Development and Implementation of an Autonomously Driven Vehicle Prototype

Luis Bill, Hamid Shahnasser
School of Engineering
San Francisco State University
San Francisco, California
e-mail: pototo@mail.sfsu.edu

Abstract—**Technology has become cheaper and more accessible, giving way to opportunities in technological areas that were previously inaccessible. Moreover, building a fully functional self-driving vehicle has become the new gold rush in the current industries. Yet, there are many issues to solve when building an autonomous vehicle. For instance, a self-driving vehicle must be reliable, consistent, predictable, and safe. There is still substantial amount of research and work that needs to be done, and more to learn, for the industry to build a truly reliable self-driving vehicle. In this paper the basic concepts of self-driving vehicles as well as some advanced concepts have been discussed. This project explores building a self-driving car using an off-the-shelf remote-controlled car, an NVIDIA Jetson TX2 GPU which has an ARM processor on board, infra-red sensors for obstacle avoidance, and a monocular CSI camera for navigation and object detection. A combination of Google's TensorFlow, OpenCV and a Robot Operating System (ROS) enables the remote-controlled car into becoming a fully self-driving car.**

Keywords-Tensorflow; graphical processing unit; robot operating system; infra-red; convolutional neural network; arduino; single-shot detector; remote control; NVIDIA jetson; camera serial interface

I. Introduction

The development of self-driving cars is a task that have the potential of solving many fundamental problems that drivers face when on the road. Reduction in vehicle accidents (and other transportation mediums) is probably the number one consequence of developing self-driving vehicles. Additionally, eliminating (or reducing) traffic will effectively increase the quality of living of people. Yet, achieving full and reliable autonomy requires using not only capable hardware to perform quick data analysis, but also software that can manage and make sense of the data that self-driving vehicles need and that the self-driving vehicles acquire. Many researchers approach the problem in different ways. M. S. Dihkle and S. Shahnasser [1] utilized a camera and an Arduino to build a surveillance robot. But more robust hardware than just and Arduino is necessary to compute more information and perceive better the environment. Part of the hardware required for full autonomy involves using high resolution cameras, radar to measure distance to obstacles, LIDAR, and ultrasonic sensors. Yet, more capable and ample hardware is needed for full autonomy and simple embedded systems, such as microcontrollers. Instruments, such as high-resolution cameras, acquire vast amounts of data. This data must then be analyzed fast enough so that the vehicle can make quick decisions while the vehicle moves. In this case, the high throughput makes a GPU a great computational device for the self-driving vehicle problem. Moreover, on the software side, Deep Neural Networks are one of the popular methods utilized to develop the software requirements. Machine Learning (and specifically Deep Neural Networks) help with the constraint that the real-world environment is dynamic and cannot always be predicted (such as pedestrian activity and intention prediction). H. Qu, T. Yuan, Z. Sheng and Y. Zhang [2] did important work using Retinex and YOLOv3 to detect pedestrians. Retinex and YOLO are Deep Neural Network algorithms that need many data samples to perform well. Moreover, SSD + MobileNets ([3] and [4]) was a better alternative in our experiments because of the mobile device focus of the algorithms. In the next sections this paper describes the proposed system, including hardware and software descriptions. Then this paper ends with a conclusion and remarks.

II. Proposed System

In this project, a cost-effective system is developed that uses low cost, of the shelf hardware components, as well as open source software frameworks, such as ROS (Robot Operating System) and Google's TensorFlow. The flexibility allows for the development of several software components.

The proposed system needs have real time response for obstacle avoidance and motor's speed control. Motor control and proximity sensors (IR sensors) are part of the critical system of the system. Real-time is necessary because the computer needs to react very fast in order keep the robot safe. The Jetson TX2 board used is running an operating system making it non suitable for real time operations. Therefore, the Jetson TX2 is interfacing with an Arduino, and the Arduino is in charge real time interface with the robot's motors.

Moreover, the proposed system runs a series of parallel processes (e.g., the motor control program and sensor interface programs are two separate processes running in parallel). If the all programs controlling the various functions of the RC car run in serial (or inside a single process), then the execution time for all the functions will be very slow. So, a way to run all the functions in parallel was necessary if correct functionality of the robot was desired. In this case, running multiple processes and allowing the operating system to handle all the memory management was decided. Moreover, to allow communication between processes, a network was uses where all processes can connect to in order to share information with each other. This is where ROS comes into play. ROS creates a network that is based on TCP/IP for multiprocessor communication.

