Enhancement of the functionality of Spartan Superway Transportation

by Machine learning Approaches

A Project Report
Presented to
The Faculty of the Computer Engineering Department

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Bachelor of Science in [Computer/Software] Engineering

By

Simran Bains
Abhishek Karnik
Surya Ram
Rohan Surana

05/2022

APPROVED FOR THE COLLEGE OF ENGINEERING

Dr. Stas Tiomkin, Project Advisor

Dr. Wencen Wu, Instructor

Dr. Rod Fatoohi, Computer Engineering Department Chair

ABSTRACT

Enhancement of the functionality of Spartan Superway Transportation by Machine Learning Approaches

By Simran Bains, Abhishek Karnik, Surya Ram, Rohan Surana

- Object and velocity detection between vehicles is critical to preventing collisions and crashes. If vehicles did not have the appropriate space/distance between each other, collisions would happen often, and injuries would be expected. Our goal is to develop a machine learning model to predict the vehicle's velocity in front using a camera.
- The robots (iRobot Create2 and Sphero) will be programmed using Python and Javascript to move in a straight line. The challenge here is that the robots would collide due to the lack of a detection system. Objects such as sensors or cameras are missing for the robot to detect any vehicles in front of the robot. The need for an object and velocity detection system is high so that all robots would not crash into each other.
- Our goal is to develop a neural network for object and velocity detection. It would first detect the object and then detect the object's velocity using the area of the object extracted from the second robot. We would be utilizing one iRobot Create 2, one Sphero Bolt robot, a phone camera, Tensorflow/OpenCV libraries, and YOLOv5. The robots will travel in a straight line programmed by us, with the iRobot having a constant velocity and the Sphero having varying velocities. Our phone will be used on top of the iRobot to follow the sphero in a hall. We will utilize the velocities generated from the sphero for our training and testing sets. Our project would be a success if we could identify the Sphero and predict the velocity of it.

Acknowledgments

- Thank you Professor Stas Tiomkin for the help and advice throughout the project.
- Thank you to the SSW team for setting up the tracks and letting us collect data with the help of team members Khoa Lam and Carlos Ortega.
- Thank you Lockheed Martin for funding us with equipment.
- Thank you CMPE Department for providing us with robots.

Table of Contents

References

Appendix

Appendix A - Github Code Link

Appendix B - Lockheed Report Link

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Project Goals and Objectives

In this project, we built a real-time perception module, utilizing advanced artificial intelligence and machine learning methods. Specifically, we developed the functionality to detect and follow robots in a straight line. We will be utilizing an iRobot Create2 Roomba which follows the Sphero Bolt. Our main objective is to find the Sphero Bolt and detect the velocities of the Sphero at instances. This is done by collecting data using these robots and building a software solution in Python to help make predictions.

1.2 Problem and Motivation

As of today, there are many problems with public transportation. Some of them include causing high carbon emissions in the environment and causing over congestion globally, which contributes to collision accidents. The world is very reliant on fossil fuels which we want to change.

Our original plan included working with the SPARTAN Superway group, https://www.spartansuperway.net/, which looks to establish an autonomous, climate-friendly, safe transportation system based on artificial intelligence. As of this semester, the engineers at SPARTAN Superway built a new and smaller track with one bogie compared to the larger track and one bogie they had last semester. This system runs on solar energy, which migrates from the existing fossil fuel resources. Our project will help decrease the congestion on the streets and help build a safer community. SPARTAN Superway also provides an efficient approach to public transportation and can become the primary mode of transportation in the near future.

We have since changed our data collection methods to using our own robots as the data collection methods at SSW were too inconsistent. However, our model developed will still be applicable for them to run with their own data once everything is developed on their side.

1.3 Project Application and Impact

We undertook this project under Spartan Superway as we were not only interested in the machine learning aspect of the project but also interested in the global, economic, environmental, and social impact it will have on the world. Globally, Spartan Superway can set a precedent for other areas using the entire project. Our model can help detect objects/bogies in front of one another. This model will be used in the transit ascendant network by helping people with transportation from region to region in the San Jose community.

We aim to build a software solution, integrating the above functionalities into an operational system for designing, training, and testing "autonomous, climate-positive, safe transportation, based on artificial intelligence. Academically and industry-wise, this software will be helpful for the design and analysis of other climate-friendly transportation systems beyond Spartan Superway. In addition, this model will also give us experience in the Machine Learning field. This experience can help us contribute to even bigger projects later in our careers.

Spartan Superway aims to provide a solar-powered automated rapid transit ascendant network (SPARTAN). This helps the environment greatly as we are taking the step to use more renewable energy resources. If we compare this to taking a bus, the rapid transit network helps clean the environment and greatly reduces pollution. This model would also not depend on fossil fuel sources at all. As confirmed by statistics taken from the U.S. Energy Information Administration website at *www.eia.gov*, approximately 26 percent of annual energy consumption in the United States comes from the transportation of people and goods. Furthermore, electricity will provide less than 1 percent of total transportation sector energy use in 2020, with petroleum, natural gas, and biofuels accounting for 99 percent of energy sources consumed for transportation. As mentioned, our project will take steps to improve these numbers and increase the number of renewable energy sources used for public transportation globally, thereby positively impacting the environment by reducing pollution from other energy sources.

This project will also have significant societal implications, especially relating to public safety and transport. Our object and velocity detection model will free up city streets and will not cause congestion in public. The model detects the Sphero robot moving and accurately identifies the velocities of the tracked Sphero. Doing so increases the safety of pedestrians and people all around the area. Our model also aims to focus on detecting real-world obstacles such as people. This adds to the safety aspect of the project as we aim to detect any type of danger using a camera around the bogie, ensuring that accidents involving individuals and objects on the rail tracks can be prevented.

Economically, the Spartan Superway vehicle will be able to replace public transport systems in urban areas, offering improvements in long-term operating costs and

accessibility. The primary economic benefit of our system will be the reduction of expenses related to energy sources. Because the bogies will be powered through solar panels, they will only require a one-time fee for the purchase and installation of solar panel equipment, thus eliminating daily costs related to fuel. Although these solar panels may require maintenance from time to time, these fees will be negligible in the long run, considering the sturdy design of the solar energy systems. Since the bogies will be automated, they will also effectively cut out costs related to paying drivers for public transport vehicles. Additionally, our transport network will be easily accessible and provide a convenient and environment-friendly low-cost ride. Compared to other public transportation such as buses and taxis, the rapid transit network seems like a much better alternative given the reasons stated above.

1.4 Project Results and Deliverables

Our final project result includes a fully functional neural network that can accurately detect a Sphero robot and the velocity of the Sphero.

1. Infrastructure for real-time multimodal (camera) data acquisition and processing for the perception module on an embedded device (Nvidia Xavier NX)
2. Infrastructure for training various AI/ML models, including server-based training.
3. Infrastructure for deployment and debugging of the models on the device.
4. Benchmarks for measuring the performance of different models and optimization.
5. Dashboard for monitoring and analysis of the results on an external PC
6. A software solution, integrating the above functionalities into an operational system for designing, training, and testing "autonomous, climate-positive, safe transportation, based on artificial intelligence.

This software will be helpful for the design and analysis of other climate-friendly transportation systems beyond Spartan Superway.

7. The results of the project will be published at a relevant conference as well as a formal paper.

Next Steps:

- Hardware Integration - Attach Raspberry Pi camera module and Nvidia Xavier NX computer hardware kit to bogie. *(Had to compromise by only attaching Raspberry Pi cam as bogie was not able to attach to the NX kit)*
- Go to the lab and collect new data using improved track and bogie designed by SPARTAN Superway mechanical engineers

- Development of Machine Learning Model - Use collected video data to train machine learning model. (Includes using YOLO to break down area of bogie to act as an identifier and development in python using training sets)
- Analysis of Results - Analyze the results of the machine learning model and tweak it to retrieve best results (Make sure bogie can be detected accurately and loops of track can be handled when bogies are traveling on track)
- Change method of data collection to use Sphero robot and iRobot Create 2 Roomba for more consistent data. Repeat steps of development in terms of YOLO detection and analysis of results.
- Create final operational solution
- Write Report on project and findings

1.5 Project Report Structure

The next sections of the document will introduce papers we researched and proposed technologies of the project. In addition to this, our requirements, design, and implementation will be presented below.

Chapter 2 Background and Related Work

2.1 Background and Used Technologies

This project involves background knowledge of Python and machine learning libraries such as TensorFlow or OpenCV. We have also changed our plan to utilize YOLO to estimate and identify the area of the bogie so the model knows what to track. Other external tools that we will be utilizing are the Nvidia Xavier NX Developer Kit and cameras to help run our neural network. Some of our group members have taken/are taking CMPE 188 - Machine Learning. Other than this course, our learning has been through Youtube video tutorials and Google.

2.2 Literature Search

The first scholarly article we researched is “Deep convolutional neural network for real-time object detection using tensor flow.” This article has reputable resources, all of which can be found from Google Scholar. Some papers are conference papers and are published by IEEE. Some other references do include peer-reviewed journals, which make this article reputable and trustworthy. This particular article has been published on ScienceDirect and is peer-reviewed. Our project relates to this article as we are also interested in using TensorFlow as a classification system for our machine learning model. The article’s researchers use TensorFlow as a classification system for a traffic signal detection system. Some external tools such as SoftMax are used as a classification system to identify features in images. Their project includes a pre-generated test set and a computational model to accurately identify objects in the original images. We aim to build a model similar to this and go through similar processes/steps as stated in the article (Import Dataset -> Normalize Data -> Set parameters -> Build model -> Train model -> Test Model -> Detect Object). From here, our model will be tested and trained in lab scenarios such as the one specified in this article.

