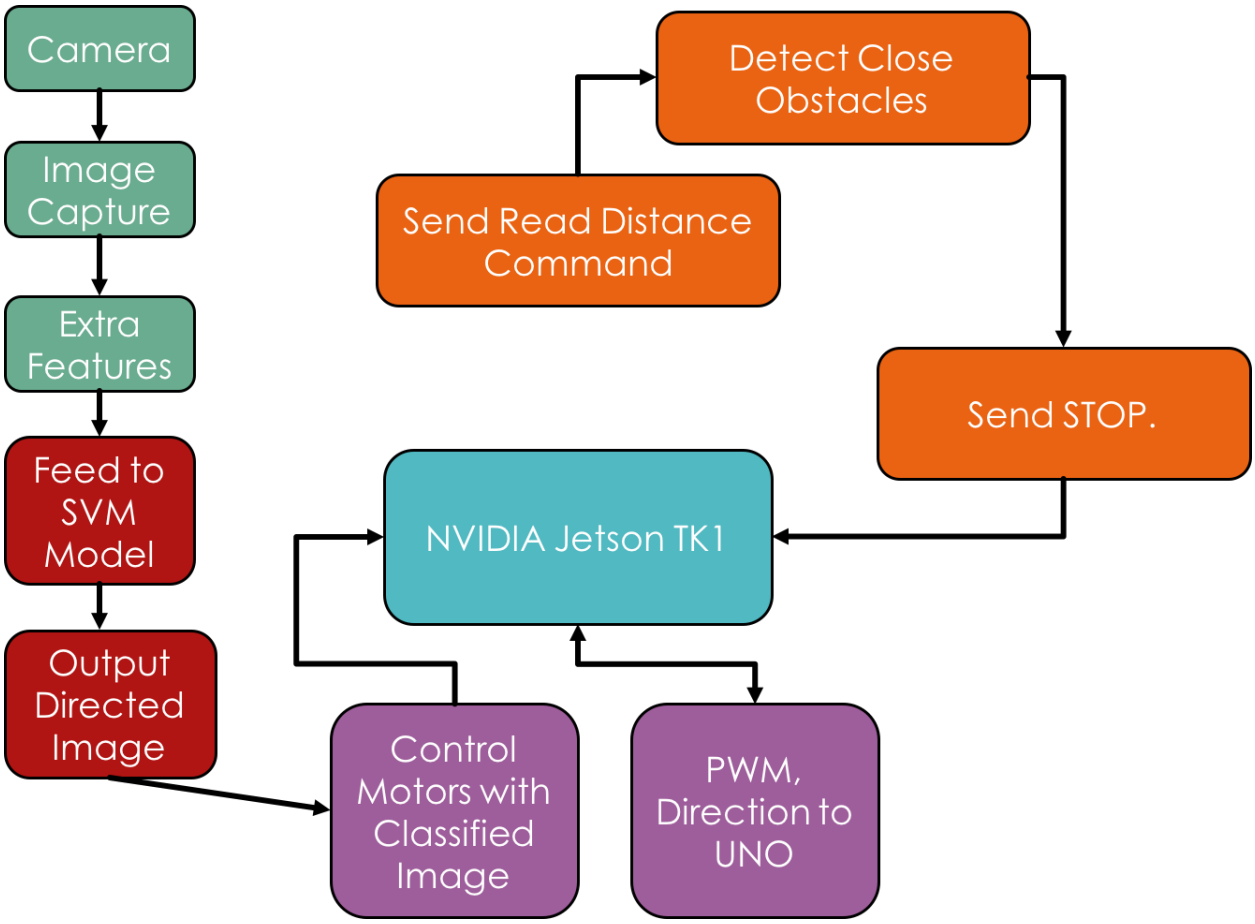
STOP Sign : 750 Images
RIGHT Sign : 690 Images
LEFT Sign : 390 Images
No Negative Images : They are required to detect if no sign is present.

