The Utilization of Computer Vision in Autonomous Vehicles for Object Detection and Avoidance

Abraham Hadaf, Ameek Nijjar,
Jared Velasco, Samuel
Demissie

ABSTRACT

This project represents an object detection model to a NVIDIA Jetson TX2 device in order to achieve an RC car that can autonomously stop and go when a stop sign is detected using YOLOv5. Additionally, it avoids people in its path to avoid collisions while using a depth camera.

1. INTRODUCTION

The objective of this project is to implement object detection on a Traxxas RC car. We want to display the capabilities of object avoidance by means of autonomous car control using computer vision. The vehicle operates on a headless TX2 device and drives in a straight line until an "object" is spotted. The car avoids objects in its path by programming the car to make one 45 degree turn followed by a 90 degree in the opposite and then a final 45 degree to straighten out the path. We hope to impediment these movements to be performed within one meter away from the object using a depth camera. Additionally, the car should stop after detecting a stop sign in its path and continue to drive forward when it has waited for three seconds. The motivation for this project is to show the capabilities and usefulness of the object detection algorithm and how it can be used to help guide the cars that people use.

2. RELATED/PREVIOUS WORK

The study of autonomous vehicles has exploded in the past decade. A major contribution for this explosion is due to the advancement in processing power found in GPUs. Furthermore, GPUs have become increasingly available, this has opened up a wide range of computationally demanding projects. The previous work that we chose to base our project on is the Nvidia JetBot project [4]. The JetBot is an autonomous vehicle that is able to navigate a course (within lane lines), follow an object, and locate and stop at stop-signs. The JetBot has its pros and cons to its build and we plan to use the JetBot as a guide, not as a direct solution. JetBot works by collecting data, training a neural network, and deploying for real time use. We do not plan to use this method. Rather, we will be using YOLOv5 for our object detection and segmentation for our lane line determination. We chose to use these methods because the highly advanced YOLOv5 CNN is much more powerful than the standard resnet18 CNN they use to train their model.

To understand more about the ESC and how the receiver on the Traxxas works, we looked at JetsonHacks project called RACECAR. In this project, he goes into detail on how he was able to control the Traxxas using a controller. Another helpful topic he discusses is Pulse Width Modulation (PWM) [3]. JetsonHacks goes into detail about controlling the throttle and the servo and finally uses a transmitter to control the RC car. Although we have gained great insight and fundamentals about the ESC, servo, motors, transmitters and receivers, our project is slightly different because we need to develop an autonomous vehicle with object detection and avoidance.

3. METHODOLOGY

3.1 YOLOV5 IMPLEMENTATION

The term 'You Only Look Once' is abbreviated as YOLO. This is an algorithm for detecting and recognizing different objects in a photograph (in real-time). Object detection in YOLO is done as a regression problem, and the detected images' class probabilities are provided. Convolutional neural networks (CNN) are used in the YOLO algorithm to detect objects in real time. To detect objects, the algorithm only requires a single forward propagation through a neural network, as the name suggests. YOLO algorithm can be used in autonomous cars to detect objects around cars such as vehicles, people, and parking signals. Object detection in autonomous cars is done to avoid collision since no human driver is controlling the car. This means that a single algorithm run is used to predict the entire image. The CNN is used to predict multiple bounding boxes and class probabilities at the same time. There are several variations of the YOLO algorithm. Tiny YOLO and YOLOv3 are two popular examples but for this project we chose YOLOv5.

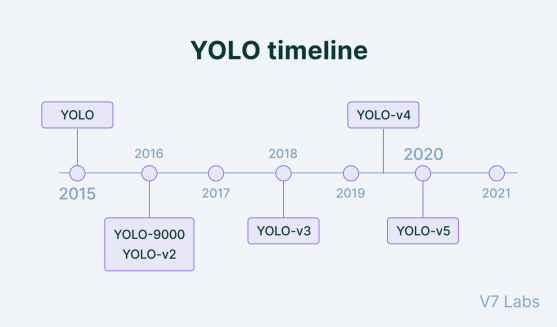

Figure 1

There are a number of differences and improvements in YOLOv5. The release of YOLOv5 includes five different models sizes: YOLOv5s (smallest), YOLOv5m, YOLOv5l, YOLOv5x (largest). This is the first native release of YOLO models written in PyTorch rather than PJ Reddie's Darknet. Darknet is a powerful research tool, but it was not designed for use in production environments. It has a much smaller user base. As a result of these factors, Darknet is more difficult to configure and less ready for production.