The second scholarly article we researched is “Object Detection: Automatic License Plate Detection using Deep Learning and OpenCV.” This article was found from the SJSU library resource. All listed references in the articles are from reputable journals and have significant merit to them. The papers used as references are also within five years, which makes this article’s information current. This project also uses TensorFlow as their deep learning library. The researchers go through a similar process as the article mentioned above by capturing images, processing them, and using testing/training methods to achieve a successful output. This reference reflects on our group as our architecture looks similar to the one in this article, and we plan to follow some of the mentioned processes. We aim to use a camera to record video for our data and process the data so our model can accurately train the system. We can then crop the bogies and store them in the database for our model to read and identify from. We aim to achieve similar

results to the article provided. They achieved a success rate of 93% with their own collected dataset. OpenCV was also utilized for image processing which is another reason why we researched this article. Both these libraries are options for us, and we look to use them with our data collection to the best of our abilities.

The third research paper discusses the computing systems for Autonomous Driving. The research paper is titled "Computing Systems for Autonomous Driving: State-of-the-Art and Challenges." In this paper, the authors present the state-of-the-art computing systems for autonomous driving, including seven performance metrics and nine key technologies followed by twelve challenges to realize autonomous driving. According to the researchers, we must determine how to make the vehicle understand the environment correctly using rich sensors like camera, LiDAR (Light Detection and Ranging), Global Navigation Satellite System (GNSS), Inertial Measurement Unit (IMU), Radar, Sonar. The paper also summarizes key technologies/ tools used and their state-of-the-art, including cameras, Radar, LiDAR, Ultrasonic sensor, GPS/GNSS/IMU. The development of object detection algorithms typically goes through two phases: the conventional object-detection phase and the deep learning-supported object detection phase. They discuss the current state-of-the-art for computing and communication technologies.

The fourth research paper we researched applies heavily to our project. In the article, "Detecting objects and people and tracking movements in a video using tensorflow and deep learning," researchers present their own experience in detecting moving objects. The authors here generate an xml file if an object exists. They also code a python script which then converts this to a csv file. From here it is converted to a TFRecord format. For the image processing library, OpenCV is used (reading and annotating images). The Pandas framework is also used for a DataFrame which helps structure the data. The collected video is then segmented into parts to identify. Each image is then annotated using the labelIMG package. An xml file is generated for each which is then converted into a TRFRecord format to train the model. Once the training and validation set is complete, the test set is introduced to enhance the model.

The fifth research paper we analyzed is titled, "Real-time object detection based on YOLO-v2 for tiny vehicle object." This paper relates heavily towards this project as we decided to use YOLO to break down the area of the bogie from the video. This will help us greatly with identifying the bogie on the track. The authors describe how YOLO-v2 has an accurate predictability model where they can identify the position, size, and category of any object. The tiny vehicles they are trying to identify are difficult to accurately follow. Thus, the authors tend to use residual models added to the model to counter the fact of dispersion. Since YOLO uses rectangles to identify objects, we might have to use some other sort of identifier such as a shape attached to the bogie, so it will be easier to identify.

2.3 State-of-the-art Summary

There are several leading-edge tools and techniques available for the area of our project that we may use. Since our project aims to create a system that allows for object detection using a neural network, we will be required to utilize various machine learning techniques. We plan to collect data using a traditional machine learning technique in this field through a long video of the bogies traveling around the track as they are expected to function. A camera will be attached to the front of a bogie, which will travel around the track along with several other bogies. The video will depict how the system functions and illustrate how the bogie in front will appear when collecting data to process. This video will provide the necessary data for us to feed to our machine learning model, after which we can process the data and determine the optimal way to perform object detection. In order to build the neural network, we will likely be using bleeding-edge technologies, including python programming, OpenCV, and YOLO.

OpenCV is a library of computer vision-related programming functions that can be utilized in order to develop an object detection system. As stated earlier in the document, we have discussed using YOLO to break down the area of the robot and use this as an identifier for our model. We may also use TensorFlow. TensorFlow is open-source software that is widely used in machine learning applications such as our project. This software is one of the more recent technologies developed by Google to provide a modern and easy-to-use deep neural network learning platform. As stated from the literature review, all these libraries and tools will help us achieve our result. These leading-edge tools have been used in research previously, and researchers have achieved significant results with their machine learning model. Some of them even used their own generated data, which we will be doing for our project. We aim to follow these processes and use similar technologies. Using all of these technologies, we can train the machine learning model. This model, once trained, will be able to perform object detection tasks accurately and determine the velocity of the surrounding bogies.

References

1. Padmapriya, G., Santhosh Kumar, B., Kavitha, M. N., & Vennila, V. (2021). Deep convolutional neural network for real time object detection using tensor flow. *Materials Today: Proceedings*. https://doi.org/10.1016/j.matpr.2021.02.671

This article explains the process of using TensorFlow in a traffic-signal detection-based system. The authors explain the process of the Object Detection based project and the steps it takes to form a successful project from planning all the way to execution. The model accuracy resulted in a 91.10% test accuracy.

2. Vishal, R. M., Maram, D., Chaitanya, P. K., & Angeline, R. (2019). Object detection: Automatic license plate detection using Deep Learning and OpenCV. *International Journal of Engineering and Advanced Technology*, *9*(1), 6022–6028. https://doi.org/10.35940/ijeat.a1842.109119

This article uses TensorFlow and OpenCV to detect license plates on cars for various applications. The process includes an image dataset and multiple processing and testing steps to achieve a successful result. The license plates are loaded into the database as an example of what should be identified and their model returns a 93% success rate.

3. L. Liu et al., "Computing Systems for Autonomous Driving: State of the Art and Challenges," in IEEE Internet of Things Journal, vol. 8, no. 8, pp. 6469-6486, 15 April 15, 2021, doi: 10.1109/JIOT.2020.3043716.

This article discusses the current state-of-the-art for computing and communication technologies, which includes rich sensors like cameras, LiDAR (Light Detection and Ranging), Global Navigation Satellite System (GNSS), Inertial Measurement Unit (IMU), Radar, Sonar.

4. J. Bornia, A. Frihida and C. Claramunt, "Detecting objects and people and tracking movements in a video using tensorflow and deeplearning," *2020 4th International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*, 2020, pp. 213-218, doi: 10.1109/IC_ASET49463.2020.9318253.

This article discusses object tracking more in detail which relates to our project in the sense that we have to detect bogies moving around the track. The authors use both tensorflow and openCV in building their neural network. Their flow diagram is also similar to our data framework flow.

5. X. Han, J. Chang and K. Wang, “Real-time object detection based on YOLO-v2 for tiny vehicle object” *Science Direct*, 2021, pp. 61-72, doi: https://doi.org/10.1016/j.procs.2021.02.031.

This article discusses using YOLO to identify tiny vehicle objects and use external resources to improve the accuracy such as residual models. Our project relates to this as we will try to use YOLO to identify the area of the bogie and use an external resource such as a shape attached to make identification easier.

Chapter 3 Project Requirements

3.1 Domain and Business Requirements

For our project, we did an object and velocity detection model between two moving robots. Our project isn't class based such as most web based applications, thus we have presented process summary and decomposition diagrams below.

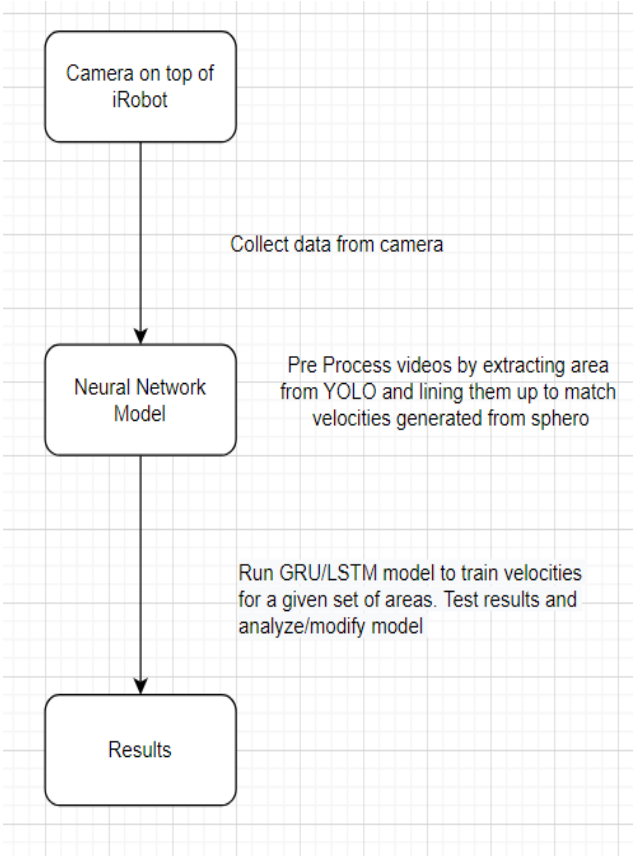


Fig 1. Process summary diagram - The bogie (iRobot) is placed on the floor and follows the Sphero in a straight line. We then train a YOLO model and extract area from frames of sphero to match the velocities at a given time. Our GRU/LSTM model will then predict the velocities given an area.

