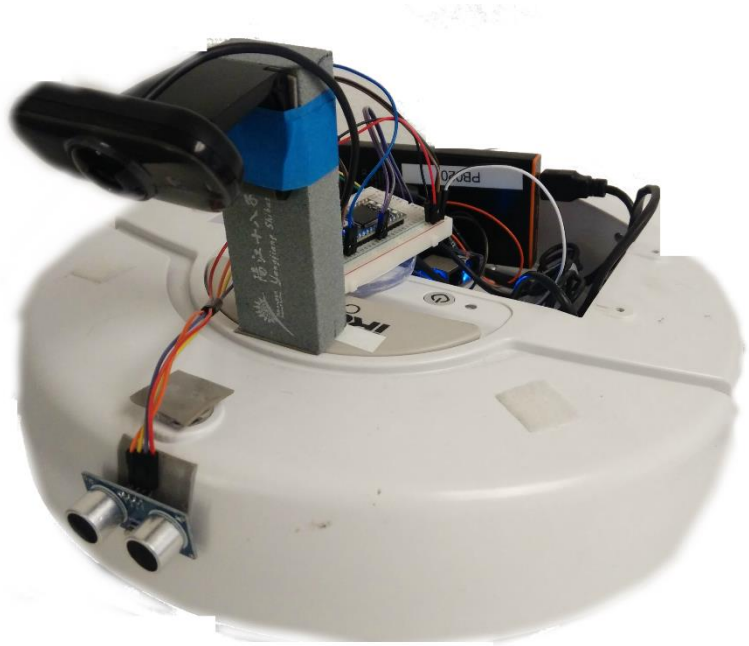
Aragon: Autonomous Remote Controlled Ground Obstacles Avoider Navigator

Himanshu Arora { A53097039}, Jayant Malani (A53102766)

Siddhant Arya (A53079389)



1. Introduction

In this day and age, autonomous systems have permeated our lifestyles in every sphere of life. From mail sorting to amazon package delivery using drones, we now live in an era of intelligent machines that can take smart decisions to make our lives easier. One such application is autonomous self-driven car.

In this report, we talk about how to design a self-driven car with a limited set of resources. We will throw some light of high level architecture, why neural network best solves our purpose for the chosen problem statement, how we used object identification to reduce our training complexity. We will also discuss the problems we faced to design our car, the limitations of our design and future suggested improvements.

2. Project Objectives

To implement an Autonomous Car (or Autonomous Roomba). The basic goals for this bot would be first to follow a path delineated by the space between two parallel white lines analogically similar to lanes on the roads. The bot would maintain its path in its own lane and then be required to detect a series of obstacles. These obstacles can be the presence of an object directly in its path, for which the aim would be to simply come to a stop a few inches away from the obstacle thereby avoiding a collision. Secondly and more interestingly, we also plan to incorporate image recognition with the help of which the bot would be required to autonomously detect a traffic signal or a stop sign. On detecting the traffic signal, the bot would be required to determine the colour of the light and stop when it witnesses a red light and carry on its path when the green signal is detected. The stop signal detection would work in a similar fashion. We plan to implement this using Neural Networks and various Image Processing algorithms tuned using a variety of filters to obtain the desired level of accuracy.

3. Implementation

System Components:

Below table comprises the list of components used in developing our Autonomous Car:

Component	Usage
Ultrasonic Sonar Sensor: HC-SR04	Detect obstacles
Mbed LPC1768	Trigger sonar and communicate distance values to the BBB via the serial port
Logitech C210 Webcam	Capture images for image processing
Beagle Bone Black	To drive the webcam, communicate with Mbed (wired serial connection) and host machine (wifi), multicast webcam stream on http port, run apache server.
Host Machine	The main engine of our project. Trains, drives, makes decision for the robot. Triggers end to end communication between all the components.
iRobot	Plays the role of a car

System Architecture:

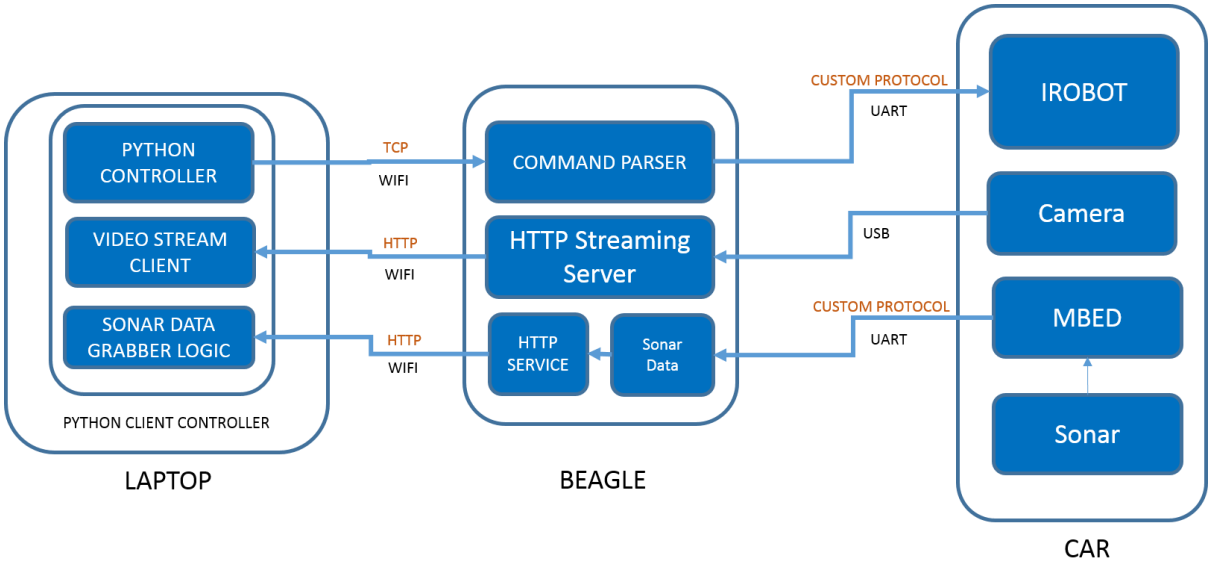


Figure 1 System Architecture

There are many subsystems in our project, which have been decoupled to perform individual tasks. The following section provides a conceptual break down of our project and role of each subcomponent in that section.

LAPTOP –

It contains the 3 Basic Components –

1) Python Controller – It's the brain of the Autonomous car which processes the images and generate the output logic to drive the RC Car. The grabbed images is processed through the trained data and generates the output directions. More explanation is given in Controller Design Section.
2) Video Streaming Client – This part of the code is used to capture the video stream from the HTTP client Streamer
3) Sonar Grabber Logic – It is used to grab the sonar data using from BBB. A simple html call to execute.php present in beagle bone is used to grab the data from the beagle. The data is then read from the index.html file.

Beagle –

1) Command Parser – It is used to parse the directions generated by the controller to the iRobot. A python server is coded which grabs the directions and parse it to iRobot using UART port on Beagle Bone.
2) HTTP Streaming Server – mjpg streamer is used to transfer live video stream from camera to http port on Beagle Bone.
3) Sonar Data Grabber – It is used to grab data from mbed & writes the data in /var/www/index.html
4) HTTP Service – A simple PHP service is developed which is called from Host/Laptop which executes the python script sonar_data_grabber.