We created a openCV capture program to take images in different angles and lighting conditions for the STOP, RIGHT and Negative Images in the room where we would demo. This gives us more focused data than a general one.

STOP Signs : 300 Images
RIGHT Signs : 300 Images
Negative Images : 390 Images.

This is the dataset we used finally in our project to demo our system working as it was more aligned with our test setup environment.

System Flow Chart



This is the flow of our system. The sequences of processes determines how our system operates and how the output is determined. The HOG features from an image are stored in a vector and classes are assigned to it based on what the image actually is. This process is repeated for all the 990 data sets that we use and the SVM model is trained from these extracted features as input.

Feature Extraction Thread

It all starts with the capture of the image from the camera. This is then pre processed, resized and the Histogram of Oriented Gradients (HOG) features are extracted from it. Some of the descriptors used in the HOG feature extraction include allocating the cell and block size, block stride size, method of normalization on the RGB pixel and the number of orientation gradients for the image. Using all these information, the features are extracted and stored into a vector.

Image Classification Thread

The extracted features are fed into the already trained SVM model and predicted as to what image it is. The output of the SVM model is a class which classifies what image is located in the frame. Some of the parameters for the image classification included the type of SVM kernel that was being used (Nonlinear INTER kernel) and the number of computations the SVM had to undergo before classifying an image. After the image is predicted by this thread, the output is received and the corresponding motion flag between move right or move forward is set.

Obstacle/ Crash Avoidance Service

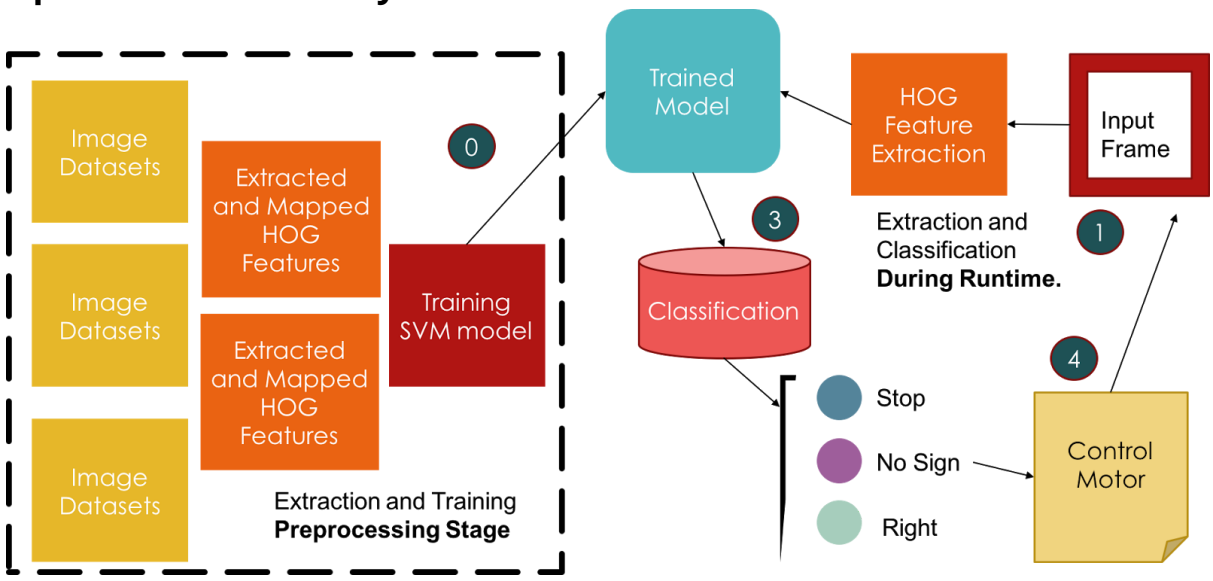
It is necessary to detect any obstacle in the way before sending a motion actuation command so that even if the camera is not able to capture any stop sign in its frame of view, the ultrasonic sensor reassures the system by detecting any obstacle on the road. This is done by reading the value of the distance from the Arduino and deciding if the obstacle is too close to the system. The motion control flag is finally set and is ready to be sent to the motor.