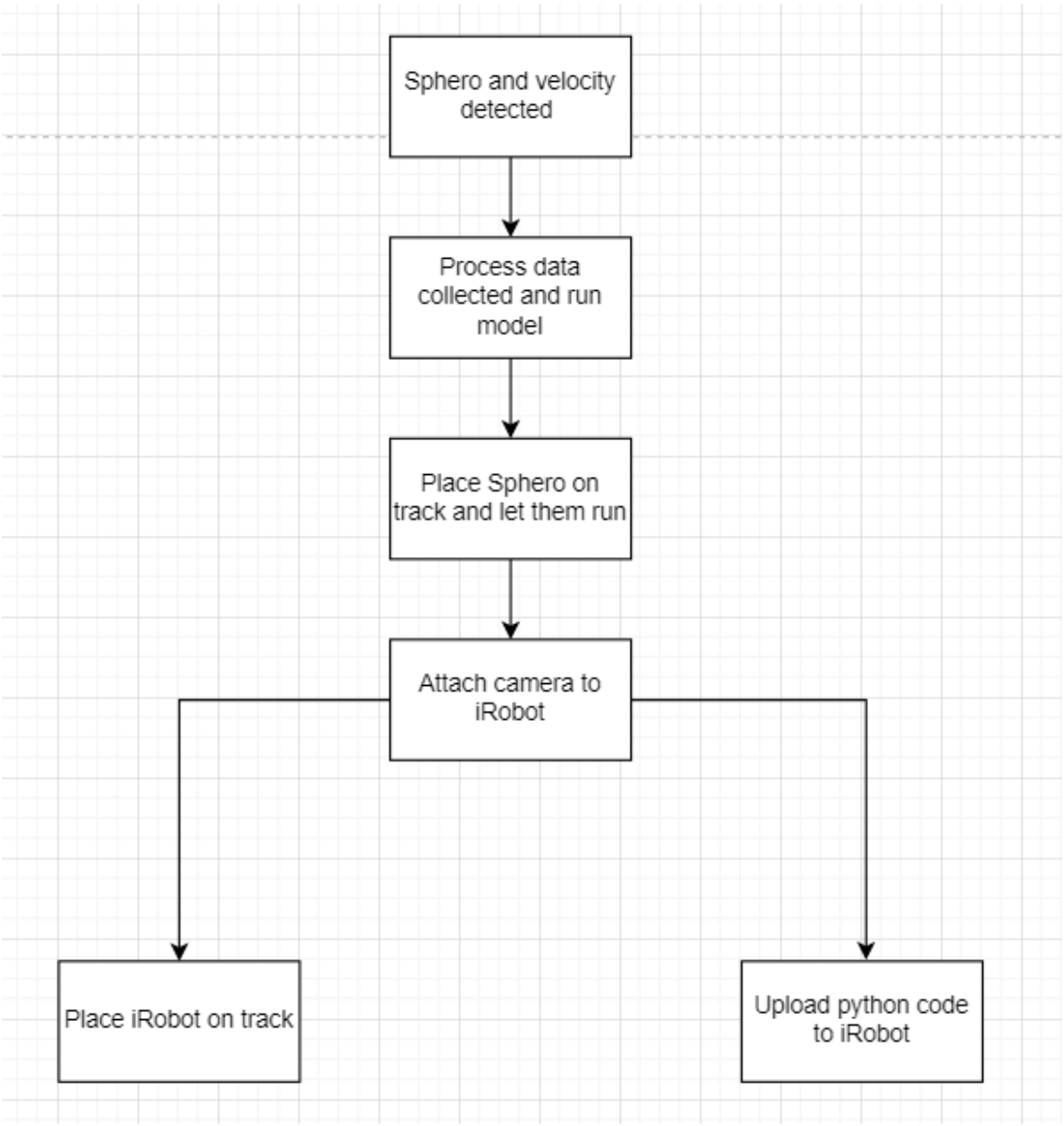


Fig 2. Process decomposition diagram - The iRobot is placed on the track and fitted with a camera. The next step is to load Python code to move the iRobot. Once the iRobot is set up, place a sphero robot on the track and let them run. Process data collected and run model to predict the object and its velocity.

3.2 System (or Component) Functional Requirements

- Our project requirements that shall deem a successful project include collecting data from the engineering hallway. This is a significant portion of our project, as, without any data, we would be unable to build and train a model that would detect the sphero robot. Gathering data is one of our essential features, and we shall do

this by having the iRobot follow the Sphero on a straight line and recording the video for about 5-6 hours.

- This brings us to the next step: implementing a model using Python and an image detection library such as YOLO to help us identify and train the model to detect the Sphero accurately. This is also an essential feature as our project revolves around delivering a lab-scale model of the robots, accurately detecting the Sphero in front, and determining the velocities of the tracked Sphero. This shall be tested using the model to verify the Sphero area and velocity.

3.3 Non-functional Requirements

- A non-functional desired requirement that we shall tackle is the model's reliability and produce at least a 80% accuracy. We would like to test this model in various scenarios on the track so that our model would be able to withstand and prevent any dangers that come about. We want our model to produce the same results for a given test and not fail due to some slight edge cases. This shall include having the robots run for multiple hours on a track. In addition to this, we shall need the robots to run in varying velocities according to a sine wave.
- A non-functional optional requirement we should attempt is to increase the response time of the model. We attempt to detect objects moving through a loop that doesn't have too much traffic between them. Optimizing the frames is a step we would like to pursue but is not as important as the requirements listed above. The frames per second we would like to achieve with the Xavier NX board is 30. This would also require a variety of other factors such as the processing power of the board which could hinder the performance.

3.4 Context and Interface Requirements

The scope of development, testing, and deployment of our project was initially supposed to be in the Spartan Superway lab. Unfortunately the data collected from this process was deemed unusable as their equipment was not set up properly. We had to shift our proces. The entire process takes place in an engineering hallway, and the result of our project will be used to further the Spartan Superway team's efforts in order to eventually build a full scale public transportation system. To begin with, the project we deploy will be used to perform object detection on the Sphero. Spartan Superway will have to make necessary adjustments to tweak the model to their liking. Our project will act as the baseline, as there is no such framework existing at the moment.

An interface requirement is that our model is run on the Xavier NX development kit. We will use the benchmark of this to see whether the model is performing up to our standards or expectations.

3.5 Technology and Resource Requirements

Hardware Requirements	Software Requirements
Jetson Xavier NX Kit	Compiler - VS Code, Jupyter Notebook, Colab Notebook
iRobot Create 2	Software - OpenCV, YOLO, Python, GRU/LSTM, Keras
Sphero Bolt	Sphero Edu Application
Raspberry Pi Camera 4 MB	
Phone camera	
Engineering Building hallway	

Table 1. Software and Hardware Requirements

- Our hardware components included the NX kit to be able to process our model. The iRobot Create 2 and Sphero bolt was arranged by our advisor as another option to collect data. The components selected include the Xavier NX kit which is used to process our machine learning model.
- Our software components include a compiler just to run our code. For now we chose VS Code. The Sphero Edu application is also needed to control the Sphero robot with javascript code. Other than this, we chose to use YOLO to break down the area of the Sphero and as an identifier. Python (OpenCV/TensorFlow/GRU/LSTM) is used for model development.

Chapter 4 System Design

4.1 Architecture Design

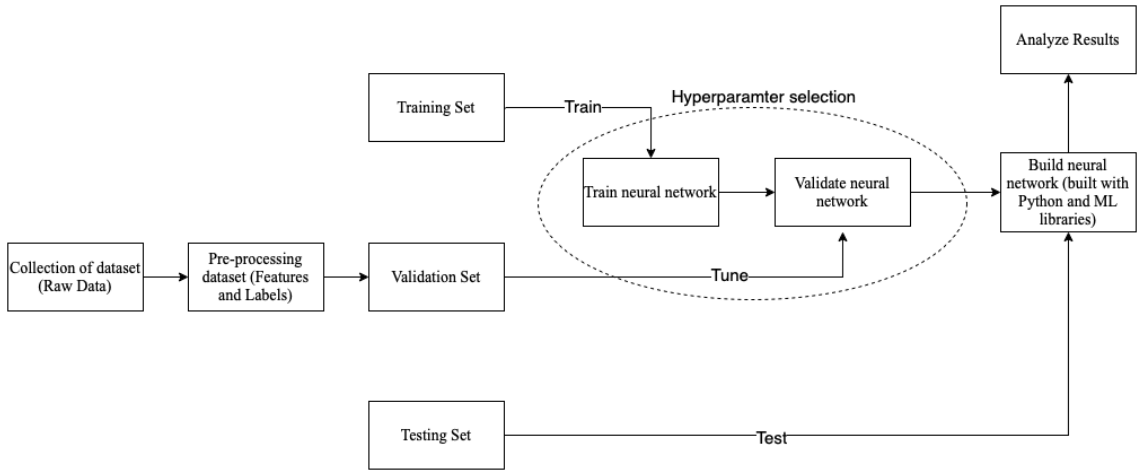


Fig. 3 Framework of Model - This diagram shows how the neural network will be built. We will collect a set of data and process/split them into 3 datasets as shown in the figure. From here we would train and validate the neural network using the training set and validation set. This would in turn be used to finally test using our testing set where we can analyze our results and fix any outliers we may have not considered.