Because YOLOv5 is initially implemented in PyTorch, it benefits from the established PyTorch ecosystem: support and deployment are both simplified. Iterating on YOLOv5 may also be easier for the broader research community because it is a more well-known research framework. This also makes it easier to deploy to mobile

devices because the model can be easily compiled to ONNX and CoreML.

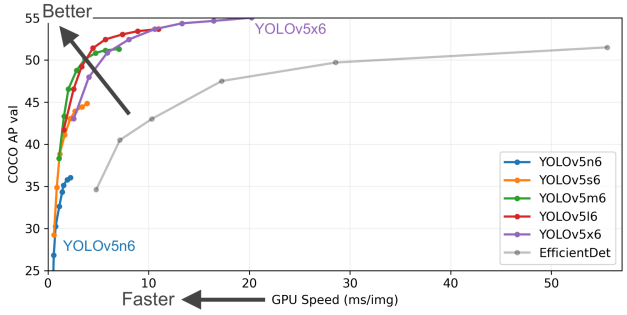

Figure 2

A weights file for YOLOv5 is 27 megabytes in size. Our YOLOv4 (Darknet architecture) weights file is 244 megabytes. YOLOv5 is nearly a third of the size of YOLOv4. This makes it much easier to deploy YOLOv5 to embedded devices.

3.2 BOUNDING BOX METHOD

YOLO uses a single bounding box regression to predict the height, width, center, and class of objects. Each of these bounding boxes is made up of five numbers: the x and y coordinates, the width, height, and confidence. The coordinates '(x, y)' denote the location of the predicted bounding box's center, while the width and height are fractions of the total image size. The IOU between the predicted bounding box and the actual bounding box, also known as the ground truth box, is represented by the confidence. The area of the intersection of the predicted and ground truth boxes divided by the area of the union of the same predicted and ground truth boxes is known as the IOU (Intersection Over Union).

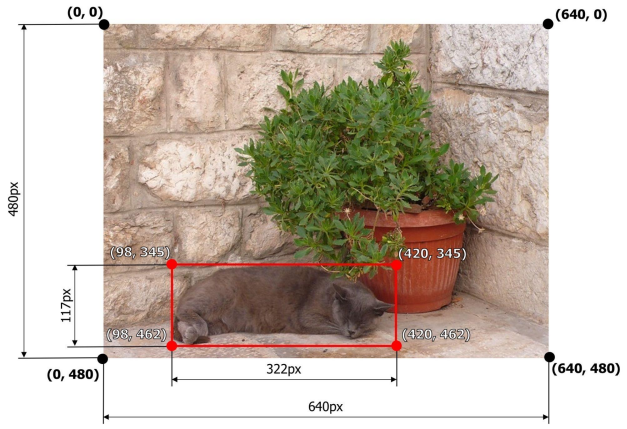

Figure 3

The bounding box values come from the pixels from the frame as denoted in figure 3. We can pinpoint the middle of the bounding box using the parameters xmin, xmax, ymin, ymax. Doing that, we can control the servo to the opposite direction where the person is located on the frame to avoid collision.

3.3 COMBINE YOLO AND PWM

The RC car comes with a servo and ESC which controls the throttle and steering of the car. We are able to control the steering and throttle using the PCA9685 PWM driver channels. The PCA9685 comes with many resources and modules for this project, such as JetsonHacks ServoKit and Adafruit modules. It's easy to use the PCA9685 sensor with Python or CircuitPython and the Adafruit CircuitPython PCA9685 module. This module allows you to easily write Python code that controls the servos and PWM with this breakout. Since our YOLO is built through python, we are able to control our servo and motor using the PCA9685.

