B.Tech - Project
on

Autonomous Car Project - Automated Driving System Design and Implementation

(Project code: PI-11)

Submitted by:
Akash Mahajan - 2016ME20748
Dhruv Talwar - 2016ME10670

Supervisor:
Prof. Sunil Jha
Mechanical Deptt., February 2020

Abstract

In recent years, a lot of research has been conducted on Autonomous cars and control. This report discusses a section of the parent project, based on the development of an Autonomous Controller for the self driving Mahindra-e2o. The primary aim is to design an Automated Driving System with lane keep assist system which has two components, namely, perception and motion planning.

We have designed algorithms and used pre existing libraries on MATLAB and then also integrated it with ROS for live communication. Gazebo simulation system, video sources and also live video feed were used for testing some of the algorithms and the qualitative comparison of the designed models. The main target that we wish to achieve is to exploit the capabilities of MATLAB, and use it inbuilt libraries, models and algorithms to achieve our desired output.

Contents

List of Figures

Chapter 1: Introduction

1.1 Background

The automobile industry seems to be on the brink of a giant technological leap. It has come a long way since the development of the first commercial car in the early 20th century. Autonomous / Self driving / Robotic cars have recently attracted a lot of attention all round the world. An autonomous car can sense its environment, interpreting different sensory information (LIDAR, GPS, Computer Vision) and guide itself safely with little or no human input.

The world's first radio controlled car was 'Linrrican Wonder' (1926). Over the years many companies like Mercedes, Google, Nvidia, Tesla have invested heavily in autonomous cars and significant progress has been made. There are broadly 5 different levels of autonomous driving -

Level 0 : The system can only issue some warnings based on certain sensor readings. The control remains mainly manual
Level 1 : The automated system can control certain systems like steering system during Parking Assistance ; speed during Cruise Control
Level 2: Partial automation. The system can control the steering, speed, breaking of the vehicle but the driver has to continuously monitor the car
Level 3: The driver can take his/her eyes off, but the driver must be ready to take control within a specified time interval
Level 4: No human attention is needed but the driver still remains in the car. The automated system should have the capability to follow all safety protocols.
Level 5: 100% automation. No driver in the car.

Some of the recent autonomous systems launched could operate at Level 3 and 4. The driverless system we are implementing works at the Level 2, where the car automates all the control systems but under continuous monitoring of the driver. In this report, we implement our autonomous system using softwares like MATLAB, ROS and Gazebo which could simulate real life data using sensors and virtual environment.

Autonomous vehicles can help solve major problems faced in India - traffic jams, accidents (93% of accidents are due to driver negligence), environment impacts, parking space etc but the unique nature of chaotic, diverse Indian roads and transport pose a big challenge in designing successful self driving cars.

1.2 Overview

For our undergraduate thesis 2, we wish to design a vision based assist system, capable of controlling the e2o car autonomously keeping the stereo camera as the global sensor. In our approach we wish to use the technology which is developed by Mathworks and mould it according to our needs. We initially understood and worked on raw color and image detection using the MATLAB vision toolbox and then we tested them on the ROS Toolbox (Robot Operating System). For the simulation environment we tested the ROS-MATLAB framework using the Kinect sensor of the turtlebot burger. Because of the real life physics involved in the gazebo environment, we are able to visualize what the code is capable without using it on a real robot.

Chapter 2: Literature survey

2.1 Nvidia Isaac

NVIDIA Isaac is a developer toolbox for accelerating the development and deployment of AI-powered robots. The Isaac SDK is the main software toolkit for NVIDIA robotics, and is comprised of the following:

- Isaac Robot Engine: A framework which allows you to easily write modular applications and deploy them on your robots.
- Isaac GEMs: A collection of robotics algorithms from planning to perception, most of them GPU-accelerated.
- Applications: Various example applications from basic samples which show specific features to applications that facilitate complicated robotics use cases.

2.2 ROS

ROS is an open-source, meta-operating system for robots. It provides services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It provides tools and libraries for building, writing, and running code across multiple computers.
ROS-based processes work on a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator, and other messages.

2.3 ZED Stereo Camera

ZED Camera is a product of Stereo Labs. It is a 3D camera for depth sensing, motion tracking and real-time 3D mapping. It provides a wide-angle all-glass dual lens image with reduced distortion. It is the main sensor for vision that is used for this project, we have tried to perform all MATLAB experiments on ZED camera feed.

2.4 MATLAB Automated Driving Toolboxes

Automated Driving Toolbox™ provides algorithms and tools for designing, simulating, and testing ADAS and autonomous driving systems. Using this toolbox we can design and test vision and lidar perception systems, as well as sensor fusion, path planning, and vehicle controllers.