4.2 Interface and Component Design

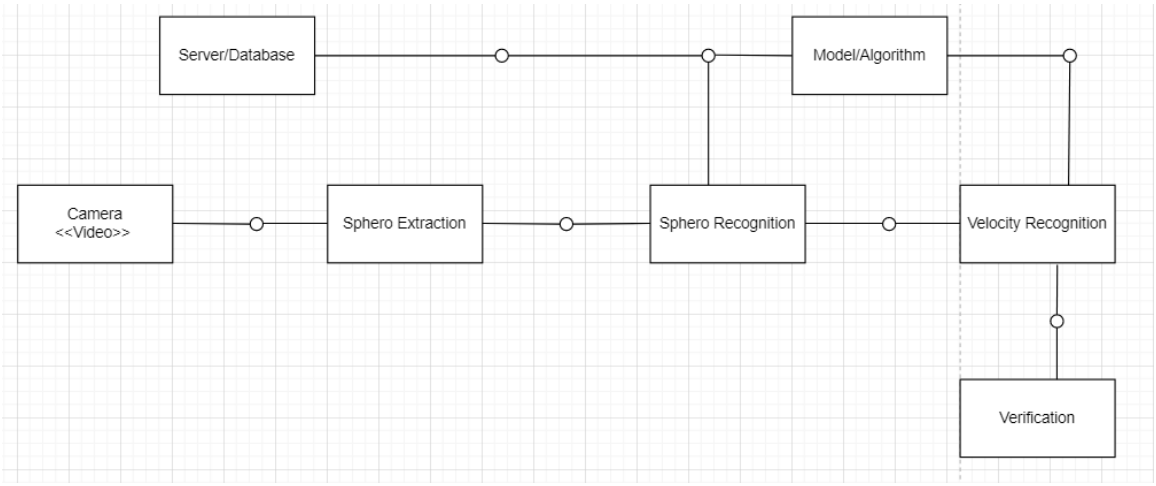


Fig 4. Component Diagram - Our Component Diagram isn't based on UML design, but rather on a flow of events. The model will be trained with data sets we have collected. From the trained model, the camera attached to the iRobot will try to identify the Sphero and velocity of the Sphero. From here the model will verify the results from previous trained sets.

4.3 Structure and Logic Design

Our logical diagram consists of the hardware component as this will be the main component used to run the model. Our setup of the hardware components is presented below in Figure 3 and will be the same structure identified from the first semester. Additionally, software components mostly include identifiers such as YOLO and the python programming language to identify the sphero.

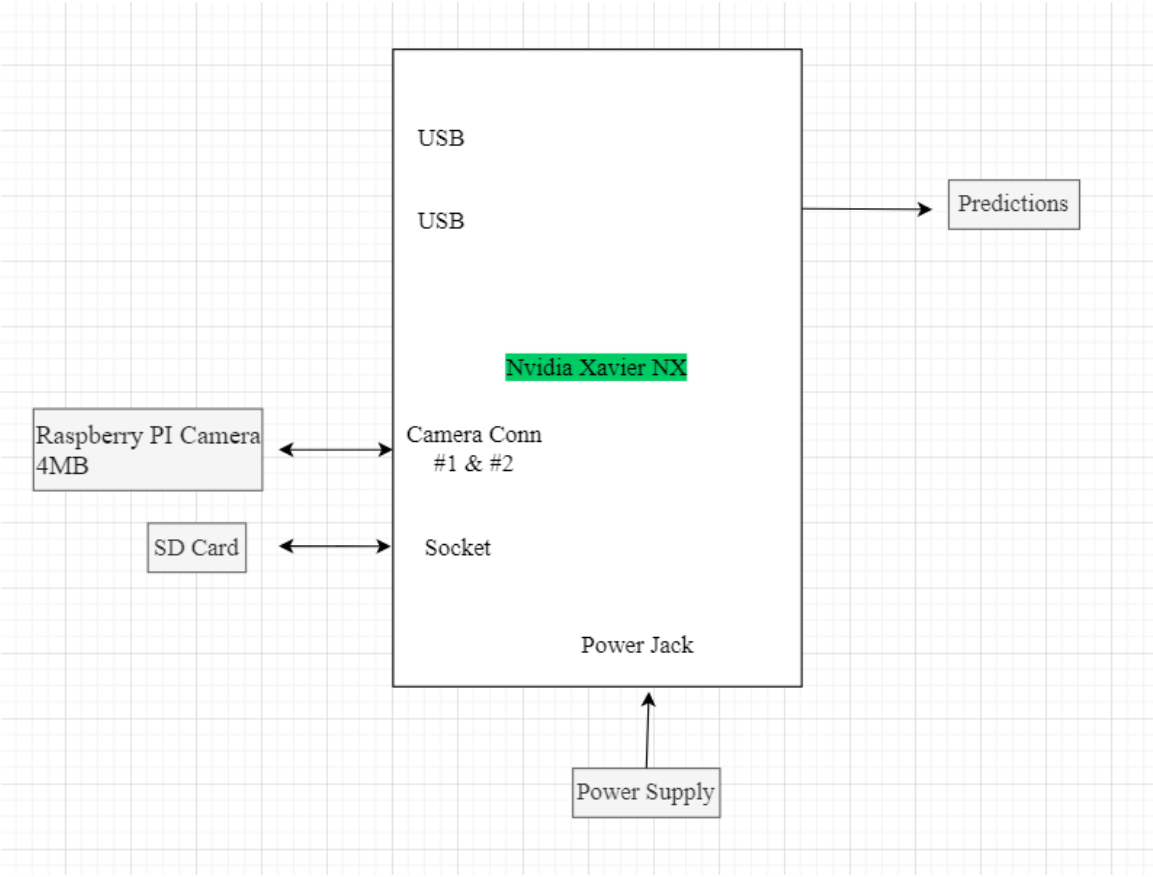


Fig. 5 Hardware Block Diagram - This is a model of the computer hardware kit we intend to use (Nvidia Xavier NX) along with the various components and connections to other hardware.

4.4 Design Constraints, Problems, Trade-offs, and Solution

Our problems and constraints for our machine learning model mostly lie within the hardware aspect of the project. Another problem we have encountered is that Spartan Superway is developing a new set of bogies and a smaller track, so we have to wait to get data. When we tried going to the lab, SSW only had one bogie available. We tried getting data by using one of their bogies and a Sphero robot on the track. The data collected was too random due to problems with the Sphero running on their track. A solution we have decided upon is to use 2 mobile robots (iRobot and Sphero) for data collection in order to train our neural network. The iRobot is programmed to move straight in a constant velocity behind, while the Sphero travels in front at a speed given by a sine wave function based on the constant velocity given below.

```
V_s(t) = V(t) * (1 + (0.1 * Math.sin(2 * Math.PI * t)))
```

4.4.1 Design Constraints and Challenges

- Our first problem lies within one of our main components of the project. One of our requirements is to acquire a small powerful computer kit to help run our neural network. For this, we have decided on 2 models - Nvidia Jetson Nano 4GB Developer Kit or the Nvidia NX Developer Kit. The Nvidia NX Developer Kit is the faster, more powerful kit between the 2. The kit is in stock for about $1300 plus we have funding from Lockheed Martin. In contrast, the Nvidia Jetson Nano 4GB Developer Kit is about $300 with everything included (power supply, SD card, and kit).
- Our second problem is related to the cost and quality of the cameras/track. The bogie being developed by the SSW team has one slot open for a Raspberry Pi Camera for data collection. We decided to buy a 4 MB Raspberry Pi Camera for this use. The track is also unstable which means the camera might be shaky.
- Another challenge of ours which we face this semester is with the bogies and tracks of SPARTAN Superway. SPARTAN Superway recently decided to improve their design of the track by making a smaller track as well as develop new bogies for this proposed track. Due to this we need to make changes in data collection as well as wait to break down the images of the bogies in YOLO.
- The final challenge was after processing the data collected from SSW. The data was not enough as the sphero was unable to consistently move on their track. This brought about finding a new solution.

4.4.2 Design Solutions and Trade-offs

- The NX kit isn't as price friendly as the Jetson Nano 4GB kit but will be efficient for our model to run on as discussed with members from Spartan Superway. The trade-off here would be to buy the Xavier NX kit which is available for a higher price, but has greater processing power and would give us better results for our project.
- We utilized the Raspberry Pi Camera as it was easier to integrate with the SSW bogie. The problem was that the bogie at corners was very shaky and did not have enough FOV to track the Sphero on multiple occasions. We had to compromise and utilize and parse data at certain intervals.
- We also decided to use some CMPE department robots for the time being to help us with the data collection. Depending on the ETA, we will use the CMPE department robots and fine tune the model to track the SSW bogies. We ended up utilizing the SSW track but they only had one bogie ready. We had to compromise by using a Sphero Bolt provided by our advisor as the second bogie to track.
- After trying our model with the data from SSW, we found it was not up to our expectations. We decided to collect data in a new form where an iRobot follows a Sphero robot in a straight line in a hallway. The data collected was about 5-6 hours and worked well with our model.