III. HARDWARE COMPONENTS

The current hardware constains two types of hardware devices: Devices that ACT on inputs such as motors, and the RC car; and Devices that SENSE inputs such as the IR sensors sensors and the CSI (Camera Serial Interface) camera.

Figure 1. Maisto RC car.

This system customs various hardware platforms like Arduino, NVIDIA Jetson TX2 board, IR sensors, and a Maisto RC car. Listed below is the description of the Hardware components used in this project.

A. Maisto RC car

The main test platform used was the Maisto RC rock crawler (see Fig. 1). This is a hobby-level RC car, which only costs $30. This RC car has three motors: one motor to steer the front wheels, one motor to make the back wheels rotate forwards and backwards, and a motor to rotate the front wheels forwards and backwards. In this case, the two motors controlling direction and speed are connected to the same motor controller port and controlled by the same Arduino PWM port. The steering motor is connected by itself to the other motor port of the motor controller and controlled by two different PWM Arduino ports. Four PWM ports are needed, because two ports control forward and backwards, and another two ports control left and right turning.

B. Arduino UNO

The Arduino is a well-known product. Due to its ease of use not only engineers use, but also the non-tech savvy, such as artists, use it for their projects. Even though the Arduino boards are mainly made to be used with for hobby projects, its capability and robustness made it a grate candidate to tackle the real-time constrains in this project. As mentioned before, the Jetson TX2 cannot provide a real-time interface because the Jetson works with Linux as its operating system. Running an operating system does not allow the embedded platform to have a deterministic behavior. Therefore, functionalities, such as the motor speed of the RC car could not be controlled efficiently. Since the Arduino does not run an operating system, then that means that it can be used in real-time, and therefore its behavior is deterministic. The Arduino oversees controlling the critical aspects of the self-driving car. In this case, the Arduino oversees control of the motors (speed, direction, and heading), and it is in charge of interfacing with the IR sensors. The Arduino communicates with the Jetson TX2 via the USB ports. The Jetson TX2 can send motion commands to the Arduino, and the Arduino can send status notifications to the Jetson (such as current motors speed and heading).

C. Infra-Red Sensors for Obstacle Avoidance

The self-driving car built contains an array of Infra-Red sensors used in order to detect obstacles that are too close the robot. The robot uses three 2Y0A21 F 43 SHARP IR sensors set 90 degrees apart from each other (one sensor on the left side, a sensor on the front center, and a sensor on the right side).

D. Jetson TX2 Board

The Jetson TX2 is an embedded computer that contains an NVIDIA Denver2 plus ARM Cortex-A57 processor combination. Moreover, it contains a Pascal architecture GPU with 256 cores and 8GB of memory in the GPU. The Jetson TX2 is a credit card size super computer build mainly with the goal of deploying computer vision and machine learning technology. The Jetson TX2 is great for this self-driving car project not only due to its 1TFLOPS of computing performance, but also due to its low power consumption of 7.5 watts. For this project the development kit version of the TX2 was used, which comes already equipped with many useful peripherals and components. Some of these components include wireless connectivity, USB 3.0 ports, GPIOs, and a CSI color camera (the TX2 can handle up to six of these CSI cameras).

IV. SOTWARE COMPONENTS

For this project, frameworks that allow for development of self-contained processes was kept in mind. Also, open source software was an important tool. In this project, the following software components were used.

A. Robot Operating System

ROS (Robot Operating System) is a framework suit that allows robotics to accelerate the development of robots by eliminating the necessity of developing all software from scratch. ROS frame is a sudo-operating system that runs on top of a regular operating system. ROS contains a suit of libraries that allows for quick interface with sensors and other open source libraries. Moreover, ROS is developed with a service-oriented architecture, as well as a publisher-subscriber architecture. ROS works by encapsulating functionalities as standalone processes running in a computer, and these processes can communicate with each other over a network using TCP/IP. For instance, in the case of the car, the program interfacing with the CSI camera runs as a standalone process, but it sends images to the object detection module by publishing camera images over TCP/IP. In this case, the camera module is the publisher, and the object detection module is the subscriber. The self-diving car was built based on separate modules using the publisher-subscribing architecture. This architecture allows all modules of the self-driving car to run independently from each. This means that if one module fails it will not affect (nor kill) the rest of the modules running in the Jetson TX2 (see Fig. 2).