Car –

1) iRobot – This is the actual hardware car on which the driving commands are executed. It provides 2 degree of freedom movements. i.e. forward, backward, right and left.
2) Camera – A Logitech USB cam was used to provide a live feed of the track used for image processing
3) Sonar - HC-SR04 device is used to detect obstacles
4) Mbed – The Mbed LPC1768 is used to trigger sonar to output the distance value for the obstacles in front

Controller Design –

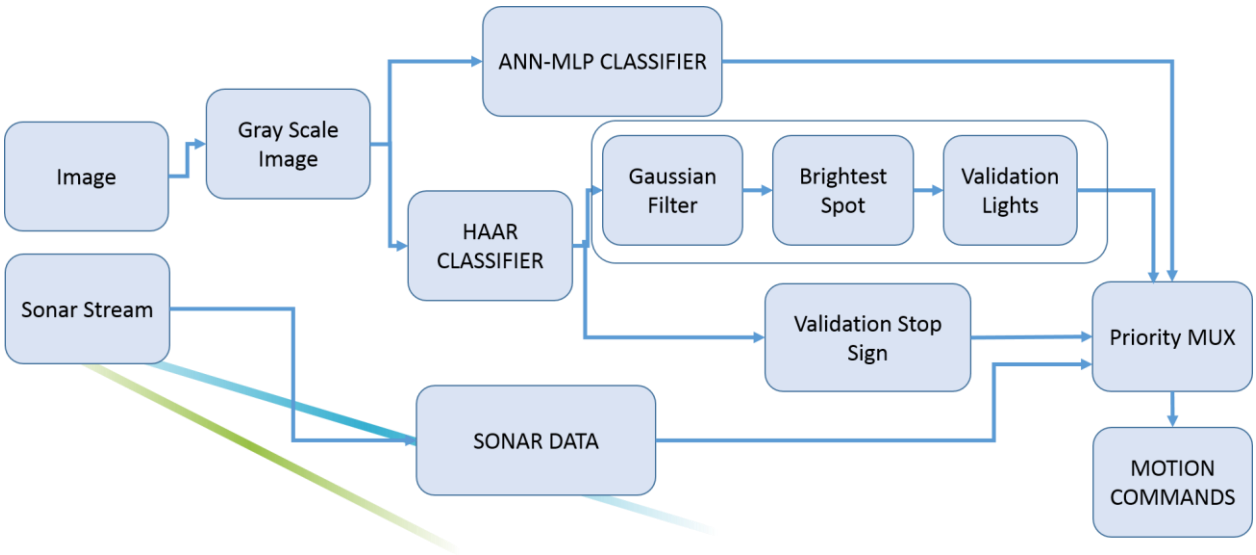


Figure 2 Schematic of Controller

Algorithm:

While(True)

- Neural_Direction ← Get direction from MLP Classifier
- Object_detected_IP ← Get Object Detection
- Object_detected_through Sonar ← Detect object from Sonar

If (Object_detected_through Sonar **Or** Object_detected_IP) send stop command to irobot

Else send Neural_Direction to irobot

The controller is written in python. It is divided into 3 Parts –

Artificial Neural Network Multi–Layer Perceptron Classifier – The image frames from the video feed are first converted to grey scale image. Then the lower half of the image is passed through the

ANN – MLP classifier which gives the direction of movement to take namely (forward, left, right & stop).

Object Detection – In parallel the image from are also passed through Haar Classifier. The trained parameters for stop sign and red lights were already present in their respective xml files which are called and by the Haar function to detect whether the object is present in image or not.

Sonar Fail Safe – The sonar data is provided as a fail-safe such that if any random object comes in range of 30 cm of iRobot it will send a stop command to the iRobot.

The control commands are then given priority with highest priority given to sonar data. The second highest priority is given to command generated by the object detection logic and lowest priority is given to commands generated by MLP classifier.

Neural Network

There are powerful learning algorithms that we can leverage to create such autonomous system. From a plethora of algorithms at our disposal, we chose Artificial Neural Networks (ANN) to implement our project. It is nonlinear model that is easy to use and understand compared to other statistical methods. ANN is non-parametric model while most of statistical methods are parametric model that need higher background of statistic. ANN with Back propagation (BP) learning algorithm is widely used in solving various classifications and forecasting problems. It guarantees convergence, although a bit slowly. However, ANN is black box learning approach, cannot interpret relationship between input and output and cannot deal with uncertainties. To overcome this, several approaches have been combined with ANN such as feature selection and reduction.

