International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 12 | Dec 2021 www.irjet.net p-ISSN: 2395-0072

Design and Development of an Autonomous Car using Object Detection with YOLOv4

Rishabh Chopda[1], Saket Pradhan[2], Anuj Goenka[3]

[1]Dept. Of Computer Engineering, Thakur College of Engineering & Technology, India, Maharashtra, Mumbai
[2]Dept. Of Information Technology, Thakur College of Engineering & Technology, India, Maharashtra, Mumbai
[3]Dept. Of Computer Engineering, Thakur College of Engineering & Technology, India, Maharashtra, Mumbai

---

Abstract - *Future cars are anticipated to be driverless; point-to-point transportation services capable of avoiding fatalities. To achieve this goal, auto-manufacturers have been investing to realize the potential autonomous driving. In this regard, we present a self-driving model car capable of autonomous driving using object-detection as a primary means of steering, on a track made of colored cones. This paper goes through the process of fabricating a model vehicle, from its embedded hardware platform, to the end-to-end ML pipeline necessary for automated data acquisition and model-training, thereby allowing a Deep Learning model to derive input from the hardware platform to control the car's movements. This guides the car autonomously and adapts well to real-time tracks without manual feature-extraction. This paper presents a Computer Vision model that learns from video data and involves Image Processing, Augmentation, Behavioral Cloning and a Convolutional Neural Network model. The Darknet architecture is used to detect objects through a video segment and convert it into a 3D navigable path. Finally, the paper touches upon the conclusion, results and scope of future improvement in the technique used.*

Key Words: autonomous, self-driving, computer vision, YOLO, object detection, embedded hardware

1. INTRODUCTION

This A 'Self-Driving Car' is one that is able to sense its immediate surroundings and operate independently without human intervention. The main motivation behind the topic at hand is the expeditious progress of applied Artificial Intelligence and the foreseeable significance of autonomous driving ventures in the future of humanity, from independent mobility for non-drivers to cheap transportation services to low-income individuals. The emergence of driverless cars and their amalgamation with electric cars promises to help minimize road fatalities, air and small-particle pollution, being able to better manage parking spaces, and free people from the mundane and monotonous task of having to sit behind the wheel. Autonomous navigation holds quite a lot of promise as it offers a range of applications going far beyond a car driven autonomously. The main effort here is to keep the humans out of the vehicle control loop and to relieve them from the task of driving. The prime requisite of self-driving vehicles are the visual sensors (for acquiring traffic insight of vehicle surroundings), microprocessors or computers (for processing the sensor information and transmitting vehicle control instructions) and actuators (to receive said instructions and be responsible for the longitudinal and lateral control of the car) [1-4]. Autonomous vehicles are also expected to be manoeuvred in many of the most complex human planned endeavours, such as asteroid mining [5]. The meteoric rise of AI along with deep learning (DL) methods and frameworks, have made possible the development of such autonomous vehicles by many venture companies at the same time.

2. SOFTWARE DEVELOPMENT

In this section we elucidate the entire software development process which includes data collection and labelling, model training and model deployment.

1.1 Data Collection & Labelling

Around 2,000 images were collected for two types of colored cones, namely: Orange and Blue. The cones were made from craft paper and were 4.5 centimeters tall with a base diameter of 3cm. The pictures included the cones laid out as track, single color cones, multiple same-colored cones and a mix of the two cones. A total of 16,382 cones were observed in the collected images with LabelImg being later used to label these cones from the images. 'LabelImg' is a graphical image annotation tool [6]. It is written in Python and uses Qt for its graphical interface. The LabelImg tool was used to label the photographed images in the YOLO format by drawing bounding boxes around the cones and naming each cone with their respective class i.e., color (orange or blue). After labelling via LabelImg, a common class file was created to all images which contained the two classes "Orange" and "Blue". Another file was created unique to each image which contained the coordinates of each cone present in that image. For example, 1 0.490809 0.647894 0.235628 0.342580 is an entry from the class file created where the first parameter determines the class of the cones, the second and third parameters determine the midpoint of the bounding box while the fourth and fifth parameters determine the height and width of the bounding box. For the randomization and renaming of the images, a software tool called 'Rename Expert' was used. It randomized the images and then named them from 0-1681. Data augmentation was used to

increase the amount of data by adding slightly modified copies of already existing data. It involves injecting some noise, rotation and flipping of the images to increase the number of images used for training. It usually helps in preventing overfitting the model and acts as a regularizer [7].