Visualization tools include a bird’s-eye-view plot and scope for sensor coverage, detections and tracks, and displays for video, lidar, and maps. Using the toolbox we can import and work with HERE HD Live Map data and OpenDRIVE® road networks.

The toolbox supports C/C++ code generation for rapid prototyping and HIL testing, with support for sensor fusion, tracking, path planning, and vehicle controller algorithms.

Chapter 3: Project Objectives and Workplan

3.1 Problem Definition/ Motivation

Recently, Mahindra took an initiative to encourage research on autonomous vehicles in India through the Mahindra Driverless Car Challenge. The challenge is divided into levels. Level 0 challenge includes simple tasks such as lane driving and reading traffic signals. The complexity of the challenge increases and level 3 will address problems such as avoiding a truck with protruding rods, identifying shallow ditches and unmarked bumps, wading through waterlogged streets and night driving capabilities. Team dLive at IIT Delhi is participating in this competition and is currently working together to develop a fully autonomous vehicle.

3.2 Objectives of the work

Our objective is to move a driverless car on road at Level 2 stage of autonomous driving with the help of lane following.

Goals of the project are:

- Exploring the MATLAB automated driving system.
- Implementing the optimal perception(vehicle, pedestrian and lane detection) on video feed
- Develop perception systems using algorithms, sensor models, applications for computer vision processing, and sensor fusion.
- Form a Bridge between ROS and MATLAB platforms for testing on Mahindra e2o.
- Design the motion planning (steering) for lane following and Acceleration/Braking control.

3.3 Methodology

The goal we opted for is to make the car move autonomously using vision system feedback. We started by studying about some of the existing platforms for Designing Autonomous Driving Systems like Nvidia Isaac and MATLAB. We proceeded with MATLAB to develop the perception systems and algorithms.

On successfully running the vision models of lane detection, pedestrian and vehicle detection, we decided to integrate these with ROS which would help in implementing them on Mahindra e2o car. We further plan to integrate the car control system with the algorithms to improve and filter out the algorithms. This will help us in providing a robust autonomous driving system.

Below is the Gantt chart representing some of the objectives we have completely achieved and also our future work plan.