We use Multi-layer perceptron (MLP), the most commonly used type of neural networks. It consists of the input layer, output layer, and one or more hidden layers. Each layer of MLP includes one or more neurons directionally linked with the neurons from the previous and the next layer. Since we use a 320px *240px input from the webcam, they can be directly mapped as features of 76800 dimensions for our training set. But we observed that using all the pixels introduces a lot of noise in the training set. After a few trail runs, we decided to use the lower half of the image, since the lane specific features were mostly present in the lower half only, thereby reducing the number of features to 38400 features, a big win!

The diagram below briefly explains the structure of our network. There are 38,400 (320×120) nodes in the input layer and 32 nodes in the hidden layer. Also, there are four nodes in the output layer where each node corresponds to the steering control instructions: left, right, forward and reverse respectively

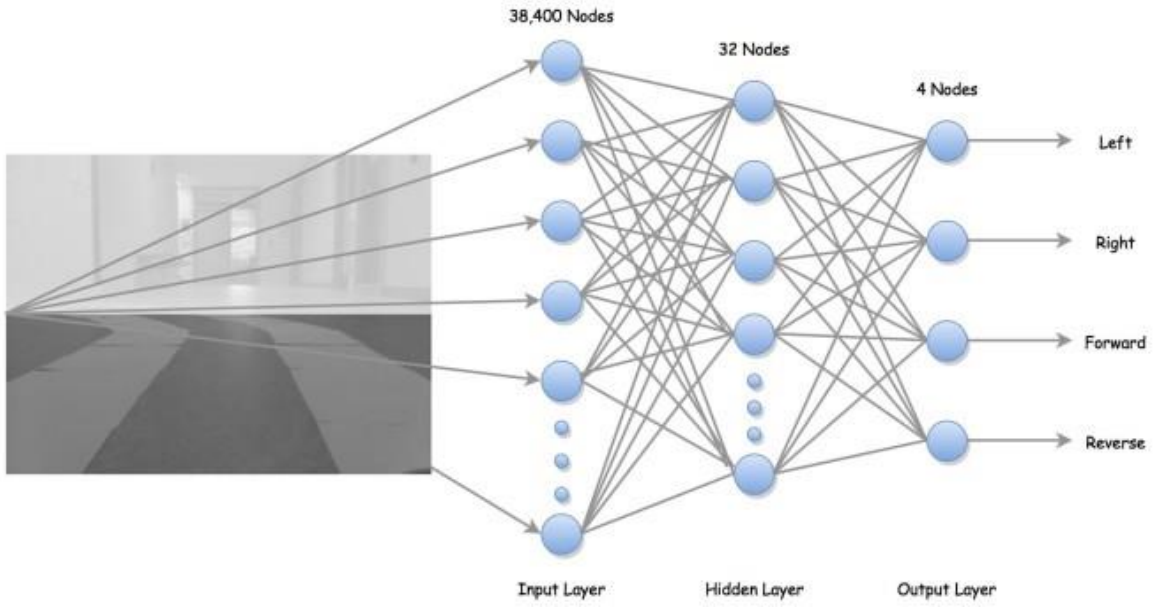


Figure 3 Neural Network

The diagram below demonstrates how we train our neural network to associate driving directions with lane images. In this example shown here, we are associating the forward direction with the image our camera captures. Doing this repeatedly, we generate thousands of samples that we can later use to drive our bot autonomously and make decisions on the basis of the learnt data.