1.2 Model Training

YOLOv4 Tiny, a version of YOLOv4 developed for edge and lower-power devices, is a real-time object detection algorithm capable of detecting and providing bounding boxes for many different objects in a single image [8-11]. The model achieves this by dividing an image into regions and then predicting bounding boxes in addition to the probabilities for each region. Relative to inference speed, YOLOv4 outperforms other object detection models by a significant margin. We needed a model that prioritizes real-time detection and conducts the training on a single GPU as well. 'Darknet' is a framework like the TensorFlow, PyTorch and Keras that proved to be apt for the task at hand. While Darknet is not as intuitive to use, it is immensely flexible, and it advances state-of-the-art object detection results. We train the model on darknet and then later convert it to TensorFlow for ease in usability. This model can be tested on a physical model or on virtual simulators [12-15]. In terms of training the model, the labelled dataset was segregated into training and validation datasets and was uploaded on cloud VM. After that, the darknet was cloned and built on which the model was trained. The parameters were configured periodically to achieve the best weights. It was important that we convert our darknet framework into TensorFlow because only then could we make use of the TensorFlow lite model which is optimized for embedded devices such as Jetson Nano to make the inference at the edge.

1.2 Deployment

YOLOv4 Deployment includes reading the coordinate text data generated from the YOlO4 model into a NumPy framework and labelling the coordinate points according to the two classes, blue and orange. This is done by iterating through the text data line by line, and appending the required point objects into a python array, and finally converting the array into a NumPy format. Matplotlib is used to visualize the set of data points from the camera's perspective, on a 10 x 10 cm2 adjusted screen. Using the Scikit-Learn Library, a Linear Regression model is trained using the NumPy data. Two different models are to be trained; one for the blue set of cones, and one for the orange. Using the 'LinearRegression()' predefined method in the Scikit-Learn library, we could easily create a simple regression model without having to build the entire code for the model ourselves. The data is zipped and iterated through using a for loop. The output generated is explicitly converted into a list format. Two lines are created that pass through the orange cones and the blue cones. Again, a graph is plotted of Matplotlib for visual aid of the lines. Next, the equations of the previously formed lines are derived using simple geometric calculations. Straight line equations of the type: ax + by + c = 0 are obtained for both blue and orange lines. Next, the point of intersection of the two lines is calculated using the formula of point of intersection. The offset of this line is calculated from the centre of the screen and the x-coordinate of each point is subtracted by the corresponding point on the centre of the screen. This value is the mean deviation and will be used further to calculate the angle by which servo attached on the assembly is to be turned. Fig. 1 shows the outcome of the entire video capture and path mapping process.

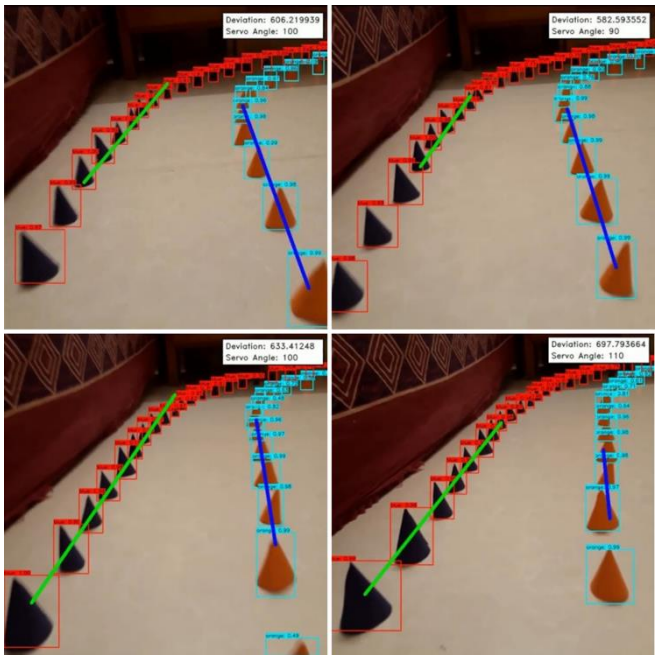

Fig -1: Video capture and path mapping process

3. Hardware Design

Before The car was designed and built with the proper placement and positioning of electronic components, such as the camera, in mind. It consists of three main parts, the steering assembly, the spur gear gearbox and the wheels. The steering system has a rack and pinion type design, chosen for its simple assembly and for providing easier and more compact control over the car. A 3-sided gear box ensures the effortless placement and positioning of the axles and larger gears. Given the opposing forces caused by the axles and front chassis, it also stays strong and sturdy. Spur gears are used in the gear box as they have high power transmission efficiencies (95% to 99%) and are simple to design and install. The wheels are designed and entirely 3D printed to have built-in suspension providing additional steering stability. Because the wheels must be flexible, TPU (Thermoplastic Polyurethane) is used to produce them. All other 3D printed components were produced using PLA (Polylactic acid) as it's easy to use, has a remarkably low printing temperature compared to other thermoplastics and produces better surface

details and sharper features. A list of all materials is given below:

List Of Materials: All components required for the prototype, including sensors, actuators, power supply, and hardware, are listed here. Fig. 2 and Fig. 3 show all the 3D printed parts and their assembly in SoildWorks Simscape respectively.

