Autonomous Road Following, Collision Detection, Traffic Light Detection, and Vehicle Detection in a Scale Model Using a Jetson Nano and ResNet-18 Modeling

Advisor:
Anas S. Eddin

Members:

Clayton Clark	Daniel E. Khoury	Ethan A. Mendez	Patrick M. Tejada	Weizhong Wang
cclark@cpp.edu	dekhoury@cpp.edu	eamendez@cpp.edu	pjtejada@cpp.edu	weizhongwang@cpp.edu

Date: May 14th, 2022

TABLE OF CONTENTS

***Abstract*—Autonomous technology has been at the forefront of innovation over the last few decades. The goal of this project is to make a self-sufficient, self-driving vehicle capable of road following, collision detection, and object detection using machine learning. Utilizing the computing power of the Nvidia Jetson Nano and a neural network based on ResNet-18, the group was able to successfully accomplish the goal. It is important to report that the success of a fully autonomous vehicle is heavily reliant on setting up a proper neural network that is trained for its specific use case. For this particular project it would be autonomous driving for a vehicle on a dedicated track. Proper data collection is crucial when creating a system like this and became the fundamental factor for this project's success. Important models like road following, collision detection, stop light detection, and vehicle detection each need to be properly trained using this data and cohesively implemented together in order to have the vehicle running smoothly. Lastly, due to hardware constraints and limitations, proper optimization for both vehicles are essential and proved to play an important role in overall efficiency.**

I. Introduction

This project uses two Jetson Nanos to construct two Jetbots that serve as the foundation for the project. The project uses NVIDIA's proprietary software to collect data and train all models [1]. Additionally, the construction of each Jetbot complied with NVIDIA's provided documentation [2]. Using NVIDIA's documentation as a baseline, the project endeavored to repurpose NVIDIA's software to implement core elements which simulate modern autonomous vehicles: road following, collision avoidance, bot following, and dynamic traffic light detection.

II. Road Following

A. Model Implementation

The model of the road following is a regression function. During operation, a flow of picture frames is used as the input and predicts the next location of driving. Using a PID controller, a two dimensional cartesian plane selects the coordinates of the next location and determines the required steering angle to reach its destination. Due to the camera delay caused by continual operation and heating throttling, this operation may cause "jittering" as the camera updates with delayed information.

B. Data Collection and Model Training

The dataset required the collection of two sets of 218 images showing the road and listing the desired destination. This was performed by taking an image and naming the file to reflect the desired x-y coordinate. Two sets of datasets were required because the cameras between robots had a different field of view which confused the model when copied into the second robot.

To train the model itself, transfer learning is used with a pre-trained model called ResNet-18 because it could later be optimized using TensorRT, which performs better on the Jetson Nano. Although another pre-trained model called AlexNet was provided, it performed slower than ResNet-18 with the TensorRT optimization. The initial model using ResNet-18 splits the data into training and testing datasets. The regression function used for training runs for 70 epochs and saves the best model at every epoch if the error, in comparison to the training dataset, is reduced.

C. Optimization

In order to optimize the road following model and reduce the effects of "jittering", the datasets took pictures from multiple different orientations to assist the model in behaving more consistently. Additionally, the desired x-y coordinate was changed to consistently be farther away from the robot in order to prevent multiple sharp angle changes due to delayed camera updates.

III. Collision Avoidance and Its Application for Object Detection

A. Introduction

The model of collision avoidance creates an output vector which is normalized to create a probability distribution. During operation, a flow of picture frames is used as the input and based on its dataset, it determines the likelihood of an object in its field of view. After implementation, it was determined that the collision avoidance modeling could be repurposed to detect the presence of single objects. Consequently, these repurposed models were developed simultaneously with the object detection engines in the event that unforeseen errors would occur with object detection.