3.4 INTEL D435 DEPTH SENSING CAMERA

The Intel D435 camera is a dual lens depth sensing camera that uses stereoscopic technology to find the distance from the camera to the object. The camera uses a known baseline value between the lenses, a focal length value from lens to object, and a disparity value found by subtracting object location on the x axis for the left lens minus the object location on the x axis for the right lens. The distance is found with the equation listed below:

$$Depth = \frac{(Baseline * Focal\ Length)}{(Disparity)}$$

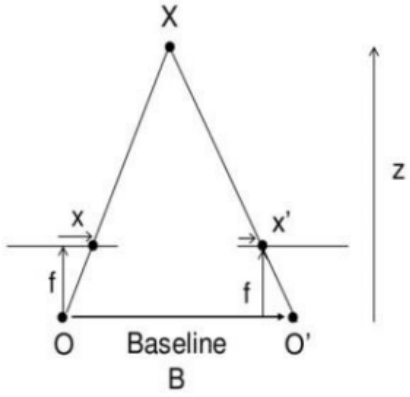

Figure 4

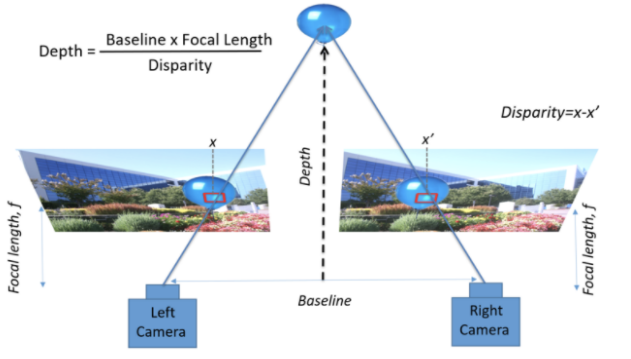

Figure 5

Figure 4 and 5 above show a modeled example of how the distance is found using the stereoscopic method described in the equation above.

3.5 PULSE WIDTH MODULATION

Pulse Width Modulation (PWM) is used to describe a signal input. As the signal modulates (turns off and on) the load receives an intermittent power supply. This allows designers the ability to customize the amount of power being supplied to the load. The total time the signal is on within a period of time is called the duty cycle. The equation for duty cycle is listed below:

$$\text{duty cycle} = \frac{t_{ON}}{t_{ON} + t_{OFF}}$$

Duty cycle can be used on loads like brushless motors or steering servos, as they were in this project. For example, if a brushless motor was desired to run at half its capability a duty cycle of 50% would be given to it by making the time on half of the total time period. See figure 6 for graphical representation of this example.

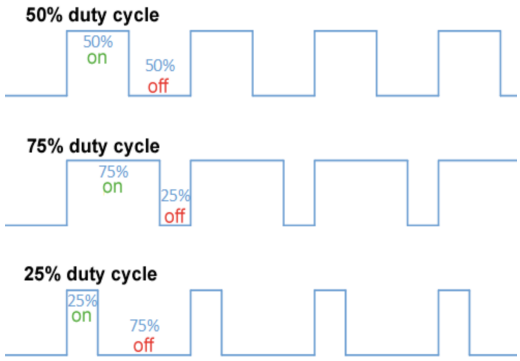

Figure 6

3.6 I2C COMMUNICATION

I2C Communication is a serial communication protocol that uses a controller and peripheral network to send and receive data. There are two links between the controller and peripheral, the SDA and SCL. The SCL is a clock control that synchronizes the peripheral to the controller. The SDA is a data linkage that sends and receives data between the two. Figure 7 below shows the link between two controllers and peripherals.

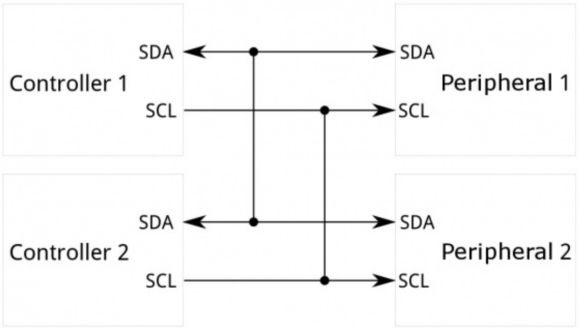

Figure 7

The SDA works by first switching from high to low voltage, signaling a START command. Then seven or ten bits are sent from controller to peripheral telling them where to store or where stored. This is followed by a read or write command, determining what to do with the sent info. If the peripheral receives the information correctly a ACK will be sent back to the controller, if not a NACK will be sent. Then the first eight bits will be sent and the cycle will continue until the message is complete.

