School of Informatics & IT
TEMASEK POLYTECHNIC

Temasek Polytechnic
School of Informatics and IT
Diploma in Applied Artificial Intelligence
AY 2022/2023 Sem 2

Major Project Reflection Report

NVIDIA TX1 Autonomous Vehicle

Submitted To: Mr. Tan Sio Poh
Submitted By: Maximilian See Tze Jie 2102869A
Supervisor: Mr. Tan Sio Poh

Students' Declaration of Originality

I hereby declare that the coursework(s) submitted is a result of my own efforts. I affirm that there is NO plagiarism and copying, either partially or entirely, from someone else's design and works, without giving proper credit and acknowledgement to the source(s)/author(s).

I am aware that I shall be subjected to disciplinary actions deemed appropriate by the School of Informatics & IT and Temasek Polytechnic if I am found to have committed or abetted the offence of plagiarism in relation to this submitted work.

<Signature> 

Maximilian See
2102869A
2023 September 9

Table of Contents

Acknowledgements

I would like to extend our heartfelt appreciation to Mr. Tan Sio Poh, our supervisor, for his invaluable guidance and active engagement throughout the project's development. Without his expert advice, insightful input, and numerous valuable suggestions, the project would not have achieved the level of success it did. Mr. Tan's support has been instrumental in our accomplishments.

Abstract

Autonomous vehicles have transcended the realm of science fiction. This transformation is driven by advancements in cutting-edge technologies such as sensors, cameras, and LiDAR systems, democratizing access to autonomous vehicles and robots for corporations and consumers alike. The contributions of Tesla and the dynamic Chinese market are pivotal in this journey. Tesla, as a trailblazer, has harnessed vast datasets to develop and train sophisticated models, enabling vehicles to autonomously navigate complex traffic scenarios. Our project aligns with this dynamic landscape, representing our commitment to contribute to the ongoing evolution of autonomous vehicles. Our focus is not on novelty, but to bring more understanding and transparency to the technology as well as hopefully spark some inspiration in the public

Chapter 1 | Introduction

Objective & Deliverable to be Produced.

For our project, we tasked ourselves with programming the TX1 paired with Arduino and Polulu boards to perform image recognition on the robots. Our primary objective was to program the robot to recognize and respond to various signs, including the Forward, Left, and Right signs, utilizing a deep learning pretrained model that is deployed to an edge device (TX1 with Arduino and Polulu Boards) running the Robot Operating System, ROS and moving according to the signs displayed. Our focus is not on novelty, but to bring more understanding and transparency to the technology as well as hopefully spark some inspiration in the public.

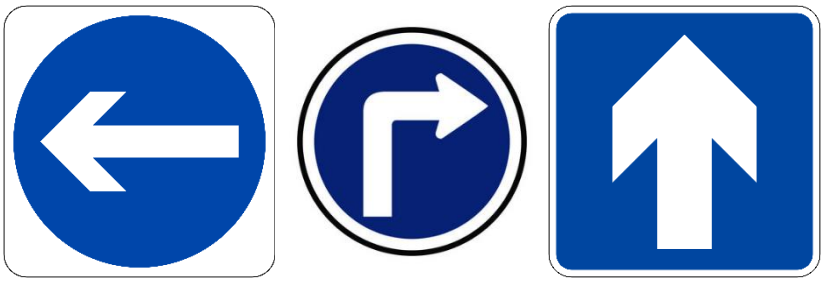


Fig 1: The images above showcase the signs we selected for training.

The following text outlines our problem-solving approach and delineates the steps we followed to tackle these challenges. To begin, we will examine the essential hardware components used in constructing the robot. Subsequently, we will provide an overview of our collaborative work process, illustrating how we addressed the issues at hand.

Furthermore, we will delve into the obstacles we encountered during this project and explore potential avenues for future enhancements.

Targeted Users

Our primary focus is on students, including those in educational institutions and high schools. We aim to ignite their interest in AI models and demonstrate the real-world applications of deep learning and models they learn in school.

Additionally, with future improvements to the robot, we plan to target companies in various sectors:

- Food and Beverage: Tray return Robots as seen in cafeterias in Changi Airport
- Logistics and E-commerce Firms: Stock Counting in warehouses

Our goal is to bridge theory and practice while fostering collaboration in the realm of AI and robotics.

What is ROS?

ROS, or Robot Operating System, is an open-source framework widely used in the field of robotics for developing and controlling robots. ROS is a middleware platform that provides a comprehensive set of tools, libraries, and communication protocols to facilitate the development and coordination of robot software.

ROS enables developers to create and manage robotic applications efficiently by offering functionalities like hardware abstraction, device drivers, communication between different components of a robot system, and a wide range of pre-built software modules. This flexibility and modularity make ROS a popular choice for researchers, engineers, and hobbyists in building and customizing robotic systems for various applications, from autonomous vehicles and industrial automation to research and educational projects.

Why ROS?

The ROS framework which was developed all the way back in 2007 is more mature than its counter parts like the Nvidia Isaac software which was launched in 2017. The maturity of the ROS software and its open-source nature has allowed for many iterations for ease of access and understanding as well as licensing. Thus, for this project ROS is chosen as our framework.