Chapter 5 System Implementation

5.1 Implementation Overview

For the implementation, we shall use Python (GRU/LSTM) to build the model. We may also utilize tensorflow to build the model. In order to identify the Sphero and estimate the area, we shall use the YOLO algorithm. YOLO stands for “you only look once” and is an algorithm that uses neural networks for object detection. It is often used for object detection uses such as people recognition due to its accuracy and speed. The main language that shall be used for implementation is python programming language. As far as hardware is concerned, we tried to collect data by attaching a PI cam to the SSW bogie and running around the track following the Sphero. One of the main implementation dependencies is having another bogie on the track to perform the object detection. Due to the Spartan Superway team not having another bogie ready, we used an iRobot Create 2 robot to follow the Sphero in a hallway. The task will be to use the video from the moving iRobot to perform object and velocity detection on the Sphero.

5.2 Implementation of Developed Solutions

- Our first proposed solution was to allow the Sphero robot to travel around the SSW track which was followed by one of their bogies developed. We tried to implement our model on this but the data collected was varying too much. This would be too hard for the model to predict based on the given values.

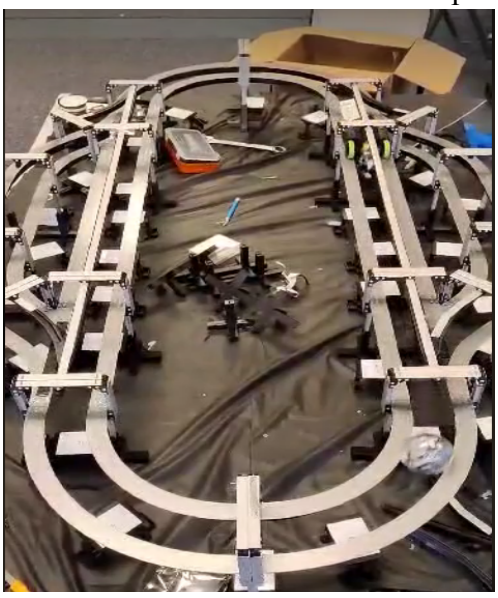


Fig 6. Original Method of Data Collection at SSW

- We changed our solution of data collection by using CMPE department robots provided by our advisor. We utilized an iRobot Create 2 and a Sphero robot to run through a hallway. The iRobot was programmed using a Python library - pycreate2. The Sphero was programmed using Javascript through the Sphero edu app. A camera was attached to the iRobot and followed the Sphero in a straight line. The velocity of the iRobot was 10 cm/s while the Sphero moved according to

a sine wave of the constant velocity (given below). We experimented running the iRobot and sphero at faster and slower speeds, but ideally 10 cm/s proved to be the best. Faster speeds proved to be hard to collect data as the sphero was inconsistent in the direction it moved and if the speeds were too slow, the sphero did not generate enough acceleration at times to move consistently.

$$V\_s(t) = V(t) * (1 + 0.1 * Math.sin(2 * Math.PI * t)))$$

Fig 7. New method of data collection using iRobot and Sphero

```
async function startProgram() {
  var t = 0;
  var vt = 10;
  var vs = 0;
  while (1) {
    let vs = vt * (1 + (0.1 * Math.sin(2 * Math.PI * t)));
    await roll(0, vs, 1);
    t = t + 0.001;
  }
}
startProgram();
```

Fig 8. Sphero Script - Script to control Sphero robot to move at a velocity according to the sine wave function provided

```
from  pycreate2 import Create2
import time

port = "COM5"
bot = Create2(port)

bot.start()

bot.full()

bot.drive_direct(100, 100)

# bot.drive_stop()

bot.close()
```

Fig 9. iRobot Script - Script to control the iRobot at a constant pace

- To extract the shape of the Sphero, the solutions we have come up with include using YOLO to break down the area of the bogie. We labeled and trained about 700 images split up into a 70/15/15 split of training, validation, and testing splits respectively. This process was done using Roboflow. Roboflow allows us to label all the images and export the data using their API provided. Once the API is provided we use the train.py script from YOLO to create our custom model.

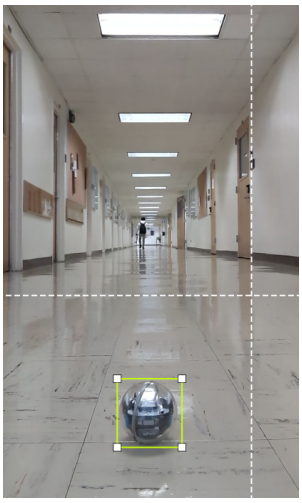


Fig 10. Process of Labeling Sphero using Roboflow

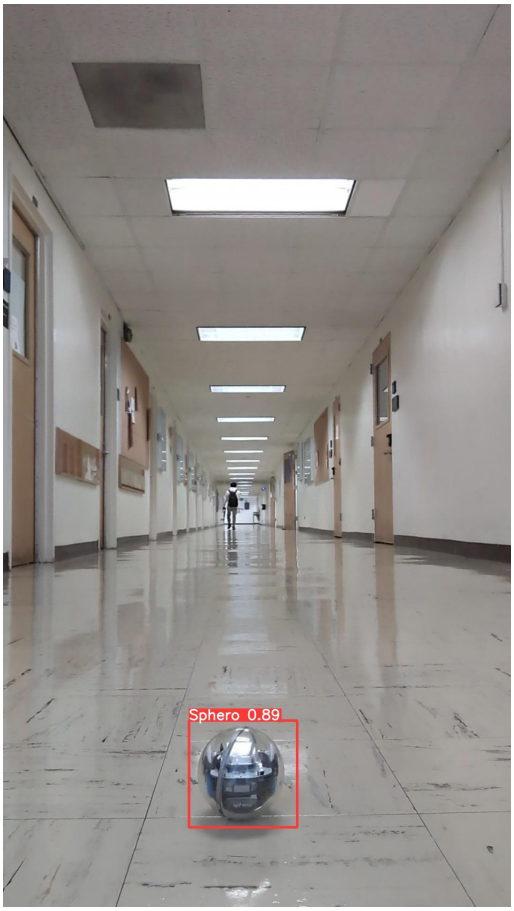


Fig 11. YOLO Results - Results of YOLO model training. Sphero is detected at an 89% accuracy in this image

- After training our YOLO model, we pre-processed all our video data collected. The sphero sensor data provides velocity in steps of around 0.15 seconds which is about 6 frames per second. We utilized a script we wrote to break down the frames using the same time steps. From here we modified the detect.py script given in YOLO to output areas of each bounding box detected in a frame. Instead of passing in direct images, it was easier to pass in a feature of the sphero as our input to predict a velocity at a given instance. YOLO already produces widths and heights for bounding boxes, thus we decided to use these values to compute the area for our input. Once this step is completed, we match the areas to the appropriate velocities and create a combined excel sheet. The original data was unable to be used since the sphero was giving us inconsistent results. We ended up using actual velocities from our sine wave function provided for the excel sheets.

```python
import cv2
import os
count=1
vid = "irobot/videos/trim_35.mp4"
vidcap = cv2.VideoCapture(vid)
def getFramePerSecond(sec):
    vidcap.set(cv2.CAP_PROP_POS_MSEC,sec*1000)
    results,image = vidcap.read()
    if results:
        cv2.imwrite("irobot/frames/trim_35/"+str(count)+".jpg", image)
    return results
sec = 0
frame_rate = 0.155

success = getFramePerSecond(sec)
while success:
    count = count + 1
    sec = sec + frame_rate
    sec = round(sec, 2)
    success = getFramePerSecond(sec)
```

Fig 12. getFrames Script - Script to get frames at every 0.15 seconds to match up the areas to the sphero generated csv velocity sheets

```
0 0.029172
```

Fig 13. Txt File Generation - Example of area generated from when we modified the detect.py script

	A	B	C
1	**time**	**area**	**velocity**
2	0	0.00778186	10
3	0.13	0.00784266	10.72897
4	0.3	0.00771915	10.95106
5	0.45	0.0074798	10.30902
6	0.63	0.00718348	9.271031
7	0.78	0.00783548	9.017713
8	0.93	0.00802505	9.574221
9	1.09	0.00795851	10.53583
10	1.24	0.00833908	10.99803

Fig 14. Sphero Excel Data - Compiled excel sheet of the area of bounding box of each frame matched to the velocity produced by the sine equation mentioned above

- The last step includes our GRU/LSTM model implementation. Our model should be able to predict the velocity given an area. For this, we experimented with various window sizes and came to the conclusion that 12 worked the best. We experimented with GRU/LSTM/BiLSTM/ models with various numbers of neurons. We tried utilizing multiple Dense layers as well as different activation functions and kernel initializers. Optimizers such as Adam with various learning rates were also tested. Utilizing an Adam optimizer helped us make predictions based on exponentially weighted average values. We decided against implementing a model too complex since our data was not as complex to be trained with. We ended up using an LSTM model of 15 units with 1 Dense layer with a kernel initializer of random_uniform (Figure 16). The learning rate that gave us the best results was 0.0001. Our network architecture and the results of various simulations are shown below.