B. Data Collection

The model training for collision avoidance required two sets of data. One database required photographs from the detected object, the other dataset required photographs where the object is not present. In practice, the data sets for collision avoidance, bot detection, and dynamic traffic light detection were as follows: Free/Blocked, Bot/NoBot, and Red/Green. Regarding collision avoidance, the Blocked dataset consisted of images where the robot would either collide with an object or fall off the edge if it continued traveling; the Free dataset consisted of images where the road is clear and the bot can travel freely. Regarding bot detection, the Bot dataset consisted of images when the rear of the robot was visible and the NoBot dataset consisted of various images where the front of the robot was visible or where the robot was not visible. Regarding dynamic traffic light detection, the Red dataset consisted of images when the traffic light was in the off/yellow/red stages and the robot would infer these inputs as a required stop. The Green dataset consisted of images where the traffic light was green and the road so the robot would infer the empty road and the green traffic light as traversable. Each dataset required a minimum of 100 images for modeling purposes.

C. Implementation

After formatting the two datasets into a single instance, The data collected needed to be formatted and filtered before loading onto the trainer. Once again, transfer learning is used with ResNet-18 and later optimized using TensorRT. The initial model using ResNet-18 splits the data into training and testing datasets. The regression function used for training runs for 30 epochs and saves the best model at every epoch if the error, in comparison to the training dataset, is reduced. Figures 1 and 2 depict the model training block diagram used for Road Following and Collision Avoidance-related models.

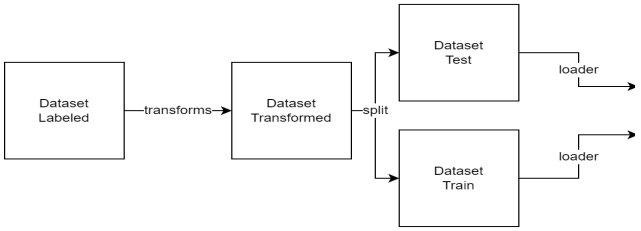

Fig. 1. Model Training Block Diagram, Part 1

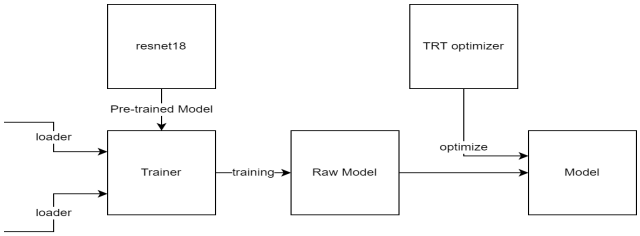

Fig. 2. Model Training Block Diagram, Part 2

IV. Attempted Implementation of the Object Detection

A. Introduction

Object Detection is a form of computer vision that is able to identify and outline different objects in an image using a neural network as the backbone. There are many object detection models that have been pre-trained to detect common items. One of the main datasets that these models get trained on is the Common Objects in Context (COCO) dataset. This dataset has over 200,000 pre-labeled images with about 80 different categories of objects in the set. The problem with using this dataset to train our model for object detection is that the object we are trying to detect is a Jetbot. The Jetbot does not completely resemble the same shape or features as the real world dataset for a car. The idea of creating a custom object detection dataset was necessary because of the uniqueness of the Jetbots features. Doing this should increase the accuracy of the detection as a form of the object "car" when a model is trained with this dataset. In order to train a model, the data needs to be manually annotated using a program called LabelImg. This program streamlines the annotation process by letting a user define an object's borders in an image manually and saves the metadata to an XML file to be accessed in association with the image. This larger annotated dataset must be separated into a training set and a testing set that it can be later used to train and validate training outcomes and adjust the values of the neural network.

The main type of model that was used was a Faster R-CNN neural network. The Faster R-CNN algorithm is an optimized region based convolutional neural network that was made to cut down on compute time and boost accuracy. The algorithm takes advantage of a Region Proposal Network (RPN) to pick what regions of the image are most likely to have the objects that are trying to be detected. The model is able to take on different network backbones to further optimize the performance or accuracy by sacrificing one for the other. The backbone neural network that was the best fit for our application was the MobileNetV3 for its lightweight design for mobile applications. This model later would need to become configured and optimized using TensorRT to run on the jetsons gpu without a performance hit.