The era in which the frameworks are developed is key as they reflect the hardware which the software is made for. As the TX1 is relatively old in today's standards this software consideration is important as it might affect the speed of inference on images, which might affect the movement of the robot.

NVidia also has specific hardware & software compatibility requirements. This means that not all devices could work with Nvidia products, and only Nvidia certified products could be used with their hardware or software. An example would be the latest Nvidia Orin which limits compatibly. (https://developer.nvidia.com/drive/ecosystem-orin#camera)

Thus, by choosing ROS, we limit the number of hardware or software issues, and increase the performance of our TX1 board though compatible hardware optimization and software compatibility.

Existing Products

	DonkeyCar	Our Robot
Price	~$250	~$800
Features	Follow Lines Stop Sign Detection	Image Recognition for Turn Signs

The image above shows the comparison between two products.

Donkey Cars

DonkeyCar is an open-source platform and community that focuses on creating do-it-yourself (DIY) self-driving cars and autonomous robotic vehicles. It was initially developed as a project to help individuals, especially hobbyists and students, learn about autonomous vehicle technology in a hands-on and affordable way.

Key components of a DonkeyCar project typically include a small-scale remote-controlled car or vehicle, a single-board computer like a Raspberry Pi, various sensors such as cameras and accelerometers, and the DonkeyCar software stack. The DonkeyCar software stack includes machine learning tools and a Python-based framework for training and running neural networks to control the car's steering, throttle, and brake systems based on input from the sensors, much like the ROS software stack.

The base model of a DonkeyCar Comes in at around 250 USD. Assuming it has a RaspberryPI4, the general performance on a YOLOv5 model is about 1.6 Frames Per Second.

Model	size	objects	mAP	Jetson Nano	RPi 4 1950	Rock 5	Rock 5 NPU
NanoDet	320x320	80	20.6	26.2 FPS	13.0 FPS	36.0 FPS	
NanoDet Plus	416x416	80	30.4	18.5 FPS	5.0 FPS	24.9 FPS	
YoloFastestV2	352x352	80	24.1	38.4 FPS	18.8 FPS	65.4 FPS	
YoloV2	416x416	20	19.2	10.1 FPS	3.0 FPS	20.0 FPS	
YoloV3	352x352 tiny	20	16.6	17.7 FPS	4.4 FPS	15.0 FPS	
YoloV4	416x416 tiny	80	21.7	16.1 FPS	3.4 FPS	22.4 FPS	
YoloV4	608x608 full	80	45.3	1.3 FPS	0.2 FPS	1.5 FPS	
YoloV5	640x640 small	80	22.5	5.0 FPS	1.6 FPS	12.5 FPS	40 FPS

Nvidia TX1

The NVIDIA Jetson TX1 is a compact and powerful embedded computing module designed for applications that require high-performance, low-power processing for computer vision, machine learning, and artificial intelligence tasks. It is part of NVIDIA's Jetson family of products, which are specifically engineered for edge computing and AI at the edge.

Key features (https://developer.nvidia.com/embedded/jetson-tx1):

- **GPU and CPU**: The Jetson TX1 is equipped with a Maxwell architecture-based NVIDIA GPU with 256 CUDA cores, as well as a quad-core ARM Cortex-A57 CPU. This combination of GPU and CPU provides the processing power necessary for AI and computer vision workloads.

- **Memory**: It has 4 GB of LPDDR4 memory, which is shared between the GPU and CPU, enabling efficient data sharing and processing.

- **Connectivity:** The TX1 includes various connectivity options, including USB 3.0, HDMI, Gigabit Ethernet, and support for Wi-Fi and Bluetooth.

- **AI and Machine Learning:** The Jetson TX1 supports popular deep learning frameworks like TensorFlow, Caffe, and PyTorch, making it suitable for a wide range of AI and machine learning applications.

- **Power Efficiency:** One of the standout features of the Jetson TX1 is its power efficiency, making it suitable for battery-powered or embedded systems.

With optimized configurations and lower image resolutions, WE can expect real-time or near-real-time performance (greater than 20 FPS) for YOLOv5 object detection on the Jetson TX1.

Coming in at around 800 dollars including all the other components, this is not a cheap product. However, although the cost is high the number of opportunities for further development is increased due to the increased ports, Wi-Fi as well as hardware performance.

Note: YoloV5 is being used as a benchmark to determine hardware performance.

What is Caffe & Why Caffe

Caffe is an open-source deep learning framework developed by the Berkeley Vision and Learning Center (BVLC). It was one of the early deep learning frameworks and gained popularity for its speed and flexibility. The project adopted the use of Caffe due to these key reasons.

- **Maturity**: Launched in 2013, Caffe has been around for quite some time and has a well-established user base and community support. It has undergone extensive development and optimization over the years, which has contributed to its maturity as a deep learning framework.

- **Efficiency**: Caffe is known for its efficiency in terms of model inference. It was initially designed for high-performance image classification tasks and can leverage GPU acceleration effectively. This can be advantageous for running computationally intensive models on the NVIDIA Jetson TX1, which has a GPU.