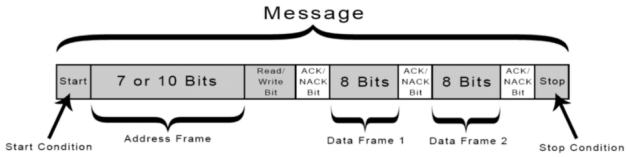

Figure 8

Figure 8 shows a representation of the sending and receiving from controller to peripheral via the SDA line.

4. RESULTS

4.1 STOP SIGN DETECTION

After the design of the object detection model and initialization of the servo kits with the RC car, we tested our code on a stop sign. Upon detecting the top sign, the car stopped immediately. However we did not account for distance. It is important that the car stopped less than 500mm away from the stop sign. To accomplish this goal we used the intel depth camera. This device simplifies distance calculation algorithms and thus an ideal choice for this project.

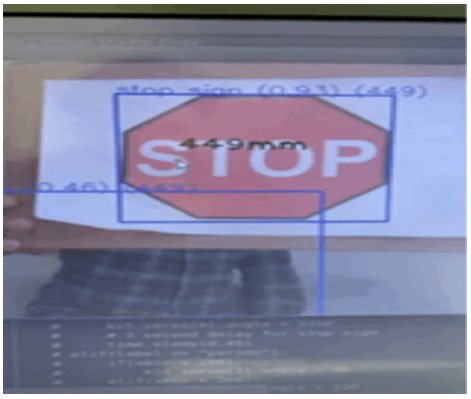

Figure 9

Fig. 9 This shows the detection of the stop sign with its confidence level. In addition, we have distance measurement displayed. As the stop sign is moved further back, the distance increases in reference from the depth camera.

4.2 HUMAN AVOIDANCE

Using the bounding box method, we achieved human avoidance. depending on the location of the person in the camera frame, the RC car will turn in the opposite direction of the person's location. Our model is only designed to work flawlessly with only one person captured by the camera. It is important to note that there is about one second delay for the RC car to react. This delay is clearly noticeable when the RC car is moving very slow.

Figure 10

RC car in motion and making turns to avoid colliding with a person. .

4.3 BENCHMARK

We received an average of 12 PFS. This is low due to the way we decided to set up our pipeline for the camera frame module. In order to have optimal PFS performance especially from a camera that is connected through a USB, it is important to use threading. This involves capturing the frames in one module while reading those frames in another module. Our pipeline used blocking operations to capture and then read the frames. We decided to implement blocking operations due to compatibility issues of the imutils library with Jetpack 4.6.

In terms of the hardware performance, we used four out of the six CPUs available each performing at 73%. We used the CPU resource for reading and capturing frames and distance calculation. The object detection model and servo kits used half of the GPU processing. Since our FPS was very low, we decided to have all the executable files and codes saved on the Jetson memory instead of an external hard drive to maintain optimal performance. The hardware data in figure 11 are statistics recorded without jetson clocks enabled.

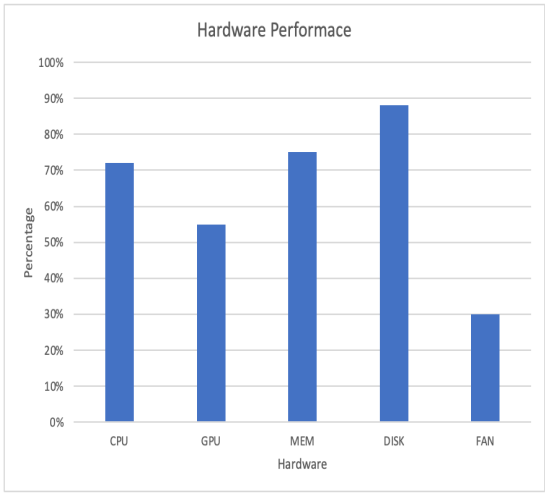

Figure 11

The total execution time to run our program averages to 19 seconds. Majority of the time is spent detecting the depth camera and initializing the servo kits.