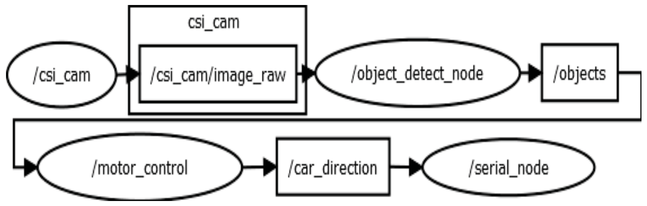


Figure 2. ROS graph of the self-driving car.

B. *Object Detection ROS Node*

As the name implies, this node oversees detection of objects. The object detection_node subscribes to the camera topic, then using a CovNet in can detect objects in every frame. The object detection module uses Deep Learning algorithm that combines SSD + MobileNets [5]. This object detection algorithm was developed to avoid a high computation of resources. Because of this lightness, SSD + MobileNets can run in real-time in a mobile device or in an embedded platform, such as the Jetson TX2. Fig. 3 shows the composition of the object detection node. The object_detect_node has the ability of no only detecting a person, but also can track a person. The object_detection_node uses a simple algorithm to track. The tracking algorithm is depended on the area of the bounding box surrounding the detected person. After a person has been detected, the object_detection_node publishes the /objects topic, which contains the objects bounding box with, height, and the bounding box's (x, y) coordinates, as well as the current camera image resolution. The resolution of the image is important because that allows the tracking algorithm to set the tracking thresholds relative to the image size instead of hardcoding thresholds. In this case, the motor_control ROS node is subscribed to the /objects ROS topic, and it uses this information to then make the decision of whether and where the robot should move. The main goal is to always attempt to center the person being tracked. Once the motor_contol has figured out in which direction the self-driving car can move, then it publishes the direction of the self-driving car using through the /car_direction topic. The serial_node is subscribed to the /car_direction topic. The serial_node is the ROS module that interfaces with the Arduino through the USB port of the Jetson TX2. The serial_node allows the Arduino to behave as if it was a software module. Once the Arduino obtains the information about the where the self-driving car shall move, then it sends the movement commands to the motors.

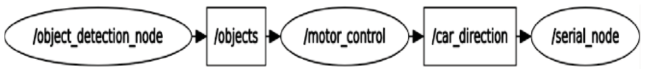
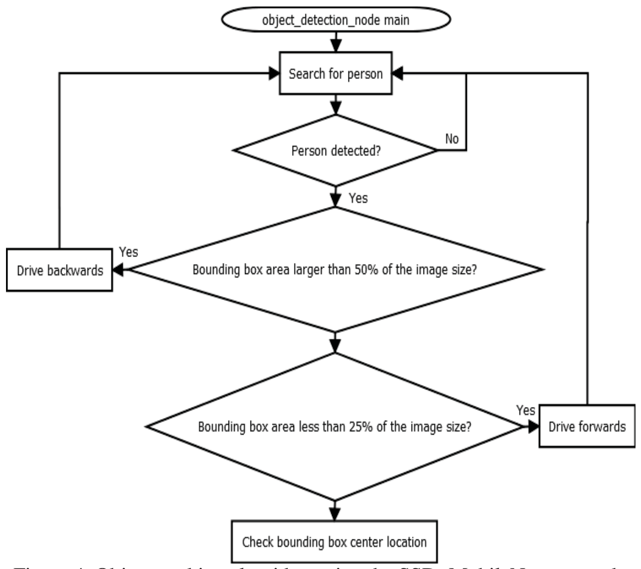
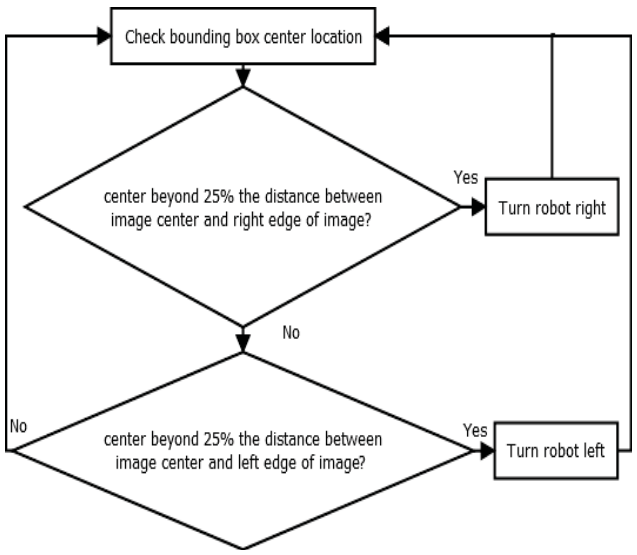