- **Flexibility**: Caffe provides flexibility in designing and fine-tuning neural networks. It has been used in various research and production settings and supports a wide range of layer types and network architectures.

Chapter 1.5 | Main Components

Main hardware components that the autonomous vehicle required.

- Nvidia TX 1 - Controls the movements of the autonomous vehicle, by running code on the robot.
- Battery 12V - Powers the autonomous vehicle (All components)
- Arduino Uno - Controls the low-level signals sent by the Arduino and Nvidia TX1 board.
- Polulu Board - Wheel Motor Power comes from the Polulu board which is powered by the battery.
- Logitech Camera 720P – For the computer vision component
- Wi-Fi Router
- Micro USB to USB Type A

Main Software components that the autonomous vehicle required.

- VMware Pro or Laptop – Virtual Machine or Laptop with Ubuntu 18.04 to flash our robot.
- Ubuntu 18.04 LTS – Linux Distribution
- Python – To allow us to pip install if required.
- Caffe – To build and convert our TF lite model to caffe model.
- OpenCV – Part of the machine learning package for our model
- C ++ - The language we would be coding our robot in
- ROS Version - Melodic
- Jupyter Lab

Assumptions

This section outlines a set of assumptions related to the presence and positioning of a sign in front of a robot operating within a laboratory setting.

- **Assumption 1: There Will Always be a Sign in Front of the Robot**

The first assumption is that there will always be a sign placed in front of the robot.

- **Assumption 2: The Robot is in a Lab Setting**

This assumption establishes the context for the robot's operations. It assumes that the robot operates exclusively within a laboratory setting. This environment is likely to have good lighting that may differ from other settings.

- **Assumption 3: The Sign Will Be Upright**

The third assumption pertains to the orientation of the sign. It assumes that the sign placed in front of the robot will always be in an upright position. Maintaining the sign's proper orientation is crucial to ensure its visibility and effectiveness of the predictions. This implies that measures will be in place to prevent the sign from tilting, falling, or becoming obscured during the robot's operation.

- **Assumption 4: The Sign Will Only Indicate Left or Right Turns**

The final assumption is the functionality of the sign. It assumes that the sign will serve as an indicator for the robot's intended direction, specifically, indicating left or right turns.

Chapter 2 | Reflection: Planning

Things I had to prepare to do our project.

Learning Linux
As we are new to the concept of Linux, we had to undergo some research on the internet. This included the basics on Linux like navigating through the directories, Secure Shell or Moving and Copying files or simply using VIM editor to edit the source code.

Learning System Constraints
After learning about the concepts of Linux, we learn about the constraints of the machine or what can be done and what cannot be done. For example, how large of a model can it run? What resolution is the camera, how many frames can we record a second? Do we have enough space to work with after installing all the necessary packages? What packages do we need and do not need in the Jetson TX1. To find answers to these questions a quick hardware survey on the internet showed that the TX1 has about 4 GB of Ram and 4 Core CPU with 16 GB of Storage (https://developer.nvidia.com/embedded/jetson-tx1) which was enough to run a small YOLO model at 5 – 12 Frames Per Second (FPS) according to some users on the internet. With regards to the finding, we limit our capture Input to 10 FPS to reduce the latency of our inferences as more frames to capture usually means higher CPU usage and Higher Latency which might affect our TX1 prediction and movement.

When flashing, the TX1 had an option to flash it with NVIDIA Deep Stream which is a streaming analytic toolkit. As we are not doing analytics on the TX1 I decided to not include that as a package to be installed during the flash. A similar process was done after flashing the robot to remove the unnecessary packages found in the TX1 like Libre Office or Chromium. Removing those packages saved us about a gigabyte of space. This would save space for essential hardware like OpenCV and Caffe to be installed. The first time I flashed the robot I ran out of space downloading software like OpenCV and Caffe due to the space NVIDIA packages took (Assumed it's an edge device thus would have lesser bloat software and had no idea that it took more than 3 Gb which is considered a lot considering its 20% of the system Storage.) and had to restart the whole flashing which took precious time.

Physical Inspection
One important aspect is to do a physical inspection of the robot. During the process we inspect if any wires are loose or cables are missing (In our case we missed a USB-A to USB-B cable). What should light up and what colour should it be (In our case it's the MOT PWR – Did not light up due to a faulty switch). There was also a physical inspection in the wiring of the robot to see if all connections are as stated in the wiring sheet provided. These hardware issues plagued the project the entire run.

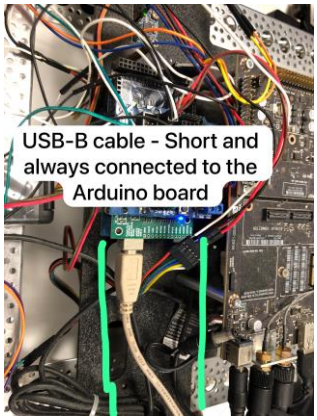
Learning ROS
Becoming acquainted with the ROS concept and exploring projects elaborated by "Articulate Robotics" on their YouTube channel (https://www.youtube.com/@ArticulatedRobotics) is an integral part of my journey. These projects shed light on crucial elements such as Odometry and the Transform systems with diagrams, which are pivotal for comprehending how our robot navigates within the ROS framework.