Drive System Control Service

The motion control flag which was determined by all the above services is sent to the Arduino for it to drive the motors. This controls the motor and actuates it to move forward. The motor motion is not a continuous motion as it has to wait for another command to

move it forward again. This is done to have some control over the system so that it doesn't overshoot and crash into any obstacle or not detect a sign properly.

Sequence Flow of the System



The diagram above shows the sequence of operations in our system from start to the actuating of the actual motor, controlling the motion of the bot.

Preprocessing and Training stage

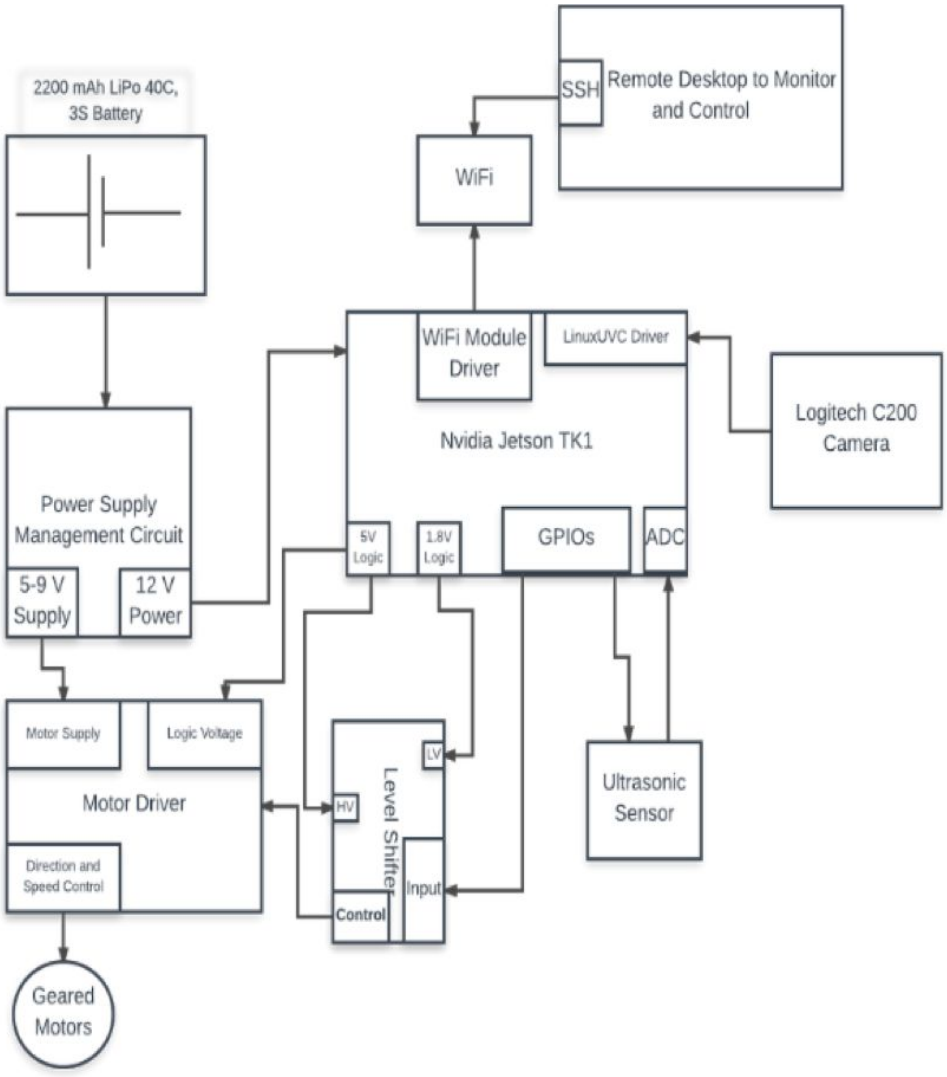
- HOG features from a .ppm image and stored in a text file
- Features are assigned classes and fed into SVM model to be trained
- Trained SVM model is obtained and will be used to classify new images