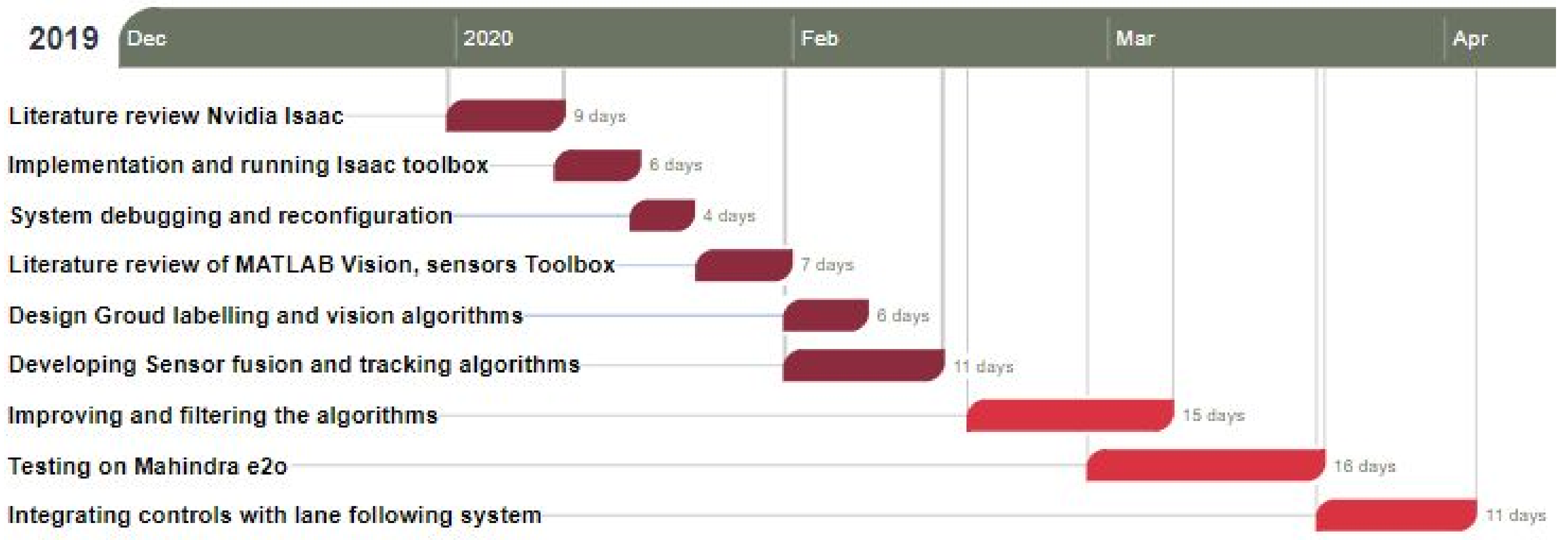


Fig 3.1 Gantt Chart

Chapter 4: MATLAB Vision ToolBox

4.1 Birds eye view and Grayscale Image

We Implemented a rough transformation to create a bird's-eye view of a 2-D scene using inverse perspective mapping. For this purpose we have to define the camera intrinsics and create an object containing these intrinsics. We calibrated the camera using the inbuilt MATLAB app, and found out the respective parameters.
We also converted the birds eye image into a grayscale image which is a binary image that represents lane features. Using Canny edge detection and applying Hugh Transform, we can generate the lines for lane following.

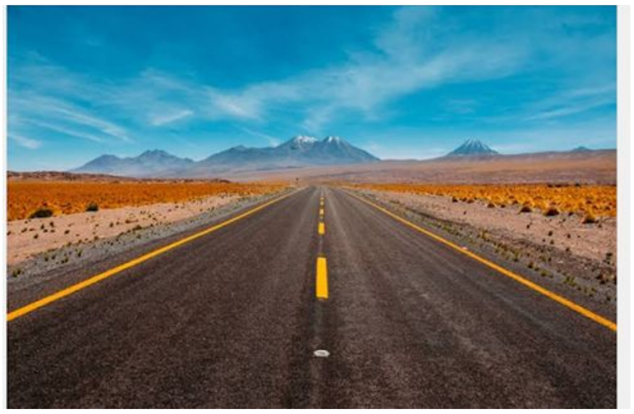


Fig 4.1 Original Road Image

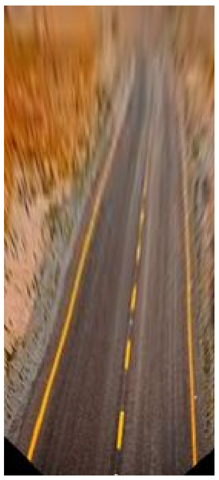


Fig 4.2 Birds Eye Image of Road

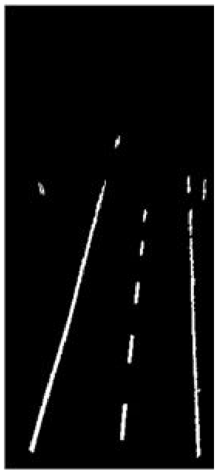


Fig 4.3 Grayscale Image of Road

4.2 Pedestrian and Vehicle Detection

We now shifted to detecting human aka pedestrians and vehicles on the road. MATLAB uses Aggregate Channel Features (ACF) Algorithm to detect humans as well as vehicles. This algorithm detects people or vehicles in an input image using the Histogram of Oriented Gradient (HOG) features and a trained Support Vector Machine (SVM) classifier. The object detects unoccluded people in an upright position, or a vehicle.
For pedestrian detection we made a video of our own and used the algorithm on it. And we downloaded a video of a road with a lot of cars for vehicle tracking. After running the algorithm we get the locations of detected upright people in the input image in bounding boxes along with the percentage confidence in the image. When no people are detected, the step method returns an empty vector. The input format must be a grayscale or truecolor (RGB), for this experiment we used RGB videos.

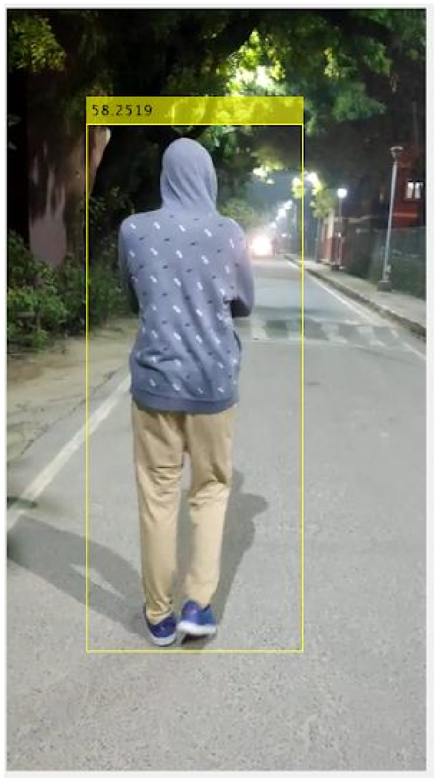


Fig 4.4(a)