Additionally, delving into the concepts of Topics and Nodes is equally crucial. These concepts illuminate the communication between the robot's motors and sensors with the central controller, akin to a "conversation." They provide a foundational understanding for deciphering the C++ Code that leverages these principles to operate effectively.

Learning how to Flash the TX1

The TX1 requires a ubuntu based software for flashing. During the installation of the packages for flashing there was a bug that hangs the installation at 99%. Due to this, I had to research the ways to flash the TX1 on the internet. Some users recommended using the command line, but others commented that using the GUI version was the new standard. So, after trying both ways which did not work, I assumed that the problem was the VM itself and searched if there were any issues with using a VM. Some commented on the NVIDIA Forum that flashing with a VM is not supported, however some have successfully done it and it was due to the Virtualisation software like VirtualBox (Which I was using to flash the TX1) or VMware having issues with the USB passthrough software. Thus, I decided to try VMware instead to flash the TX1 which worked due to VMware allowing for USB Passthrough.

Learning how to Flash the Arduino



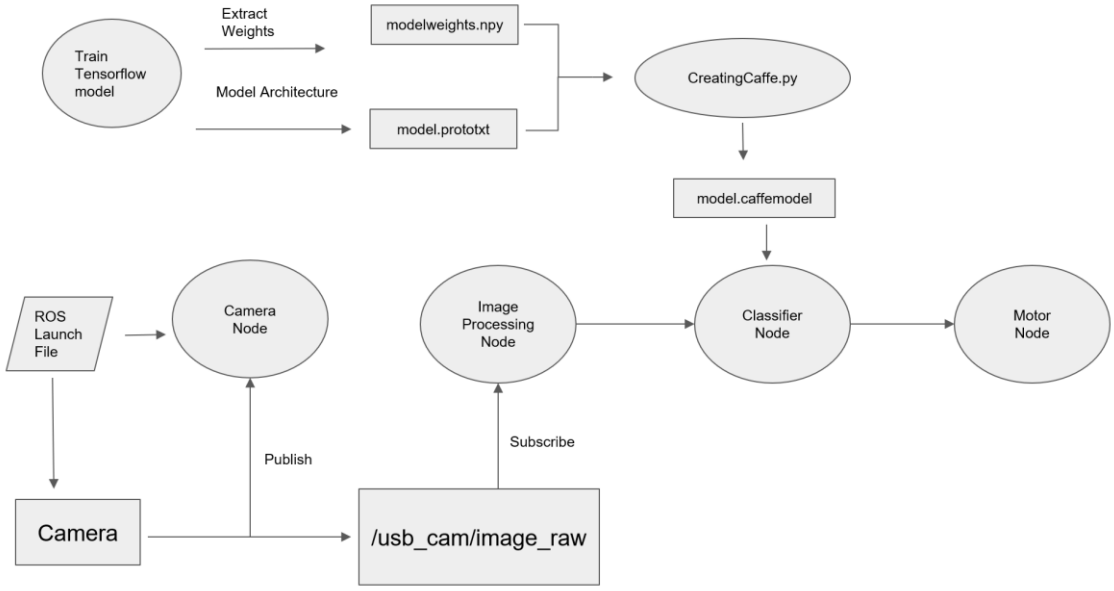
The Arduino required a USB A to USB – B adapter. To flash the Arduino, I must send the file to the TX1. Two reasons why it is more convenient to flash from the TX1 is due to the short a USB A to USB wire which means that I must carry the laptop and type on it and flash it. The other reason would be convenience. If I had to change any of the Arduino code I could do it on the TX1 and immediately flash, it. To flash the Arduino bot, I would also require the getplatform.io package.

Learning ROS Topic Commands

The usage of ROS Topic commands has allowed the findings of several hardware issues like the Motor receiving no power due to the faulty switch, as well as trouble shoot software issues like modelling. (Ryu models were reporting wrong opposite/predictions due to the way the model was trained on his side (See Figure 4 below for more clarity). By learning the commands like *roscore* and *rostopic list* I was quickly able to spot the error and remediate it.

Chapter 2.5 | System Architecture

The diagram below shows how the robot works.



The tutorial to convert the tflite model to the caffe model:
https://youtu.be/9iJheyF7x4Y

The first section describes the modelling process. First, a tensorflow model is trained. Using the tutorials as linked above we extract the weights of the model to a modelweights.npy numpy file, at the same time we also create our intended model architecture, a simplified version of the VGG19 model which removed the last 2 groups of convolutional and max pooling layers.