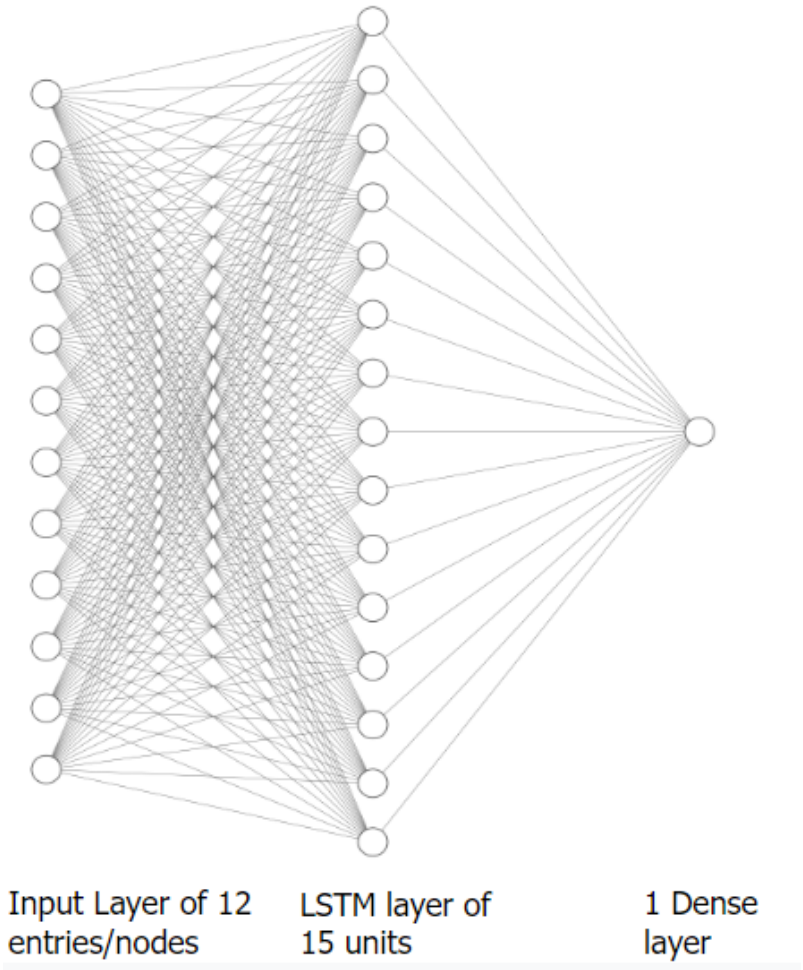
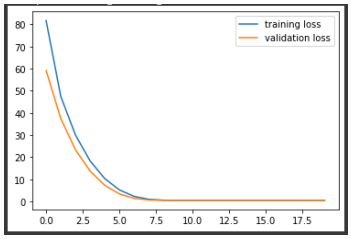
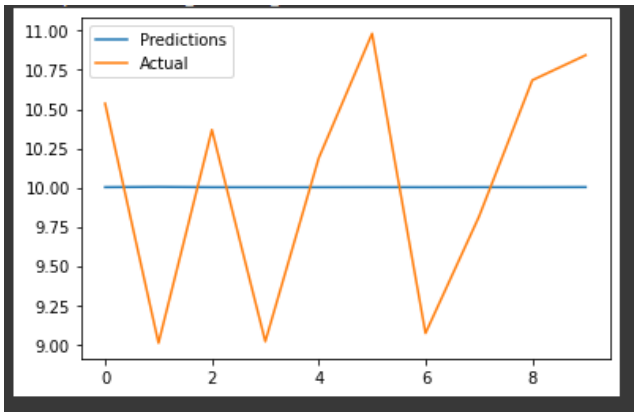


Fig 15. Network Architecture - Model that worked best for us was an input layer of 12 entries, a hidden LSTM layer of 15 units, and 1 Dense layer



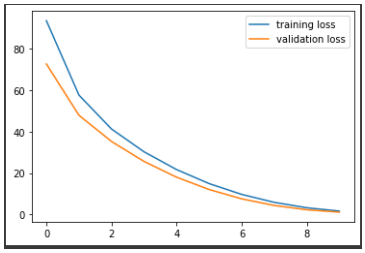

```
Loss: 0.497089719323628
RMSE: 0.7050458987354142
```

	Test Predictions	Actuals
0	10.002045	10.535827
1	10.003927	9.007885
2	10.001685	10.368125
3	10.001586	9.017713
4	10.001659	10.187381
5	10.002028	10.982287
6	10.001779	9.070224
7	10.002092	9.812619
8	10.001847	10.684547
9	10.002411	10.844328



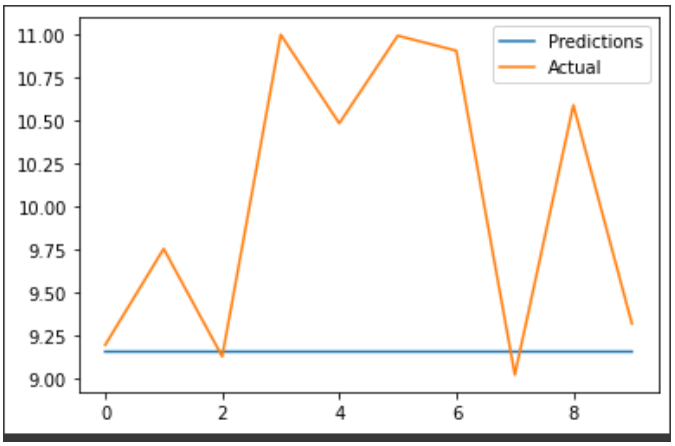

```
model = Sequential()
model.add(InputLayer((12, 1)))
model.add(LSTM(15))
model.add(Dense(1, name = 'output', kernel_initializer = 'random_uniform'))
```

Fig 16. Trial 1 - Model of LSTM of 15 units with window size 12 and one Dense layer



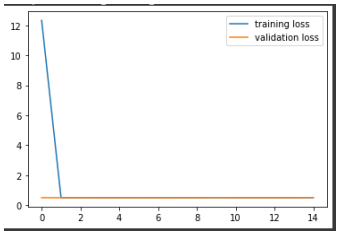

```
Loss: 1.1405155283812667
RMSE: 1.0679492161995656
```

	Test Predictions	Actuals
0	9.152164	9.190983
1	9.152149	9.751310
2	9.152167	9.123693
3	9.152162	10.998027
4	9.152149	10.481754
5	9.152148	10.992115
6	9.152151	10.904827
7	9.152147	9.017713
8	9.152159	10.587785
9	9.152154	9.315453



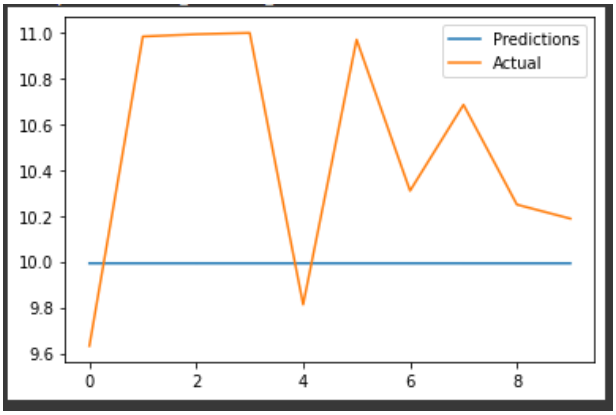

```
model = Sequential()
model.add(InputLayer((6, 1)))
model.add(GRU(10))
model.add(Dense(1, name = 'output', kernel_initializer = 'random_uniform'))
```

Fig 17. Trial 2 - Model of GRU of 10 units with window size 6 and one Dense layer



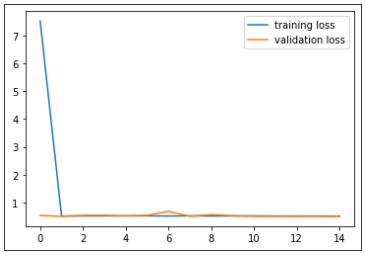

```
Loss: 0.49919575905508484
RMSE: 0.706537868097022
```

	Test Predictions	Actuals
0	9.992624	9.631875
1	9.992610	10.982287
2	9.992598	10.992115
3	9.992657	10.998027
4	9.992628	9.812619
5	9.992637	10.968583
6	9.992616	10.309017
7	9.992591	10.684547
8	9.992591	10.248690
9	9.992602	10.187381



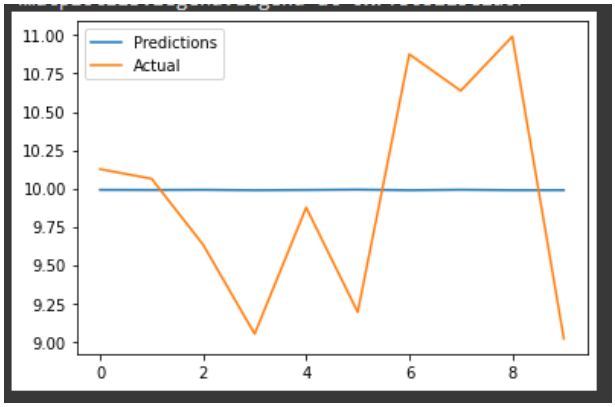
```
model = Sequential()
model.add(InputLayer((24, 1)))
model.add(Bidirectional(LSTM(64)))
model.add(Dense(8, kernel_initializer = 'random_uniform', name = 'fc1'))
model.add(Dense(1, name = 'output', kernel_initializer = 'random_uniform'))
```

Fig 18. Trial 3 -Model of BiLSTM of 64 units with window size 24 and two Dense layers



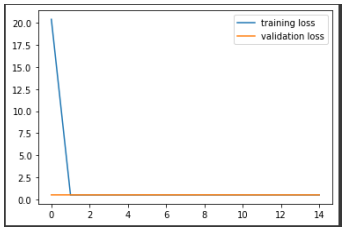

```
Loss: 0.5001563785963571
RMSE: 0.7072173489079274
```