- 3D Printed Parts
- 608zz Bearings (4x)
- Nvidia Jetson Nano
- 1200KV Brushless DC Motor
- 20A ESC (Electronic Speed Controller)
- 5000mAh Power Bank
- 11.1V - 2200mAH LiPo Rechargeable Battery
- PCA9685 16 Channel Servo Driver
- TowerPro SG90 180° Rotation Servo Motor
- Logitech C615 HD Webcam

Fig -2: 3D Printed parts

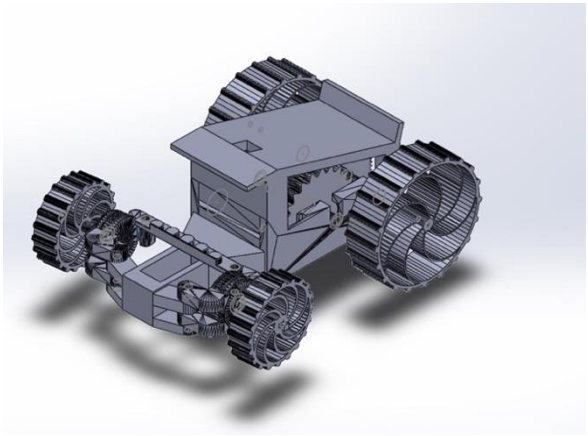

Fig -3: Car assembly on Solidworks Simscape

4. Functionality

Before A Nvidia Jetson Nano single-board computer (SBC) serves as both the brain and the communication node in the prototype control system. This SBC receives data from the camera, analyses them, and integrates them into the navigation system to determine the steering angle. A 11.1V - 2200mAH LiPo battery is used solely to power the vehicle's propulsion system, that is, the 1200KV Brushless DC Motor with a 20A ESC. A 180° rotation servo motor with a torque of 1.2KgCm, controlled by the PCA9685 16 Channel Servo Driver, is used to steer the car. Fig. 4 and Fig. 5 show a flowchart of the instruction feedback loop and a schematic diagram of the hardware connections respectively. Fig. 6 shows the entire assembled car.