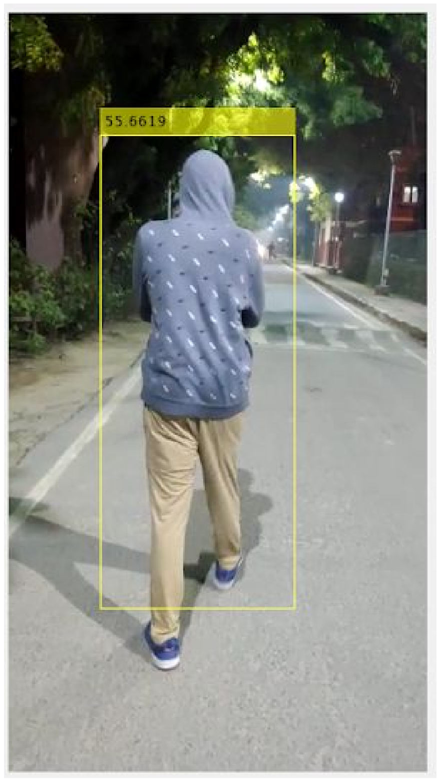


Fig 4.4 (b)

Fig 4.4 Human Detection from live video

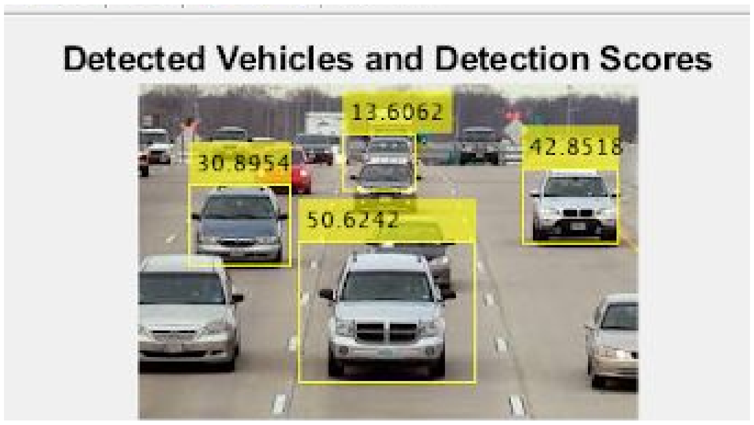


Fig 4.5 Vehicle Detection from live Video

4.3 Neural Networks for Object Detection

We used Google's pre existing googlenet for object detection. GoogleNet is a pretrained convolutional neural network that is 22 layers deep. The network trained on ImageNet classifies images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

These networks have learned different feature representations for a wide range of images. The networks both have an image input size of 224-by-224. We used Googlenet just to get a hang of CNN, as we will be using models trained by our own data sets to detect the road signs in the institution. The problem with using googlenet is that we are not aware of its training data, and it might be trained with objects which are usually found in the USA.

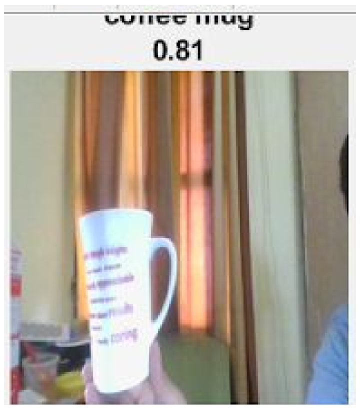


Fig 4.6 Object Detection: Coffee Mug

Chapter 5: MATLAB ROS Toolbox

5.1 ROS MATLAB Interface

ROS Toolbox provides an interface connecting MATLAB with the Robot Operating System enabling us to create a network of ROS nodes. The toolbox includes MATLAB functions and Simulink blocks to import, analyze, and play back ROS data recorded in rosbag files. The toolbox lets us verify ROS nodes via desktop simulation and by connecting to external robot simulators such as Gazebo.

First we set a connection between MATLAB and ROS so that they can communicate. Then we receive some sensor data from ROS, it could be image data or Lidar data, then MATLAB processes the image or sensor input and sends its output back to ROS, to a robot or to a robot simulation.