After Ryu have the two ingredients we create a python file called CreateCaffe.py to convert the tflite model to a caffe model. The first part of the process is best done in a Linux enviroment with Caffe installed which I (Max) has/have setup. This is due to the deprecation of the caffe packages on windows

```
Package libsodium conflicts for:
pyzmq -> libsodium[version='>=1.0.16,<1.0.17.0a0|>=1.0.18,<1.0.19.0a0']
ipykernel -> pyzmq[version='>=20'] -> libsodium[version='>=1.0.16,<1.0.17.0a0|>=1.0.18,<1.0.19.0a0']
zeromq -> libsodium[version='>=1.0.16,<1.0.17.0a0|>=1.0.18,<1.0.19.0a0']
notebook -> pyzmq[version='>=17,<25'] -> libsodium[version='>=1.0.16,<1.0.17.0a0']
nbclassic -> pyzmq[version='>=17'] -> libsodium[version='>=1.0.16,<1.0.17.0a0|>=1.0.18,<1.0.19.0a0']
jupyter_server -> pyzmq[version='>=24'] -> libsodium[version='>=1.0.16,<1.0.17.0a0|>=1.0.18,<1.0.19.0a0']
jupyter_client -> pyzmq[version='>=23.0'] -> libsodium[version='>=1.0.16,<1.0.17.0a0|>=1.0.18,<1.0.19.0a0']
libsodium

Package backports conflicts for:
entrypoints -> configparser -> backports
tornado -> ssl_match_hostname -> backports
pywinpty -> backports.shutil_which -> backports
ipython -> backports.shutil_get_terminal_size -> backports
matplotlib -> backports.functools_lru_cache -> backports

Package webencodings conflicts for:
webencodings
bleach -> webencodings
nbconvert -> bleach -> webencodings[version='>=0.4']
python=3.7.0 -> pip -> webencodings
pip -> webencodings
```

The image above shows the conflicting versioning with existing libraries - Ryu

Chapter 3 | Reflection: Execution

What went Well, Bad & Lessons learned?

What went Well

As I already had VMware Pro Installed with Ubuntu 18.04 I proceeded to use the VM on my laptop to flash the robot (https://www.vmware.com/sg/products/workstation-pro/workstation-pro-evaluation.html). After that I installed Nvidia SDK to flash the latest drivers and packages to the TX1. (https://developer.nvidia.com/sdk-manager)

The second part of the hardware setup consists of flashing the Nvidia TX1 and the Arduino board. To flash the software, the first step is to download an ubuntu distribution preferably 18.04 on the Host Laptop or Virtual Machine (VM) on your own laptop.

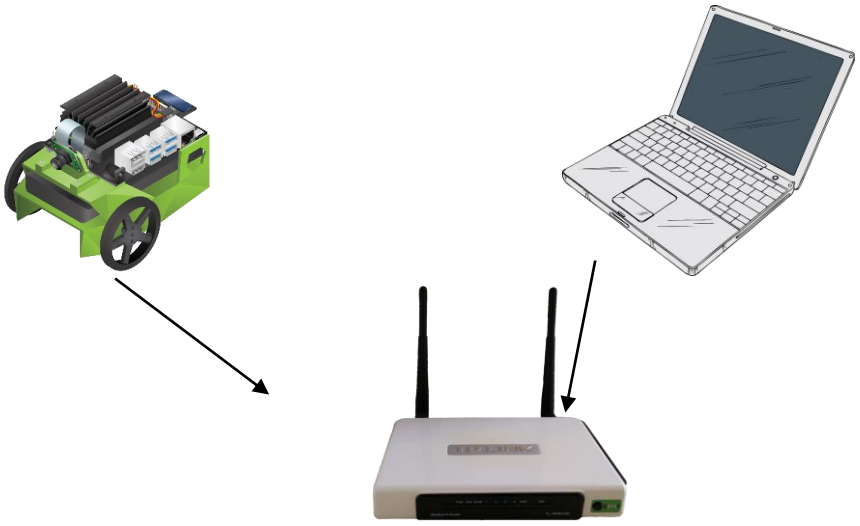
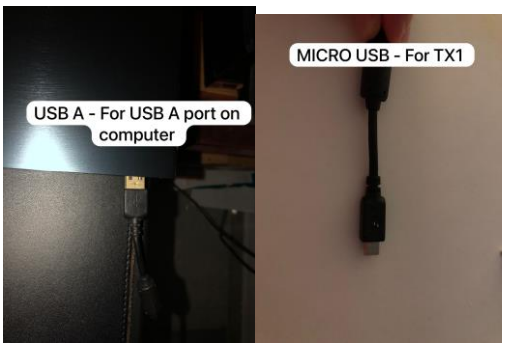


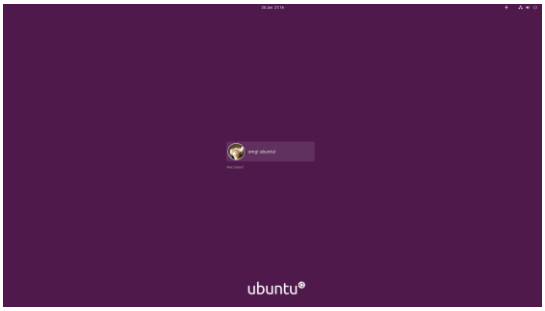
Figure 2: The figure above shows the Robot (Left) we are working with, and the Host Machine (Right) used to control or set up the robot.

To manage and configure the robot, follow these steps:

After completing the installation and setup of the Ubuntu software, the next essential step is to install the Nvidia SDK on the Host VM for flashing the robot. It's crucial to note that VMware should be used instead of VirtualBox to avoid compatibility issues with USB ports. VirtualBox VMs cannot flash the Nvidia TX1 due to a USB bug. This setup typically takes around 40 minutes for a fresh download.