B. Implementation and Results

A dataset of over 150 images was collected using the Jetbot's Raspberry Pi camera and annotated using LabelImg software. The Faster R-CNN model was pre-trained with the COCO dataset so that there was a good baseline for the model to work from when the project's dataset was added. Getting the model from pytorch to the tensorRT library format presented challenges with the versions of torchvision that were supported by the Jetbot software. Upon operation, it was observed that the Jetson Nano's performance was not sufficient to process object detection. The initial engine file resulted in a 5 second delay between updated frames. Creating a more optimized engine file and running it on a laptop with even more processing power resulted in a 0.4 second delay. The resulting delay on better hardware was sufficient to determine the integrated ObjectDetector class unacceptable for the project's standard. By this time, the replication of object detection using a regression function had been successful and efforts transitioned to optimizing these models to meet project standards. Object detection using the Faster R-CNN was still working and accurate, but did not meet the expectations on the performance evaluation.

V. Standards and Constraints

The project's final implementation must hold up to a high standard: Road following, Bot following and collision avoidance must always work before temperature throttling occurs. Additionally, dynamic traffic light detection must always work without human interference.

The high standard for road following required the reconstruction of the road network. This is because the initial road tiles and subsequent a glossy table top reflected the overhead light fixtures above the testing area. This caused the models to have difficulty differentiating between overhead lights and the middle line of the roadway. Consequently, the road network was reconstructed on a matte table surface which improved the performance and consistency of the robot to acceptable standards. Additionally, expanding the dataset for road following to include more edge cases improved the final system's consistency in all scenarios.

The constraint regarding the dynamic traffic light is primarily due to how the traffic light is constructed. Rather than having a separate panel to change the lights, the panel is attached to the traffic light. Consequently, human interaction with the light can confuse the model into thinking no traffic light is present. As a result, the constraints for the dynamic traffic light were changed to reflect the lack of human intervention in real-world traffic junctions. Lastly, the model was sensitive to the color red and sometimes misread red objects, such as shirts or redness in hands, creating false positives for stopping scenarios. As a result, a false positive detector was implemented to prevent false positives from impacting the functionality of the robot.

VI. Project Planning And Task Definition

A. Task Definition

The project was determined to integrate a minimum of four core elements: road following, collision avoidance, bot following, and dynamic traffic light detection. Road following referred to two robots independently following the middle line of a track and the ability to reorient itself if placed outside of the track's outer boundary. Collision avoidance attempts to replicate emergency braking found in modern commercial vehicles. If an object is detected in close proximity to the bot whereas further travel would result in a collision, the bot would be programmed to stop until the object is removed or until it can reorient itself in a manner which allows traversal. Bot following refers to when the two robots are in close proximity to each other whereas the leading robot prioritizes road following and the lagging bot simply follows the leading bot regardless of the track being present. Dynamic traffic light detection refers to the ability for the robot to detect whenever a traffic light is present and either continuing or stopping depending on the status of the light: green enables the robot to go, off/yellow/red prompts the robot to stop until the light changes to green.

B. Project Planning

The first stage of the project was to construct two functioning robots that could connect to WiFi routers and communicate via Jupiter Notebooks. This phase involved 3D printing a chassis, gathering all hardware materials (shown in the Appendix), and properly connecting all electronic components to the chassis. After an internet connection was established, all basic functionalities of the robots were tested including the camera, independent motors, and timing operations.

Stage two involved data collection to create functional models for road following and collision avoidance. This stage involved creating a consistent track and understanding the fundamental requirements for related datasets and model training which would be the foundation for the next stage of the project.

Stage three involves the optimization of road following and collision avoidance in addition to the implementation of bot following and dynamic traffic light detection. This was expected to be the most time consuming process because it would require in-depth understanding of object detection and our own implementation method for the robot.

The final stage involved optimizing the core elements of the project with the possibility of adding optional features. These optional features would not be possible in the end due to continuous obstacles in every prior stage.