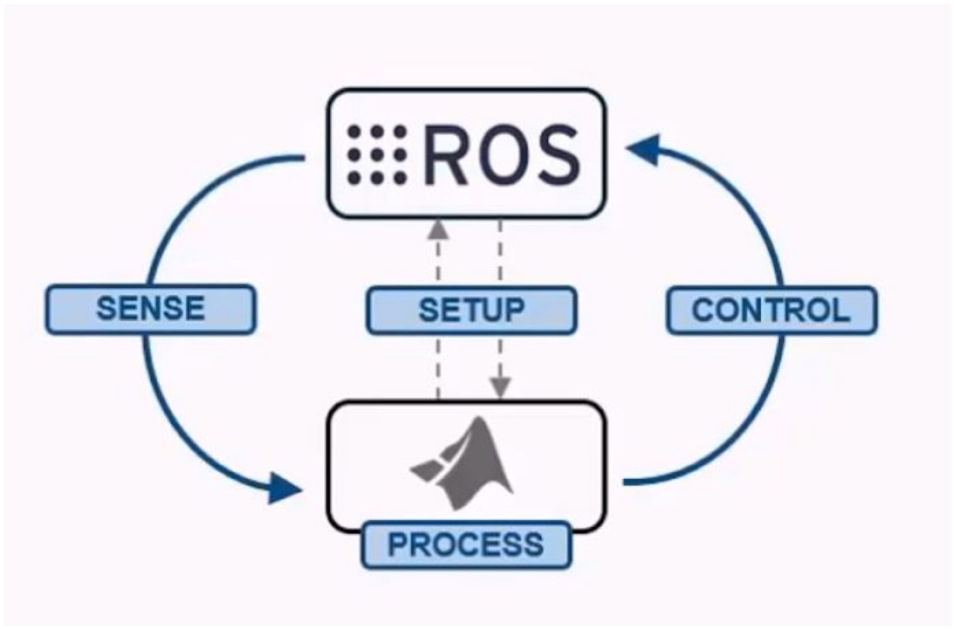


Fig 5.1 Workflow between ROS and MATLAB

5.2 Object Detection in Gazebo Environment

We launched a turtlebot simulation in ROS Gazebo, we then connected our MATLAB with the IP address of the ROS masters. This way both ROS and MATLAB can communicate with each other. Then we wrote a MATLAB code which subscribes to the camera topic of the turtlebot and then the image feed which is received is then processed according to the algorithm that we wrote in MATLAB. In this case we wish to detect the STOP sign.

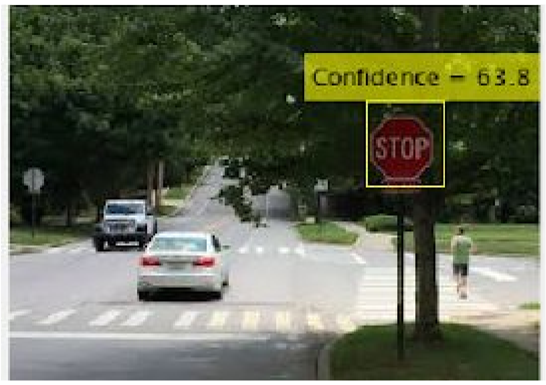
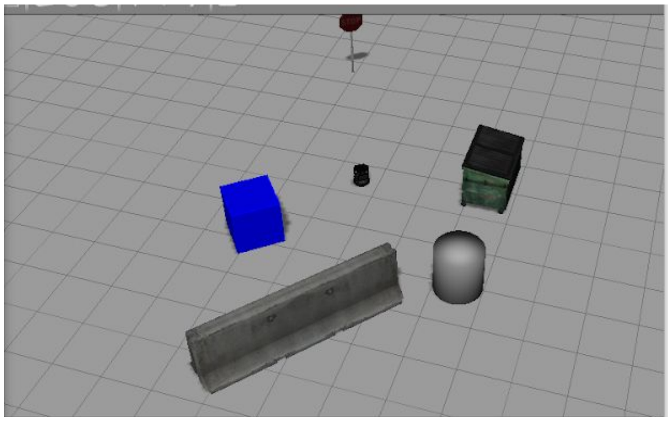