Extraction and Classification during runtime

- The image is obtained from the camera capture and resized
- HOG features are extracted and predicted using the SVM predict function
- The output is used to determine which flag to set to control the motor

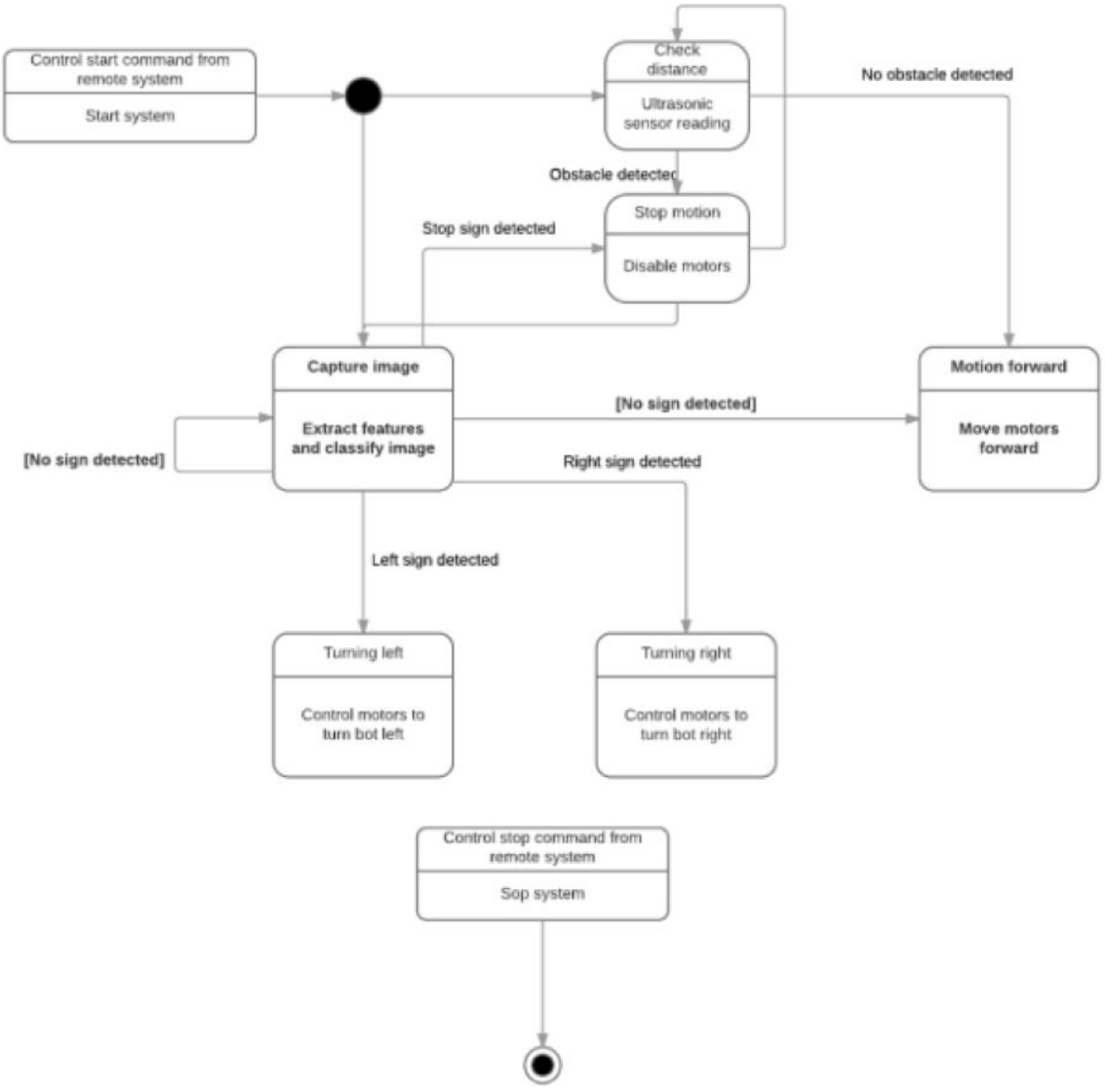
Detailed Hardware and Software Block Diagram

The hardware block diagram shown below describes the various hardware parts used in our system.