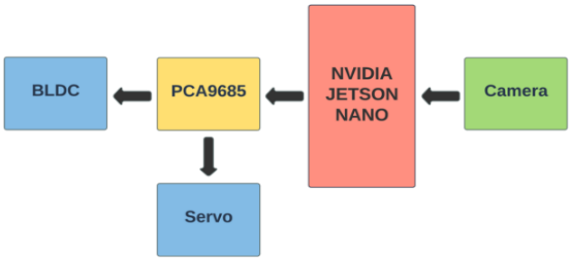

Fig -4: Flowchart of the instruction feedback loop

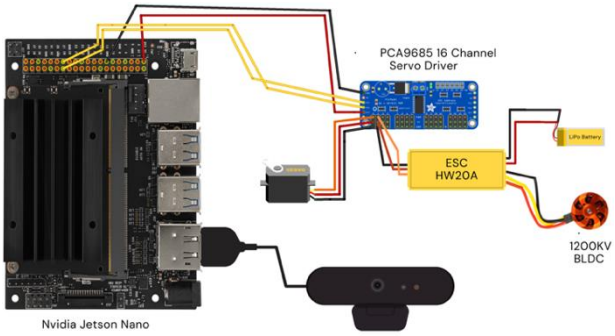

Fig -5: Circuit Diagram

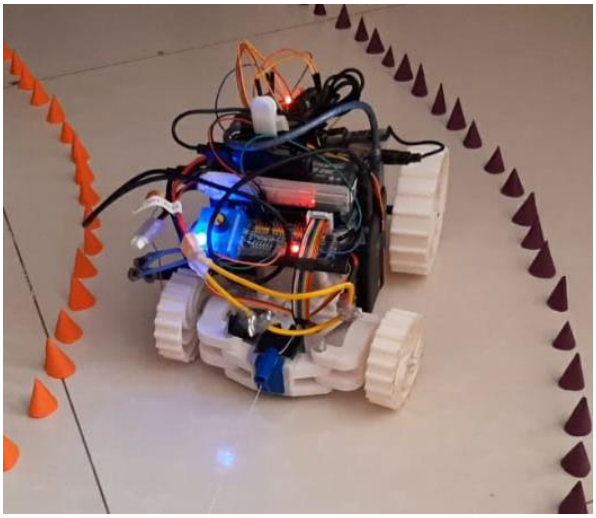

Fig -6: Assembled Car

5. CONCLUSIONS

Through this paper, we present an approach for designing and building a model self-driving car based on the concept of Behavioral Cloning. This approach being an end-to-end one does not require any of the conventional tasks of feature extraction or connection of various modules, which are often monotonous, manual in nature and necessary for efficient working. Our model car is tried and

tested in real life against various standard models such as DenseNet-201, Resnet-50, and VGG19 for the comparison and performance. The final proposed model is a convolution-based, ten 2D-Convolutional Layers, one Flat Layer and four Dense Layers model. When compared with other Deep Learning based models, our model seems to have outperformed all of the aforementioned standard models by a substantial margin. The work presented through this paper can be realized to build vehicles capable of autonomous steering and driving. Additional training data of real-world obstacles with different track situations and conditions may be required to increase the agility and robustness of the system.

6. Future Scope

Through this project, we aimed to provide proof of concept for self-driving cars that can solely rely on vision-based object detection techniques for navigation, rather than the conventional feature extraction-based lane detection techniques. Results obtained on our model car made it clear that our approach towards object detection as a means of steering has either outclassed or is at-par with humans in the parameters being tested for. Reinforcement learning methods can be introduced in addition to this method to better performance. This method can be used as a prototype for future citywide self-driving cars projects. It can also be used exclusively, or in addition to conventional lane detection, to further improve on accuracy of self-driving cars. Via these techniques, automobiles might truly serve as end-to-end personal transportation devices and may give rise to an entire ecosystem of car-pooling or car sharing services as well as numerous start-ups thereby making personal transport cheaper, faster and safer. However, when implementing in the real world, many more parameters might be introduced which may increase the complexity of such a system while affecting the performance of the car.

REFERENCES

[1] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," IEEE Transactions on Robotics, vol. 30, no. 1, pp. 177–187, 2014.

[2] M. Tipping, M. Hatton, and R. Herbrich, "Racing line optimization," in US Patent, March 2013.

[3] L. Cardamone, D. Loiacono, P. Lanzi, and A. Bardelli, "Searching for the optimal racing line using genetic algorithms," in Computational Intelligence and Games (CIG), August 2010.

[4] K. Kritayakirana and J. C. Gerdes, "Using the centre of percussion to design a steering controller for an autonomous race car," Vehicle System Dynamics, vol. 50, no. sup1, pp. 33–51, 2012.

[5] H. Fujiyoshi, T. Hirakawa, and T. Yamashita, "Deep learning-based image recognition for autonomous driving," IATSS Research. Elsevier B.V., Dec. 2019, doi: 10.1016/j.iatssr.2019.11.008.

[6] darrenl, (2015) LabelImg (Version Window_v1.8.0) [Source code]. https://github.com/tzutalin/labelImg

[7] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," Nov. 2018.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, Jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[9] R. Kulkarni, S. Dhavalikar, and S. Bangar, "Traffic Light Detection and Recognition for Self Driving Cars Using Deep Learning," Proc. - 2018 4th Int. Conf. Comput. Commun. Control Autom. ICCUBEA 2018, pp. 1–4, 2019, doi: 10.1109/ICCUBEA.2018.8697819.

[10] A. K. Jain, "Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino," in Proceedings of the 2nd International Conference on Electronics, Communication and Aerospace Technology, ICECA 2018, Sep. 2018, pp. 1630–1635, doi: 10.1109/ICECA.2018.8474620.

[11] J. Kim, G. Lim, Y. Kim, B. Kim, and C. Bae, "Deep Learning Algorithm using Virtual Environment Data for Self-driving Car," in 1st International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2019, Mar. 2019, pp. 444–448, doi: 10.1109/ICAIIC.2019.8669037.

[12] Y. Kang, H. Yin, and C. Berger, "Test Your Self-Driving Algorithm: An Overview of Publicly Available Driving Datasets and Virtual Testing Environments," IEEE Trans. Intell. Veh., vol. 4, no. 2, pp. 171–185, Mar. 2019, doi: 10.1109/tiv.2018.2886678.

[13] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," 2018, pp. 621–635.

[14] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," Nov. 2017.

[15] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié, and C. Guionneau, "TORCS: The open racing car simulator," 2015.