4.4 REAL TIME DEMONSTRATION

Our final project demo consisted of the RC car throttling until it detected a stop sign at a close distance. The depth camera gave us the capability to calculate the distance of the stop sign as well as the distance for every object detected by YOLOv5. The car would then throttle again after a three second interval to model a real in person driving situation. To avoid a collision, the car would also turn left or right depending on where a person is located.. This is due to the bounding box method and calculating the location of the object in the frame.

5. DISCUSSION AND FUTURE WORK

The approach we took had many limitations. Our initial approach to controlling the motor and the servo through the Arduino was not successful. Although we were able to control the motor, we were not able to control its frequency. Apparently, Ardunino sends maximum frequency signals which makes the car throttle operate at its maximum. Our approach to solving this problem would be to use a potentiometer which acts as a resistance as to the amount of voltage supplied to the motor. However, this required new circuit design and more hardware resources. The biggest issue we had with the Arduino was controlling the servo. We wrote a script that would enable the servo to sweep its full rotation. However, the response time was very slow and the servo could only make half the sweep. In addition, after sending the sweep signal we noticed that the Arduno was disconnecting and our jetson was not detecting it. We realized that the servo was not getting enough voltage. We had connected the power supply of 5v from the Arduino to the servo. However, the servo operates at 6v. Since the arduino was powered by USB connection to the Jetson TX2, this sort of connection could have damaged our processors. Fortunately, the Arduino disconnection from the Jetson acted as a fail safe so that the processors do not burn out. Using the PWM driver board will make all Arduino complications much easier. The PWM board has its own enable line for frequency. This will make it easier to control the motor and servo at desired operation. It also eliminates the concern about voltage supply to each

component. Since it is connected to VCC, both components will have a common voltage supply base.

The system was only able to detect a stop sign and a person. If we had another three months, we would increase the number of applications that the car could perform. For example, stopping and accelerating at a stop light or avoiding multiple people in the frame. A major issue we had was arming the Electronic Speed Controller (ESC). Each time the code was executed, the ESC would disarm. We are not completely sure what was causing this, but we believe it is related to the voltage. Increasing the frames per second (FPS) is another area we would improve on given more time. The depth camera, despite giving the best results out of the 3 cameras tested, still only outputted around 10 to 12 FPS. We would like to increase this number so that there is less delay in the video input. A proposed solution would be to simplify the YOLO algorithm so that it is not as demanding on the Jetson TX2. This brings me to the final area of future work, which is utilizing a stronger, more powerful GPU. This would allow for faster processing speeds and allow the system to handle more data at once.

6. ACKNOWLEDGMENTS

Our deepest thank you to Professor Chen-Nee Chuah and Kartik Patwari for their guidance and sponsorship of this project.

7. REFERENCES

[1] Supeshala, Chamidu. "YOLO V4 or YOLO V5 or PP-YOLO?" Medium, 23 Aug. 2020, towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109.

[2] “Yolov5 And Vision AI .” Ultralytics, https://ultralytics.com/yolov5.

[3] JetsonHacks. “Jetson Racecar Archives.” JetsonHacks, 6 June 2017, https://www.jetsonhacks.com/category/robotics/jetson-racecar/.

[4] JetBot, https://jetbot.org/master/.

[5] https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python

8. APPENDIX

Ameek Nijjar and Samuel Demissie were on the hardware team and contributed to designing the architecture of the PWM, Servo, NVIDIA Jetson TX2, and ESC connections. They also contributed and worked hands on troubleshooting and calibration of the RC Car which involved throttle/servo control.

Abraham Hadaf and Jared Velasco were on the software team and contributed to many aspects involving the software of the project such as implementing YOLOv5, programming the depth camera and PCA9685 using Python. Additionally, they programmed the bounding box method and the car to stop when a stop sign was detected at a close distance.

Github: https://github.com/abrahamhadaf/Jetson-Autonomous-Car