State Diagram

The state diagram shows the various states that the system can be in between transition and processing the data. This gives an idea of how the system functions during the thread executions.



Proof-of-Concept and Tests Completed

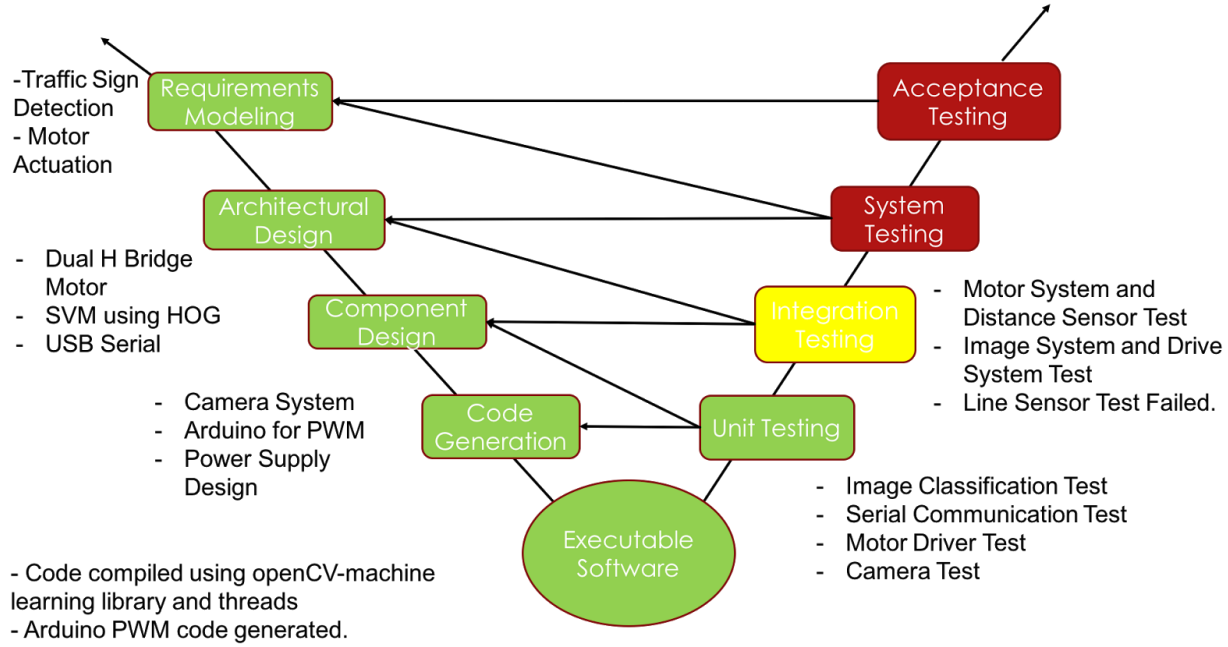
The project had separate working parts that were used together finally to make the entire system work. The testing, verification and integration was done in an incremental approach. The main modules that were working and required regression testing were

- Capture images, extract HOG features and store them reliably in a file on the Jetson (Sample code attached)
- Train the SVM model with the extracted features and store the trained SVM structure. (Sample code attached)
- For the image to be classified, extract the same features and predict and give a classified output.
- Drive / stop the motor on the bot based on the classified output.

Furthermore, the proof of concept was demonstrated during the project demo as the output was obtained as required by the project proposal.

As proof of the output, a link to the video of the project working has been attached in the Supporting Materials.

Verification and Validation



The testing and verification was done in an incremental approach. We first developed code for each of the threads separately to check whether the functionality is working.