Fig 5.2 Detecting the Stop sign using Vision Toolbox



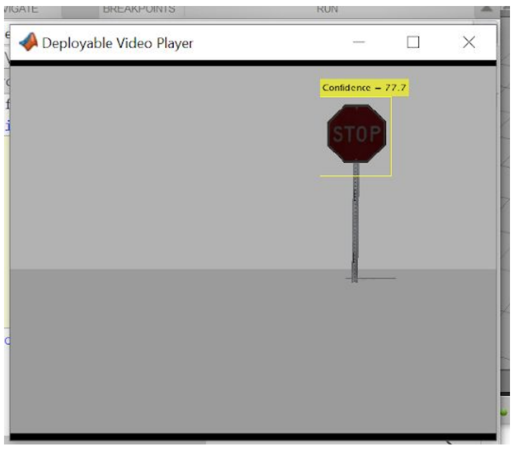


Fig 5.3 Stop Sign in the ROS Environment

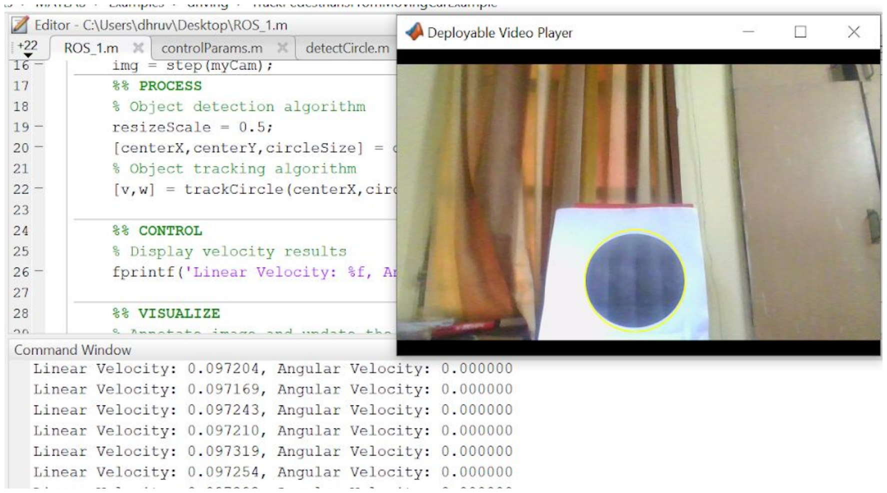
Fig 5.4 Detecting the STOP sign in

MATLAB

5.3 Object Detection and Tracking for TurtleBot

To establish a proper communication with the We launched a turtlebot simulation in ROS Gazebo, we then connected our MATLAB with the IP address of the ROS master. We ran an algorithm that will first track or detect the position of a blue circular object and then it will send out command velocities to the robot so that the object remains in the center of the frame and at a particular distance.We are using blob analysis for the circle detection.

In the following figure we just detected a circle with the webcam of the laptop, there was no ROS-MATLAB communication here. It was just a stand alone MATLAB code, which detected the circle and printed the linear and angular velocity that should be sent to the robot.



```
        img = step(myCam);
        %% PROCESS
        % Object detection algorithm
        resizeScale = 0.5;
        [centerX,centerY,circleSize] = d
        % Object tracking algorithm
        [v,w] = trackCircle(centerX,circ

        %% CONTROL
        % Display velocity results
        fprintf('Linear Velocity: %f, An

        %% VISUALIZE
```

```
Linear Velocity: 0.097204, Angular Velocity: 0.000000
Linear Velocity: 0.097169, Angular Velocity: 0.000000
Linear Velocity: 0.097243, Angular Velocity: 0.000000
Linear Velocity: 0.097210, Angular Velocity: 0.000000
Linear Velocity: 0.097319, Angular Velocity: 0.000000
Linear Velocity: 0.097254, Angular Velocity: 0.000000
```

Fig 5.5 Detecting the Circle to get control Velocities

Continuing on the above example, we now published the above printed Linear and Angular velocities to the turtlebot in the Gazebo environment. As expected the robot detected the blue circle and moved a particular distance away from the object.

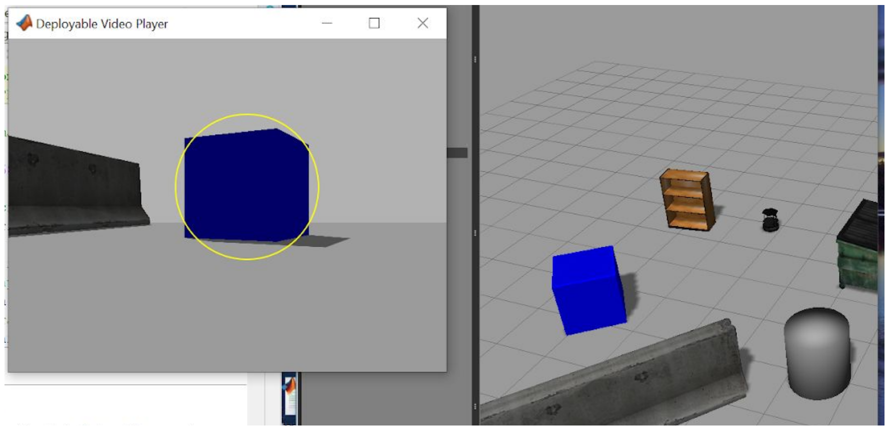


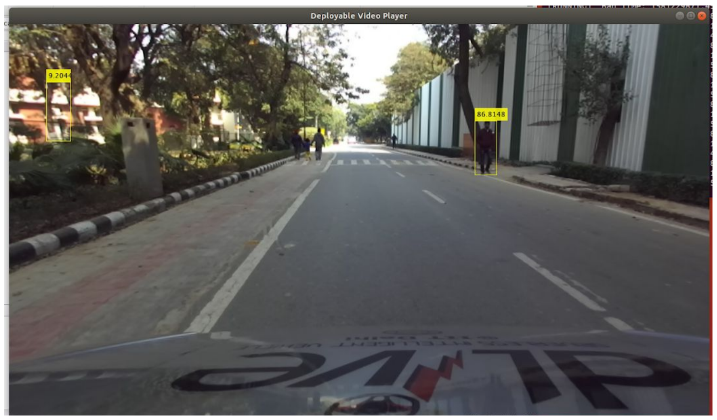
Fig 5.6 Detecting the Circle and publishing Control Velocities to the turtlebot

Chapter 6: Pedestrian and Car Detection in Real World

6.1 Pedestrian Detection

Using the same algorithm as used in section 4.2 we detected humans on the road which were recorded when we took our car for testing. We were able to record the zed image topic of the journey of the car in a rosbag file. After this we played this bag file in Ubuntu 18.04 and subscribed to the recorded ROS topic. We then ran the pedestrian detection algorithm on this.