	Test Predictions	Actuals
0	9.989912	10.125333
1	9.989146	10.062791
2	9.990205	9.631875
3	9.987528	9.048943
4	9.989190	9.874667
5	9.991509	9.190983
6	9.987648	10.876307
7	9.990561	10.637424
8	9.987921	10.992115
9	9.988103	9.017713



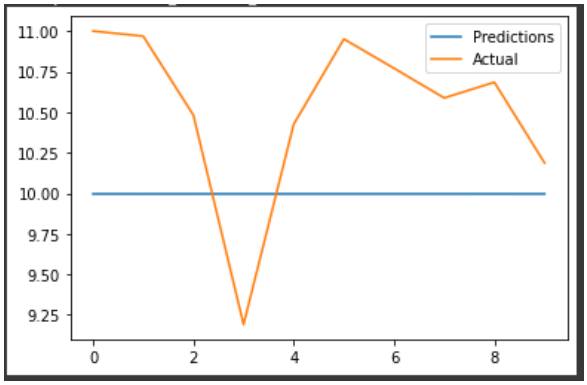

```
model = Sequential()
model.add(InputLayer((16, 1)))
model.add(LSTM(128))
model.add(Dense(100, kernel_initializer = 'he_normal', name = 'fc1'))
model.add(Dense(50, kernel_initializer = 'he_normal', name = 'fc2'))
model.add(Dense(10, kernel_initializer = 'he_normal', name = 'fc3'))
model.add(Dense(1, name = 'output', kernel_initializer = 'he_normal'))
```

Fig 19. Trial 4 - Model of LSTM of 128 units with window size 16 and four Dense layers




```
Loss: 0.5050496789559312
RMSE: 0.7106684733094125
```

	Test Predictions	Actuals
0	9.996069	11.000000
1	9.996031	10.968583
2	9.996067	10.481754
3	9.996206	9.190983
4	9.996227	10.425779
5	9.996156	10.951057
6	9.996141	10.770513
7	9.996149	10.587785
8	9.996048	10.684547
9	9.996192	10.187381




```
[ ]  model = Sequential()
     model.add(InputLayer((48, 1)))
     model.add(GRU(64))
     model.add(Dense(50, kernel_initializer = 'random_normal', name = 'fc2'))
     model.add(Dense(10, kernel_initializer = 'random_normal', name = 'fc3'))
     model.add(Dense(1, name = 'output', kernel_initializer = 'random_normal'))
```

Fig 20. Trial 5 -Model of GRU of 64 units with window size 48 and three Dense layers

5.3 Implementation Problems, Challenges, and Lesson Learned

Our implementation problems include being delayed due to being unable to get extra data. The main challenge is due to SPARTAN Superway designing a new track and multiple bogies. Due to this, we have to wait until the parts get delivered to start utilizing YOLO to break down the area of the bogie. This has also delayed our implementation in tracking the object and velocity prediction. To combat some of these challenges we decided to use two mobile robots provided by the computer engineering department in place of the bogies we are currently unable to use. We collected around 6 hours of data with the iRobot following the

Sphero in the engineering hallway. After collection of data and preprocessing, the model produced a loss of about 0.5 which is optimal.

This project has allowed us to go through the process of a machine learning project. We underestimated the amount and type of data needed for training models. We later learned the amount of data needed and started to collect more in the engineering hallway. The data collection was by far the hardest part of the project and machine learning in general. The success of any machine learning project depends on good data. This will be helpful in our future if we all decide to pursue machine learning.

Contributions:

We all contributed equally to this project. We were all involved in data collection and preprocessing. Any of us who were free during the day would collect data and we would take turns doing this. We all split up the labeling parts of YOLO as well as the preprocessing part of the data. We all looked into the implementation of GRU/LSTM models and tried out different solutions.

Simran Bains - Data Collection, YOLO training/testing, Data Preprocessing, Model Implementation/Testing

Abhishek Karnik - Data Collection, YOLO training/testing, Data Preprocessing, Model Implementation/Testing

Rohan Surana - Data Collection, YOLO training/testing, Data Preprocessing, Model Implementation/Testing

Surya Ram - Data Collection, YOLO training/testing, Data Preprocessing, Model Implementation/Testing

Chapter 6 Tools and Standards

6.1. Tools Used

Hardware used:

1. **Sphero**
    - The sphero was used as the main bogie to be tracked since the SSW team only had one bogie available on hand. We programmed this bot to run around the track in front of one of the SSW bogies. We later used the sphero in the engineering hallway in front of the iRobot. From here we extracted the velocity information and used this for our object and velocity prediction model.

2. **Bogie x 1**
    - This bogie was provided by the SSW team for the new track they built. This was allowed to run behind the bogie to collect data from. *This eventually did not work for our project needs.*
3. **Raspberry Pi Cam**
    - This camera was used as an attachment to the bogie to follow the sphero. From this camera, we collected the resultant video and broke down frames to utilize to build our model. *We were unable to use these frames due to inconsistencies with data.*
4. **Jetson Xavier NX Developer Kit**
    - This kit is used to load the model we built on. We will attach this to the bogie which will help identify the Sphero later on.
5. **Spartan Superway Bogie Track**
    - This track was built by Spartan Superway and we used it to collect information for our model. One bogie and a sphero were programmed to travel around the track and collect the appropriate amount of data. *The data collected with this track was unusable due to inconsistencies.*
6. **iRobot Create 2**
    - This was used for the new form of data collection. The iRobot was programmed using python to travel in a straight line at a constant velocity to follow the sphero. A phone was attached to this iRobot to collect the video data.

Software Used:

1. **Jupyter Notebook/Google Colab**
    - Jupyter Notebook and Google Colab was our main IDE used in this project. We installed and ran our dependencies through these two tools. Google Colab was used to train our model primarily due to the free GPU provided for faster results. Jupyter Notebook has our other code which was used to test our results with. Colab was also used to share GRU/LSTM code/models with.
2. **Pytorch/YOLOv5**
    - Pytorch/YOLOv5 was used as the object detection model. YOLO is now the state of the art system in terms of object detection and we used this to train our sphero frames collected from our video.
3. **Github**
    - Github is used as our code collaboration tool. This is where we uploaded our weights and code to experiment with. We all tried

different epochs to train with and this is where we would upload our best weights for everyone to try in terms of object detection.

4. **Zoom/Messenger**
    - Zoom and Messenger are our primary collaboration tools. We set up calls with one another to work and figure out any problems we have on our model. We also use Zoom to communicate with our advisor.
5. **Roboflow**
    - Roboflow was used as the image labeling tool. Roboflow provides easier access. Results were stored in the cloud which was easier to access through their API. We used this as our input to train our model.
6. **Python (Keras)**
    - GRU/LSTM is used for the velocity prediction taken from our Sphero Script. We currently have a dataset of areas and velocities at that given time. From here, we are still trying to predict velocities from a given area.
7. **Python (OpenCV)**
    - Python and OpenCV are used to read a sample test video and display results. For our object detection model, we would load our model and predict the accuracy of each area of our input video. From here we would display the results on the video which accurately detects our Sphero.

6.2. Standards

- One of our main software standards is to use industry specific coding standards. In terms of object and velocity detection, we chose to implement this with Python as this is the industry standard. The main reason Python is due to the frameworks and tools it provides. The frameworks, libraries, functionality it provides is simpler in terms of implementation. The access to the amount of libraries is a main standard of why Python is used for this project. It is also easier to understand by other peers.
- Another software standard we incorporated is code readability. Code readability is very important as we have divided the code blocks into appropriate blocks in Google Colab/Jupyter Notebook. This helps us

arrange the order of code and separates imports from code. Segmenting these functions makes it easier for any of us accessing it at a later time.

- Another software standard we followed is prioritizing documentation/comments. Throughout 195, we have been documenting and keeping track of hardware and flow models. We also have been following this procedure and have been keeping track of our progress. We also include comments to our code blocks to let each other know what is happening at this code block.
- We also incorporate backups of our code as a standard software practice. The reason we do this is because we try to experiment with different solutions. In case our main script does not work, we still have the working backup to rely on.

Chapter 7 Testing and Experiment

7.1 Testing and Experiment Scope

- Our test process mainly consists of blackbox and whitebox testing. For example, our object detection model included labeling images with roboflow, and then we would first test with sample images for the time being to check if the model works. In terms of integration and regression testing scenarios, we would then test with a sample video to see if the model is accurate enough to predict our sphero. Our results and examples of unit and integration tests are shown below for the object detection model.
- Another test process of ours included using GRU/LSTM to predict a model's velocity. As of right now, we had a testing, validation and training split of about 15%, 15%, and 70% respectively. Our whole set consisted of about 44000 images with time and velocities matching each frame. Our whole goal is to match our predictions with the appropriate true values per frame. Our RMSE value stands at about 0.705. We later switched to a model of an 80% training and 20% testing split with YOLO areas matching the appropriate velocities.
- Our regression tests also included retraining the model numerous times to get the best results. The model would sometimes inaccurately detect other forms of objects as spheros, but we ended up training the best possible model we could to get rid of false predictions.