Once we integrated all the code together we found that there were a lot of timing issues present, which were explained in the Real-Time analysis and design with timing diagrams section.

Challenges Faced :

Installing Opencv machine learning library for Jetson.

Using machine learning libraries on Opencv was a relatively difficult task as compared to using Matlab or Python to build and train a dataset of features. The reason is C++ does not support easy use of multidimensional vectors for training using a machine learning. Since our system had to be deterministic, we decided not to use any high level language like Python or Matlab and implemented all the algorithms in C itself.

Stable PWM output for Jetson.

We found out during our tests that Jetson has issues with PWM generation. We were not able to get a stable output from Jetson. We needed it to control the speed of our motors. Our initial chassis failed and broke on initial test runs due to high power of our battery and no speed control over it. We realized we would need PWM to control motor speeds by reducing the average voltage on the motors through the EN pin. Also, our motors were rotating at different speeds while applying the same voltage. This was due to the errors in gear ratios for the motor, therefore using PWM to control speed separately was our solution.

Failure of Line sensors
Our, line sensors failed to provide accurate readings. (The paper we placed on carpet was slippery and had air gaps). Analog Based Line Sensors are not very reliable. Their threshold data was varying easily, making detections inaccurate.
So we decided to skip integrating them with the main system.

Model accuracy is limited by the limited dataset.
If we increase the dataset a considerable amount of time would go in training it. Also, it is difficult for us to take over 3000 images in different angles, lighting conditions.

Conclusion

The Autonomous Traffic Sign Detection Bot, was implemented using the knowledge and information gained in Real Time Systems and OpenCV during this course. We also learned basic Machine Learning for classification between the different traffic signs.

The Bot was able to accurately identify and classify STOP Sign, Turn RIGHT Sign and No Sign and act accordingly in real time. The Bot would also detect obstacles in front of it and would preempt sign detection to avoid the obstacles first with the given hardware and cost restrictions. We were able to achieve a reasonable response to actuate the bot with the sign classified.

We learned a lot during the course of the project. We learned how to implement systems with real time requirements and deadlines. We were also able to explore image capture and image processing using OpenCV.

The challenges we faced during the project were lessons that would help us in the course of our academic and professional life.

Acknowledgements

We would like to thank Prof. Timothy Scherr for giving us the opportunity to work on a real time project which helped us improve our knowledge and learn from the challenges faced when designing a real time system. We would also like to thank Mayank Patwari, a graduate student from the University of Munich,TU for giving us valuable suggestions for the machine learning used in our system.

We would also like to thank Prof. Christopher Heckman from the C.S. department here at CU Boulder. He directed us towards a lot of useful information for Machine Learning and OpenCV. We would also express or thanks towards the ITLL Staff including Prof. Derek Reamon, Tim May and Dan Godrick for their valuable resources.

We would also like to thank both the TA's for their support and like to thank our fellow guidance through this project and throughout the semester. We would also classmates and friends for giving us valuable inputs about the project which helped us in improving our project and troubleshooting some of the problems we faced during the project.

References

[1] Machine learning tutorial on Python:
http://cs231n.github.io/python-numpy-tutorial/

[2] Color detection:
http://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/

[3] Template_matching:
http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html

[4] Camera calibration:
http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html

[5] Object detection using Haar features
http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html

https://www.cs.auckland.ac.nz/~m.rezaei/Tutorials/Creating_a_Cascade_of_Haar-Like_Classifiers_Step_by_Step.pdf

[6] HOG Features with SVM to train:

http://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/
http://stackoverflow.com/questions/11626140/extracting-hog-features-using-opencv
http://study.marearts.com/2014/04/example-source-code-of-extract-hog.html

[7] Example code using HOG features:
https://github.com/vedaldi/practical-object-category-detection/blob/master/doc/instructions.md

[8] HOG Descriptor for Open CV with a Normalizer:
http://docs.opencv.org/3.1.0/d5/d33/structcv_1_1HOGDescriptor.html

Appendices [Attached in the Folder]

1. Code
2. Support Material