To flash the TX1 one would need a Micro USB connected and a computer with VMware. Connect the USB A port to the computer and the Micro USB to the TX1.



Example of a ubuntu login page: It should look something like this.

Once the robot has been successfully flashed, it should display an Ubuntu login page, running Ubuntu 18.04. Ensure that both the Host and the machine have matching Ubuntu versions to minimize the chances of encountering bugs.

After Flashing one should configure the host Wi-Fi to be on the same network as the TX1 bot. Note that the ideal Wi-Fi connection should have connection to the internet. This allows for packages to be easier to download on both host and TX1.For the configuration of the robot the Wi-Fi network I will be using would be my home network. However, for this project we would be sticking with what was given, a Local Portable Wi-Fi Network. This is because the school Wi-Fi network security implementations block the connection.

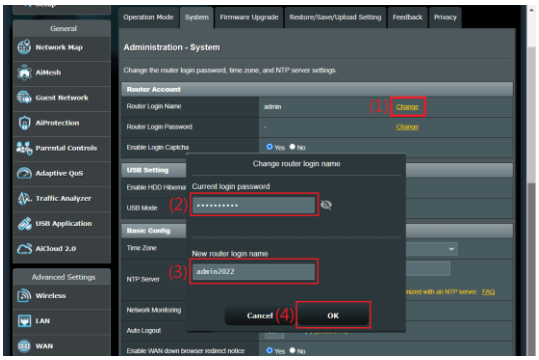
The video was specifically chosen to be in the AV1 format to save space and transcoding time (Essential in our robot as well as future implementation as cloud storage is expensive when dealing with large volumes of data). Although as compared to .H264 I am losing some clarity, it is minimal and will look lose less to humans which are examining the video. The implementation of this process is in the submission folder where it shows the C++ code required to create the video. The implementation of this process is in the submission folder where it shows the C++ code required to create the video.

```
vid_path = base_path + ".avi";
data_path = base_path + ".csv";
```

This code piece is found under the drive_train.cpp file.

After flashing, it's essential to configure the host Wi-Fi to be on the same network as the TX1 bot. Ideally, this Wi-Fi connection should also have internet access, simplifying the process of downloading packages on both the host and TX1. For one to configure the robot with ease one can opt for a less restricted network like that of our homes.

In our case, we'll be configuring the robot to connect to a Local Portable Wi-Fi Network, as provided, instead of my home network. This choice is necessitated by the security measures implemented on the school's Wi-Fi network, which unfortunately restricts the connection.



Above shows the login screen when I connect it to the Wi-Fi network in the browser. The password is ‘admin’, and the user is ‘admin’. The TX1 robot would be using the IP address of 192.168.1.158 (New) or 192.168.1.116 (Old). While the Host machine would be that of 192.168.1.235.

To test the connection, I sent a ping command: ping 192.168.1.158 and received a response indicating that I have successfully connected to the Wi-Fi network and can reach the TX1.

While the set up initially was straight forward, it soon became clear that the packages were of age. Some of these packages were discontinued, aged several years, and had numerous dependencies that clashed with other packages. Additionally, we encountered issues related to camera access, necessitating modifications. To the cameras configuration file located under the folder “USB_Cam” This was solved in the **Caffe Models & OpenCV** section below.

Furthermore, I faced a documentation gap midway through our project. There were limited tutorials available that specifically addressed the setup we were working towards. After quite a while of trial and error I realized that I was required to build the packages inside the **src** directory. This meant the transfer of our **jetlabs** files to the source directory.

Lastly there were miscellaneous issues like general package errors which took a few hours to solve. This was solved in the **General Errors** section below.

Chapter 3.5 | Trouble Shooting, Modelling & Difficulty

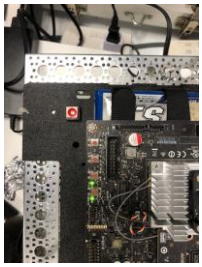
Challenges and Difficulties Encountered Chronologically:

Software Difficulties: Caffe Models & OpenCV
To start with, after connecting to the Wi-Fi Network, install OpenCV 3 by using the script provided by luke-han on GitHub (https://gist.github.com/luke-han/55047815e0dc2c446f5e184d9d0201cc).

This is necessary because the standard installation of OpenCV3 may not be recognized as a properly installed package in the system, leading to issues like "Package not found" or errors when trying to use OpenCV. By building the software from the source code, it allows the system to configure paths correctly and compile the necessary code for using OpenCV3. While the space required remains the same as a standard installation, the installation process through source code takes longer. After successfully installing OpenCV3, proceed with the installation of Caffe, following the instructions provided at https://caffe.berkeleyvision.org/install_apt.html.

Once both Caffe and OpenCV are installed, use the provided script 'rosjet_install.sh' and make sure to switch the ROS package to 'melodic' instead of 'kinetic,' as this is the version we will be working with. 'rosjet_install.sh' is a bash script designed to install various libraries.
Following the library installations, one will have a 'catkin_ws' directory. To complete the setup, run 'catkin_make' in the directory and then execute 'source ~/.bashrc' and ‘source devel/setup.sh’ to configure the robot effectively.