Figure 3. ROS graph of the object deteciton node.


object_detection_node main
Search for person
Person detected?
No
Yes
Drive backwards
Yes
Bounding box area larger than 50% of the image size?
Bounding box area less than 25% of the image size?
Yes
Drive forwards
Check bounding box center location
Figure 4. Object tracking algorithm using the SSD+MobileNets network.

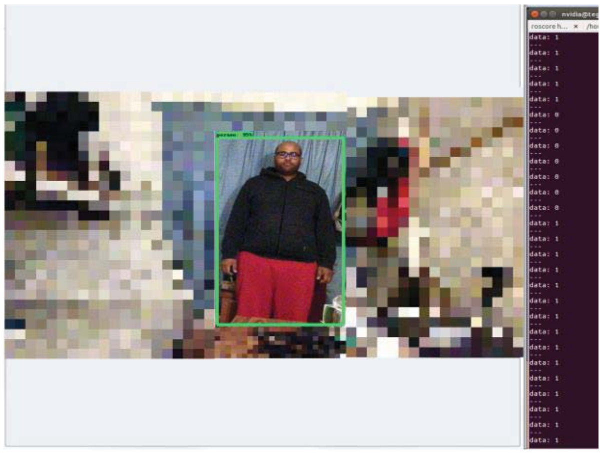

Check bounding box center location
center beyond 25% the distance between image center and right edge of image?
Yes
Turn robot right
No
No
center beyond 25% the distance between image center and left edge of image?
Yes
Turn robot left
Figure 5. Object tracking algorithm using the SSD+MobileNets network (continuation).

C. TensorFlow

Tensorflow is a relatively new library which was created by google. It is an open source library created with the purpose of allowing the public to be able to create, test, and change neural network algorithms. It is a common Deep Learning platform that has been very popular since its inception in 2015. Tensorflow is the main Deep Learning library that is being used by the self-driving car project. The current SSD+MobileNets graph is loaded and interfaced with using Tensorflow. Tensorflow has APIs available in many languages, including Python, which is the language in which the object_detection_node was written in.

V. EXPERIMENTS AND RESULTS

In order to verify that the self-driving car can in fact follow a person after detecting the person, a set of experiments were conducted. The experiments consisted on standing on different places around the robot, allow detection to happen, then print out on the computer screen the motion state of the robot. The motion state of the robot is motion direction in which the robot shall move (backwards, forward, stop, left, right). The /car_direction (see Fig.3) topic from the motor_control node to serial_node (Arduino) where coded with numbers from -1 to 3, and each number is mapped by the Arduino into a direction where the robot should move to maintain the person being tracked as centered as possible. The description of the car movements is mapped into a number-to-direction mapping scheme. For example, -1 means "Move car backwards," 0 means "Stop car," 1 means "Move car forwards," 2 means "Move car left," and the number 3 means "Move car right." The following figure shows the image of a person far away from the robot, as well as the print out of the self-driving car's motion state. Since the person is far away, the program outputs a "1," which means the cars state is "Move car forwards" towards the person. The state is decided by the car itself to keep a person as centered as possible, and the keep the person at a minimum distance from the car.



```
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 0
---
data: 0
---
data: 0
---
data: 0
---
data: 0
---
data: 0
---
data: 0
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
---
data: 1
```

Figure 6. Person tracking using SSD+MobileNets network, with command output.

VI. CONCLUSION

This project combines many of the modern hardware components and technologies that are part of the cutting edge in the Machine Learning industry and self-driving vehicles. The autonomous driving platform used in the experiments allowed for the chance of using a modern Deep Learning framework, such as TensorFlow, while at the same time interfacing with a GPU. Moreover, the utilization of multiple software frameworks, such as TensorFlow, OpenCV and ROS demonstrates how leveraging the power and accesibility of open source software can help further advance the field through software and algorithm sharing. Moverver, SSD+MobileNetsv1 have proven to be a well formed framework for object detetection and recognition. Future work will focus on testing different other algorithms, such as YOLO, SSD+MobileNetsv2 (or even MobileNets+YOLO) and compare them in order to understand which other algorithms can also be suitable for real-time applications in autonomous vehicles.