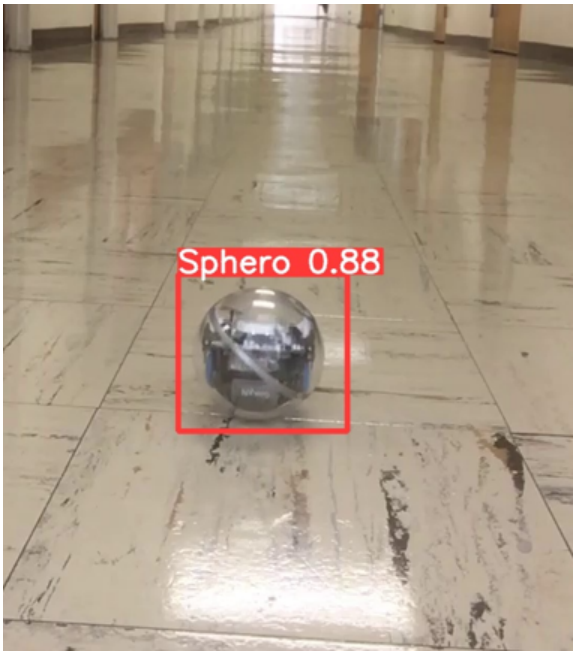


Fig 21. Unit testing - Unit testing an image from the YOLO model we trained.

Fig 22. - Integration Testing 1 - Image captured from video output showing functionality of YOLO model

Fig 23. - Integration Testing 2 - Image capture from YOLO video showing functionality of model

7.2 Testing and Experiment Approach

In order to test our model, we primarily used various forms of blackbox testing. In order to test the object detection model, our approach was to use sample images to do initial testing. This would give us an idea if the model functioned as per our expectations. Then we moved on to using video data to test the model, and make changes as necessary using the results from that testing. This approach proved useful as initially testing with images reduced the amount of time taken for testing and allowed us to fine-tune the model. We performed several regression tests involving the retraining of the model in order to improve the predictions. Initially, the predictions were inaccurate and random objects were detected as the sphero. This has been significantly improved as the model has continuously been refined. For testing the GRU/LSTM velocity predictions, we went with a traditional approach of splitting the data into a 20% testing and 80% training split. This allowed us to use 80% of the data to provide adequate data to train the model, and left us with 20% to run tests afterwards. We have mapped velocities to around 44000 images(areas) with time and analyzed our output velocities from our testing sets which have areas. We have experimented with different factors as mentioned in section 5. We also look at the graphs of the loss vs epoch. We ideally want the loss to decrease immensely and stand below 1. Figure 23 shows our graph of training and validation loss versus the epochs. The success of this test is measured using RMSE, which stands at 0.705 using the sine wave function provided previously in the document.

7.3 Testing and Experiment Results and Analysis

The YOLO model trained for object detection was able to accurately locate and identify the sphero in all of the videos from our test set. As seen in the image capture from figure 20, it successfully finds the sphero within the video and forms a label around it. Upon running predictions from our GRU/LSTM model on the test data, the model achieved a root mean squared error (RMSE) of about 0.705. We collected and processed around 44000 frames of data where we matched the areas and appropriate velocities. The iRobot runs at a constant velocity of 10 cm/s in a straight path while the Sphero runs at a velocity based on the sine wave function using the constant velocity. We took about 6 hours of data to feed the model to ensure the amount of data was sufficient to train on.

The YOLO model performs well and fully satisfies all of the requirements. The GRU/LSTM model implementation seemed to work well with the data we have collected. Our loss has decreased immensely while training the model as seen in Figure 22. The loss (mean squared error) currently stands at about 0.497 which is optimal for a machine

learning model. We tested the predictions of the model where we got almost a constant value which correlates with the sine wave data we feed it in.

From testing with different RNN models, LSTM worked best for us. The results of the LSTM model trained with one hidden layer of 15 units and a Dense layer of 1 neuron is shown below.

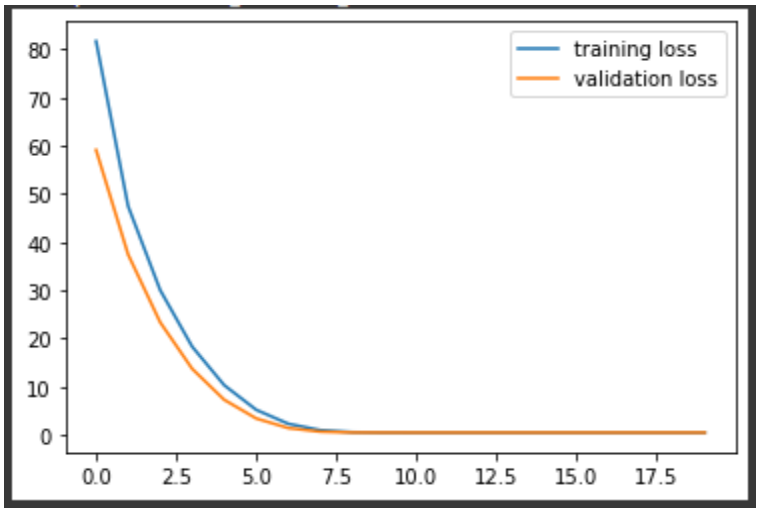


Fig 24. Loss (Mean Squared Error) Graph - Graph showing training and validation loss for 20 epochs

```
Loss: 0.497089719323628
RMSE: 0.7050458987354142
```

Fig 25. Loss (MSE)/RMSE Values - Results of mean squared error and root mean squared error of model

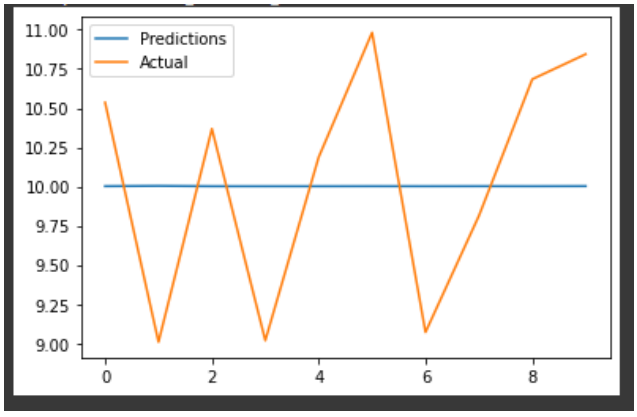


Fig 26. Graph of predictions versus actual velocities.

	Test Predictions	Actuals
0	10.002045	10.535827
1	10.003927	9.007885
2	10.001685	10.368125
3	10.001586	9.017713
4	10.001659	10.187381
5	10.002028	10.982287
6	10.001779	9.070224
7	10.002092	9.812619
8	10.001847	10.684547
9	10.002411	10.844328

Fig 27. Table of predicted vs actual values. The model being constant in terms of sine wave values predicted shows our model is set up properly.

Chapter 8 Conclusion and Future Work

Conclusion

We prepared a model that would predict the velocity in real-time using the area extracted from the frame. We first trained a YOLOv5 model for predicting the sphero from the video and, using the model, created a box surrounding the sphero, which is used to extract the area. After preprocessing the data, we train a GRU/LSTM model, where the input is area and the output is velocity.
We shifted our focus from SSW to our own method of data collection which proved to be tedious but provided better results overall. Our best RMSE value stands at about 0.705 while our loss (mean squared error) stands at about 0.497. The velocities produced by the sphero given the sine wave function were still too inconsistent to train with. For this reason we used the time generated from the script and generated values directly using the equation on excel. After concatenating the excel sheets and running the model, our results improved immensely. When compared to the data collected from SSW, our RMSE from our own method was at 0.705 while at SSW the RMSE was around 12. Our model is

trained with different objects and speeds, but SSW can still utilize this for their own purposes.

Future Work:

We would like to work on data collection using efficient methods for future work. The current method of data collection included creating a script to match the frames to the velocity and extracting the area of the YOLO output at each frame. We integrated all this data in an excel sheet and got data for each video. This was tedious and time-consuming. Using the iRobot and Sphero met our needs for this project, but we would like to utilize Spartan Superway's bogies and tracks once they are ready. Once their equipment is ready, data collection would be easier and the real world data would be applicable. This would be a better alternative compared to having a sphero run around the track for about 1-3 minutes at a time. The predictions from smoothly moving bogies would be consistent. Future work would include integrating the robots/bogie completely with the Nvidia NX board.

References

Jocher, G. (2020, June). Ultralytics/yolov5: Yolov5 🚀 in PyTorch ONNX CoreML TFLite. Retrieved February 20, 2022, from https://github.com/ultralytics/yolov5

Khandelwal, R. (2019, November 17). Multivariate time series using gated recurrent unit -GRU. Retrieved April 3, 2022, from https://medium.datadriveninvestor.com/multivariate-time-series-using-gated-recurrent-unit-gru-1039099e545a

Rasifaghihi, N. (2020, August 30). Predictive analytics: Time-series forecasting with GRU and BiLSTM in tensorflow. Retrieved April 10, 2022, from https://towardsdatascience.com/predictive-analytics-time-series-forecasting-with-gru-and-bilstm-in-tensorflow-87588c852915

Solawetz, J. (2020, June 10). How to train yolov5 on a custom dataset. Retrieved March 4, 2022, from https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/

Appendices

Appendix A – Github Code Link

https://github.com/rohan2810/object-detection

Appendix B – Lockheed Report Link

https://docs.google.com/document/d/1m_FZbQegXxHvrI7kwiOvllzemmfN7Lvl/edit?usp=sharing&ouid=111528020608987887823&rtpof=true&sd=true