Hardware Issues 1 (New Robot)



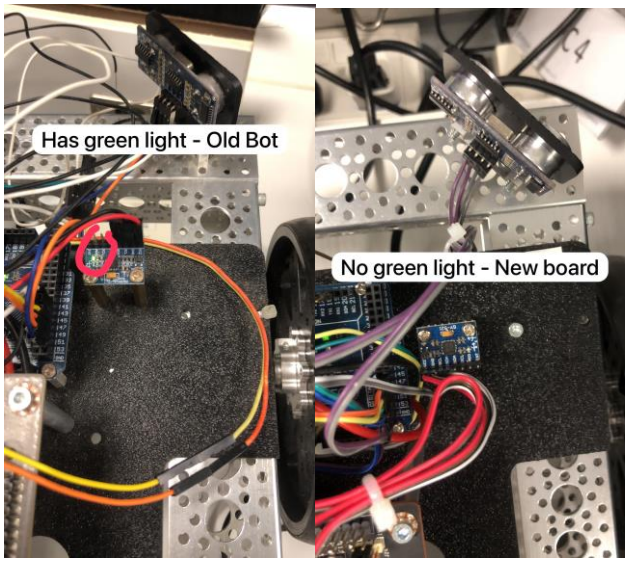
The figure above shows the component, the red button that was the first hardware issue.

During weeks 4 and 5, while configuring the system, I encountered a notable issue where the Output Current of the motor was consistently reading as 0. This raised concerns about potential problems with either the battery or the wiring. Initially, I suspected loose wire connections, as I had observed some during an earlier examination of the TX1. However, even after ensuring that all wires were securely connected and properly seated, the issue persisted when running the command "rostopic echo /arduino/motor_right_current."

After several days of troubleshooting both software and hardware, including checking for code errors (some minor ones were found, e.g., lab8_autonomous_vehicle instead of lab4), I finally identified the root cause: a red switch required to power on both the robot and the TX1 board. Upon raising the issue, we promptly visited Sim Lim Square to procure the necessary switches, with Mr. Tan graciously providing the funds. While the robot was repaired, I proceeded to develop and test a Reinforcement Learning model to see if it’s of any better than Ryus currently proposed model.

Reinforcement Learning Model Development:
Exploring the application of Reinforcement Learning to distinguish between different signs yielded results comparable to the CNN-based model. However, it exhibited shortcomings in terms of both accuracy and training time. For instance, training a Deep Q-Network (DQN) took significantly longer compared to a CNN, and the results were no better than those achieved with standard or pre-trained CNN models. Consequently, we decided against implementing Reinforcement Learning, especially considering our robot's limited processing power and the increased training time.

Hardware Issue 2 (New Robot)



The picture above shows the accelerometer board not lighting up.

After the switch had been fixed by Dr Yap, there was yet another issue with the Accelerometer board. This might be due to loose wiring but even after making sure they are well seated and are in the correct position, the board still does not light up. This would not be a software issue as all boards are supposed to light up when power is supplied. The boards that are supposed to light up are the Accelerometer, Arduino, Polulu and TX1 board. There were suggestions to switch the old board with the new robot's board. However, it comes with the risk of both robots not working, thus switching the board would not be implemented.

Hardware Issue 3 (Old Robot)
While all the boards were working on the old robot, the right wheel, however, was not working. The old robot had a wheel that could not move. However, Dr Yap manages to fix the robot as he found that the wires were not connected to the right motor, essentially fixing the robot. However, once the wires were connected the speed of the wheel decreased. This is a smaller issue as the solution would be to choose a stronger battery. However, the current set up for this TX1 should be sufficient for a demonstration.

Enhancing the Robot's Video Recording and Cloud Integration:
One of our objectives was to empower the TX1 to record and transmit videos to the cloud. This capability would have allowed us to analyze the robot's predictions more effectively, thereby enhancing our model development process. Unfortunately, this endeavor was hampered by the inability of the Wi-Fi network to connect to the internet due to security restrictions imposed by the school's network. (Note: It is possible on networks that have less strict security policies, as mentioned above)

General Errors (New Robot)
If one encounter camera issues, it's likely related to the camera settings. Typically, these settings result in warnings rather than major problems. One example is the "White Exposure" issue, where white exposure is a camera setting that establishes the true color of white. This serves as a reference point for all other colors the camera captures. It aids the camera in adapting to lighting conditions to accurately represent the colors it sees. Fortunately, for our robot, this setting won't have a significant impact since it will be operating in well-lit environments. In the case of general errors, there may be runtime issues or other unforeseen problems when running the TX1. If you encounter errors not covered in this text, such as camera exposure or package compatibility issues, the most probable cause might be system-related runtime issues. These can often be resolved by recompiling the package using the commands listed below:

- rm -rf build devel – remove packages.
- sudo reboot
- cd catkin-ws
- catkin_make – rebuild packages.
- source ~/.bashrc – update the bashrc file and initialize
- source devel/setup.sh