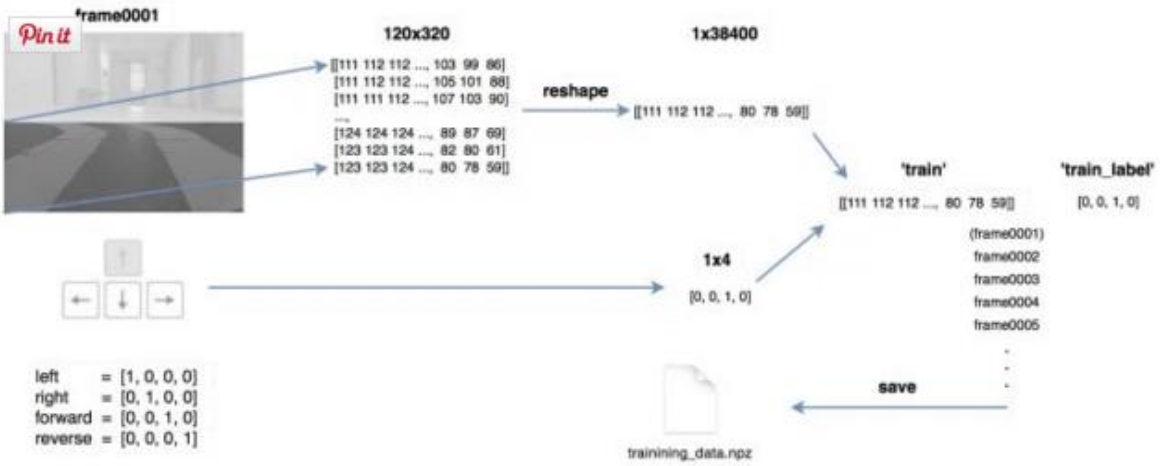


Figure 4 Neural Design Flow Chart

Object Identification

We used Haar feature-based cascade classifiers for object detection. A Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image. Since each object requires its own classifier and follows the same process in training and detection, we restrict ourselves to detect the stop sign and the stop light. Using the open CV trainer and detector, we take numerous positive and negative sample and crop the noise

out of the positive samples. The negative samples are random images the haar classifier uses to create the cascade XML. The diagram below shows the flow of our classifier algorithm for objects.

Figure 5 Object Detection Flow Chart

Image detection for Red/Green lights –

To recognize different states of the traffic light(red, green), some image processing is needed beyond detection. Flowchart below summarizes the traffic light recognition process. Firstly, trained cascade classifier is used to detect traffic light. The bounding box is considered as a region of interest (ROI). Secondly, Gaussian blur is applied inside the ROI to reduce noises. Thirdly, find the brightest point in the ROI. Finally, red or green states are determined simply based on the position of the brightest spot in the ROI.

Team composition and effectiveness

The tasks in this project were mostly done in a cohesive way. Broadly speaking, the following section will describe a high level classification of the tasks and responsibilities. Some tasks like software installations, hardware configuration cannot be divided, since they done by each one of us.

Jayant Malani –

- Worked on the python scripts to train and drive the robot.
- Wrote scripts that interacted with mbed to get the data serially and configured pygame to drive robot using direction key strokes.
- designed the environment on which the training was to be done
- trained the robot
- wrote neural network scripts to return the driving direction on the basis of current input

Siddhant Arya –

- Conceptualised the project idea.
- His knowledge of linux environment helped us accelerate the project at a fast pace.
- Most of the linux related issues we take care of by him.
- Wrote the C program for HAAR classifier

Himanshu Arora –

- Video stream portion of the of the project.
- Worked on getting the sonar data from the Mbed controller and redirecting the output on the serial port.

- Wrote web service to get the sonar data from the Beagle Bone.
- worked on algorithm analysis to find the best suited algorithm for our problem statement

3. Result

Prediction on the testing samples returns an accuracy of 85% compared to the accuracy of 96% that the training samples returns. In actual driving situation, predictions are generated about 10 times a second (streaming rate roughly 10 frames/s).

Haar features by nature are rotation sensitive. In this project, however, rotation is not a concern as both the stop sign and the traffic light are fixed objects, which is also a general case in real world environment.

4. Conclusion

Overall, the RC car could successfully navigate on the track with the ability to avoid front collision, and respond to stop sign accordingly. This project gave us an insight of how different components can be amalgamated to solve an innovative problem. We were able to architect an intensive in a modular fashion which led to efficient utilization of the limited resources at our disposal.