The results were not quite accurate as the algorithm would also sometimes identify the trees as humans. It would also sometimes miss humans depending upon the lighting conditions. The processing frame rate was also quite low as MATLAB and ROS running together require huge computation power.



`*Fig 6.1(a)*

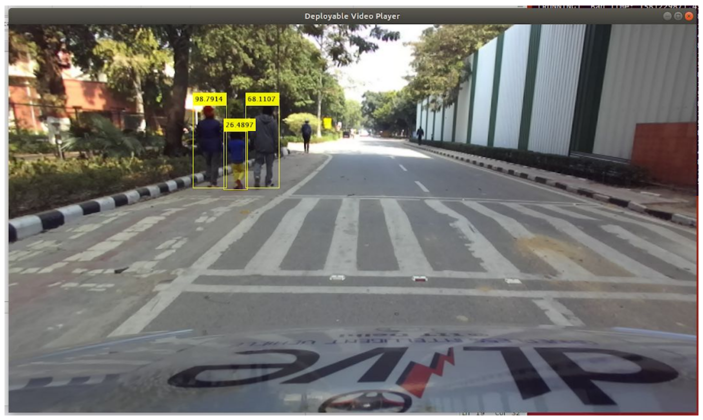


Fig 6.1(b)

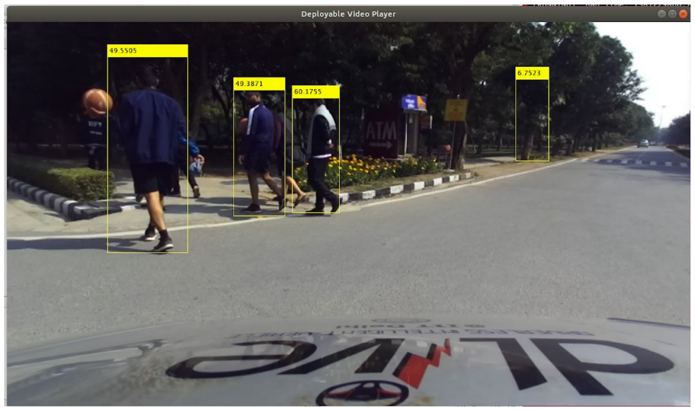


Fig 6.1(c)

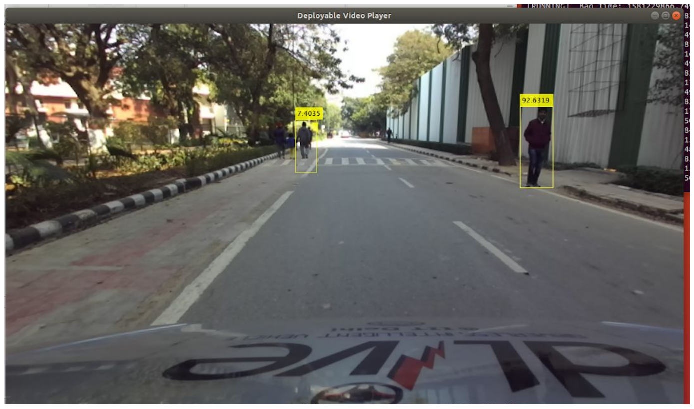


Fig 6.1(d)

Fig 6.1(a)(b)(c)(d) Live Pedestrians Detected during Testing

6.2 Vehicle Detection

Using the same algorithm as used in section 4.2 we detected vehicles on the road which were recorded when we took our car for testing. We followed the same steps as mentioned in Section 6.1

The results were quite accurate and it would even identify cars which were parked in a shady and dark place. The algorithm identified both moving as well as stationery cars.The processing frame rate was again also quite low as MATLAB and ROS running together require huge computation power.