VII. Conclusion

Autonomous technology is the next step in society's future endeavors. The goal for this project was to make a vehicle utilizing machine learning and capable of road following, collision detection, and object detection. Utilizing the Nvidia Jetson Nano and a convolutional neural network called ResNet-18, the project's goal has been fulfilled. Continuing to deepen the layers on a neural network allows for more complex tasks to be performed while also improving the accuracy of classification and recognition of objects. This neural network, ResNet-18, is an 18-layer deep network that returns state of the art image classification and recognition. To further optimize, TensorRT was used to populate the network using the ONNX parser. Following that, the builder was used to generate an engine that is optimized for our task. The success of the project heavily relied on properly establishing this network and training it to accomplish the project's tasks. Data collection was a crucial task that was done on a dedicated track to ensure consistency when training the model. However, the project encountered unforeseen constraints that altered the project's pacing. For example, the initial road surface was too reflective to gather proper data to train the model; consequently, a hand made track was placed on a matte desk to circumvent this constraint. Using the regression function, the model ran for 70 epochs and saved the best model at every epoch. This reduced any errors that the training dataset once had. Road following, collision detection, stop light detection, and vehicle detection were constructed to run in unison to accomplish the project's tasks. Overall, this project has proven to its members that group cohesion is vital to creating a functioning system. By properly outlining the project plans and wanted results, the group was able to successfully divide a complex problem into manageable parts that allowed for efficient use of time and available resources. While this computer vision technology is still evolving, it is a revolutionary advancement that is currently being both studied and tested for real life scenarios, and in this case, autonomous driving.

References

[1] NVIDIA (2021) *Jetbot* [Source Code]. https://github.com/NVIDIA-AI-IOT/jetbot.

[2] "Jetbot," *JetBot*, 2021. [Online]. Available: https://jetbot.org/master/. [Accessed: 02-Nov-2021].

[3] S. R. Rath, "Custom object detection using Pytorch Faster RCNN," *DebuggerCafe*, 24-Oct-2021. [Online]. Available: https://debuggercafe.com/custom-object-detection-using-pytorch-faster-rcnn/. [Accessed: 11-Apr-2022].

[4] R. Gandhi, "R-CNN, fast R-CNN, Faster R-CNN, YOLO - object detection algorithms," *Medium*, 09-Jul-2018. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. [Accessed: 11-Apr-2022].

[5] Tzutalin, *LabelImg* [Source Code]. https://github.com/tzutalin/labelImg. [Accessed: 11-Apr-2022].

[6] A. F. Gad, "Faster R-CNN explained for Object Detection Tasks," *Paperspace Blog*, 09-Apr-2021. [Online]. Available: https://blog.paperspace.com/faster-r-cnn-explained-object-detection/. [Accessed: 08-May-2022].

[7] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of Representative Deep Neural Network Architectures." [Online]. Available: https://arxiv.org/pdf/1810.00736. [Accessed: 09-May-2022].

Appendix

A. List of Materials

Jetson Nano Developer Kit	x2
64Gb MicroSD Card	x2
DC Gearbox Motor - "TT Motor" - 200RPM - 3 to 6VDC	x4
DC Motor + Stepper FeatherWing Add-on For All Feather Boards	x2

2 port 5V/3A output, 10,000mAh Portable Battery	x2
Caster Ball	x2
Type A USB to MicroUSB right angle cable	x4
Adafruit PiOLED - 128x32 Monochrome OLED Add-on for Raspberry Pi	x2
Break-away 0.1" 2x36-pin strip right-angle male header	x1
LI-IMX219-MIPI-FF-NANO-H136 Camera	x1
Raspberry Pi Camera V2 + 160 FoV Lens Attachment	x1
Waveshare AC8265 Wireless NIC Module for Jetson Nano	x2
Thin White Wheel for TT DC Gearbox Motors - 65mm Diameter	x4
M2 x 8mm screw	x40
M3 x 25mm screw	x8
M3 nut	x8
Female to Female Jumper wires	x8

B. Video Demonstration

The demonstration of the operating system can be found at the following link: .