Many other experiments can further be performed, and many updates can be added as part of future improvements as well. For instance, the infrared sensors may be replaced by sonars, because IR sensors can fail in outdoors condition. What's more, a LIDAR can be used in order to obtain a wider obstacle detection angle, as well as to extend the detection range that IR sensors lack. Adding a LIDAR can also help in terms of adding localization capabilities to this self-driving car. A LIDAR could help in terms of mapping the environment to make the self-driving car's navigation safer. Yet weight constrains and a low budged are constrains that need to be solved before a LIDAR component can be added to this self-driving car. On the other hand, as part of the navigation framework, a new node, which is dependent on the camera module, can be used for visual odometry so that speed and direction of the self-driving car can be computed.

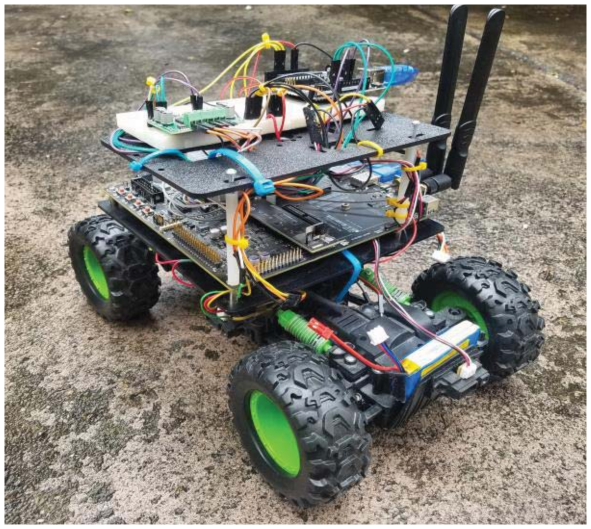


Figure 7. Person tracking using SSD+MobileNets network, with command output.

REFERENCES

[1] M. S. Dhikle and H. Shahnasser, "An Android Application Controlled Video Surveillance Vehicle," 2018 3rd International Conference for Convergence in Technology (I2CT), Pune, 2018, pp. 1-5.

[2] H. Qu, T. Yuan, Z. Sheng and Y. Zhang, "A Pedestrian Detection Method Based on YOLOv3 Model and Image Enhanced by Retinex," 2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Beijing, China, 2018, pp. 1-5.

[3] Howard, Andrew G., Zhu, Menglong, Chen, Bo, Kalenichenko, Dmitry, Wang, Weijun, Weyand, Tobias, Andreetto, Marco, and Adam, Hartwig. MobileNets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. Ssd: Single shot multibox detector. arXiv preprint arXiv:1512.02325, 2015.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow. org, 1, 2015.

[6] Y. Zhang, H. Peng, P. Hu, CS341 Final Report: Towards Real-time Detection and Camera Triggering. Stanford University, 2017.

[7] Abhineet Saxena, "Convolutional neural networks: an illustration in TensorFlow". ACM Crossroads 22(4): 56-58, 2016.

[8] D. L. Rosenband, "Inside Waymo's self-driving car: My favorite transistors," 2017 Symposium on VLSI Circuits, Kyoto, 2017, pp. C20-C22.

[9] J. Redmon A. Farhadi "Yolov3: An incremental improvement" arXiv 2018.

[10] M. Duong, T. Do and M. Le, "Navigating Self-Driving Vehicles Using Convolutional Neural Network," 2018 4th International Conference on Green Technology and Sustainable Development (GTSD), Ho Chi Minh City, 2018, pp. 607-610.

[11] Y. Tian, K. Pei, S. Jana and B. Ray, "DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars," 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), Gothenburg, 2018, pp. 303-314.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc. of CVPR, 2016.

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in arXiv:1801.04381v3, 2018.

[14] Mobilenetv1 source code. Available from github.com/tensorflow/models/tree/master/research/object_detection.

[15] TensorFlow Object Detector with ROS. github.com/osrf/tensorflow_object_detector/blob/master/README.md

[16] Supercharge your Computer Vision models with the TensorFlow Object Detection API. ai.googleblog.com/2017/06/supercharge-your-computer-vision-models.html.