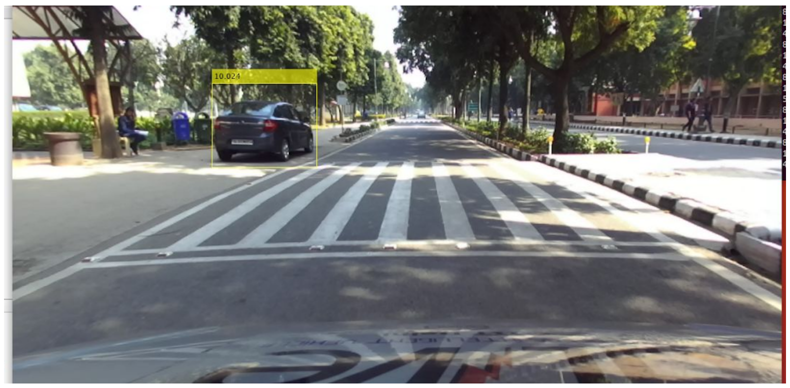


Fig 6.2(a)

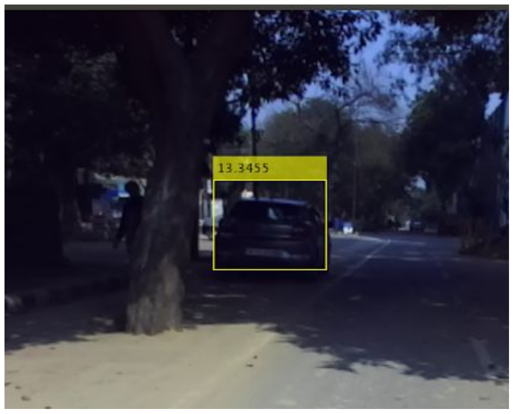


Fig 6.2(b)

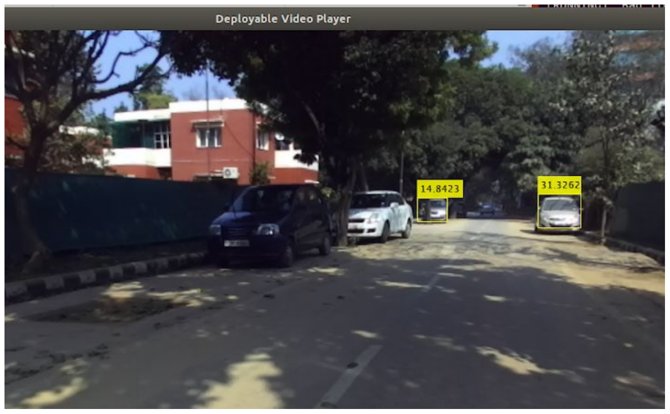


Fig 6.2(c)

Fig 6.2(a)(b)(c) Live Cars Detected during Testing

Chapter 7: Conclusions and Results

7.1 Conclusion

7.1.1 Vision Implementation

We were successfully able to use MATLAB Vision ToolBox to implement Pedestrian and Vehicle Detection on images and then also on video feed from the e2o ZED camera. We also tested existing Neural network models by Google for object detection and then also used the MATLAB vision toolbox for object tracking.

7.1.2 ROS MATLAB Integration

We set up connections between MATLAB and ROS for integrating their functioning. On ROS (Robot Operating System) we created a simulation environment using 3D physics engine Gazebo with a turtle bot to test various algorithms. Object detection was implemented to identify the STOP signs. along with controlling the robot based on the vision system outputs.

7.2 Future Work

Now we aim to improve our algorithms for object, pedestrian and vehicle detection and then integrate them with the Mahindra e2o control. On achieving successful perception functioning, the car could run driverlessly by a robust lane keep assist system that is responsive to the immediate environment and be safe to run even in the IIT Delhi campus.

References

1. Coursera Course on Self Driving Cars by University of Toronto
   https://www.coursera.org/learn/intro-self-driving-cars
2. MATLAB Birdseye tutorials
   https://in.mathworks.com/help/driving/ref/birdseyeview.html
3. MATLAB People Detection
   https://in.mathworks.com/help/vision/ref/peopledetectoracf.html#bvn_p7e-
4. ROS
   http://wiki.ros.org/
5. Gazebo Tutorials
   http://gazebosim.org/tutorials
6. MATLAB Deep Learning Tutorials
   https://www.mathworks.com/help/deeplearning/examples/classify-images-from-webcam-using-deep-learning.html
7. MATLAB and ROS tutorials
   https://in.mathworks.com/help/ros/ug/get-started-with-ros.html
8. MATLAB Object Detection Tutorials
   https://www.mathworks.com/help/ros/ug/sign-following-robot-using-ros-simulink.html
9. Kalman filter
   https://www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/introduction-to-kalman-filter-and-its-applications

10. MATLAB Track and Follow Tutorials
    https://www.mathworks.com/help/ros/ug/track-and-follow-an-object.html

11. MATLAB Automated Driving toolbox
    https://in.mathworks.com/products/automated-driving.html