Chapter 3.5 | Finishing Touches

The TX1 required a change in several of the launch configuration file as well as the source code.

```
<launch>
  <include file="$(find jet_bringup)/launch/jet_real.launch"/> <-- make sure this line is uncommented
  <node name="drive_inference" pkg="lab4_autonomous_driving" type="drive_inference"/>
</launch>
```

Figure 4: Shows the launch inference configuration file codes that need to be uncommented to launch the inference file needed for the robot to use the camera and recognise the signs.

```
/*
if (pred.first == "FORWARD") {
  vel_msg.linear.x = 0.7;
  vel_msg.angular.z = 0.0;
}
else if (pred.first == "LEFT") {
  vel_msg.linear.x = 0.2;
  vel_msg.angular.z = -0.5;
}
else if (pred.first == "RIGHT") {
  vel_msg.linear.x = 0.2;
  vel_msg.angular.z = 0.5;
}
vel_pub.publish(vel_msg);
*/
```

Figure 5: Shows the lines to be uncommented. These lines control the movement of the robot.

```
string model_file   = base_path + "/neuralnetwork/deploy.prototxt";
string trained_file = base_path + "/neuralnetwork/models/train_iter_157.caffemodel";
string mean_file    = base_path + "/resources/data/mean_image.binaryproto";
string label_file   = base_path + "/neuralnetwork/labels.txt";
classifier = new Classifier(model_file, trained_file, mean_file, label_file);
```

Figure 6: Shows the Path to the files that need to be changed to accommodate to your new model and protext files. (The picture above shows what need to be changed, this needs to be one's file directory)

```
FORWARD
LEFT
RIGHT
```

Figure 7: Shows the labels.txt file. The order of the targets needs to be changed to accommodate to our new model as we have trained the targets differently.
Switch the position of Left with Position Right

File Edit Format View Help

```
<launch>
  <include file="$(find jet_bringup)/launch/jet_real.launch"/>
  <node name="drive_train" pkg="lab8_autonomous_driving" type="drive_train"/>
</launch>
```

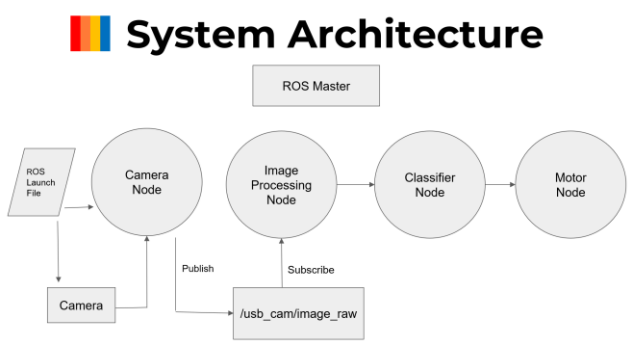
Figure 8: Shows a Typo, there is no such file as lab8_autonomous_driving

Chapter 4 | Reflection: Deliverable

In this section write up about your reflections about the final deliverable that you produced for your project.

New Robot (Weaknesses)
While the intended purpose of the robot was to do image recognition and move according to the signs, only about 50-75% of the project was successful. This meant that while the software of the robot, the ROS framework and the Modelling worked the wheels of the robot could not be moved despite having power and direction, due to the failure of the accelerometer board.

What went well was that the new robot could successfully recognise the signs and steer the robot in the correct direction through the TX1 and ROS Framework with a high degree of accuracy. The old robot was also able to successfully navigate according to the given sign with relatively okay/moderate accuracy in the terminal, considering that it is a 720P camera with scaled down resolution in the data preprocessing part, the **Image Preprocessing Node**.



The image above shows the process as to how the robot was launched and how it makes its predictions. The Capture input of the camera was (320, 240,3) scaled down to (80,60,3) in the image processing node

What did not go well was the fact that our new robot could not move. This as mentioned above is mainly due to the hardware issues present on the newer robot. As mentioned above the accelerometer has problems turning on. Despite having power readings and Directional inputs to the Arduino board, due to the failure of the Accelerometer board, the board is not able to send the correct electrical signals to the wheels of the robot.

One oversight we made was when we trained the robot on signs with colour. For our models, colour will play an important role in prediction. Converting an image to grayscale would have allowed our model to focus more on the shape and orientation of the sign rather than the colour of the sign. This could be remediated with an OpenCV library to change the captured and trained image to grayscale (320,240,1) instead of (320,240,3).

Old Robot (Strengths)

With the robot able to respond, react and move to the signs inferred, the robot's intended purpose is fulfilled (100% Completion Status). While the new robot suffered from hardware issues, the old robot in contrast had little hardware issues. This allowed for a fast set up with little to no errors. (The hardware error on the wheel was also easily fixed with Dr Yap's help)

Old Robot (Weakness)

Like the newer robot, one oversight we made was when we trained the robot on signs with colour. For our models, colour will play an important role in prediction. Converting an image to grayscale would have allowed our model to focus more on the shape and orientation of the sign rather than the colour of the sign. This could be remediated with an OpenCV library to change the captured and trained image to grayscale instead or (320,240,1).

Review the process you took to produce the final deliverable. Was it sufficient?

The process I followed to contribute to the final deliverable for our major project involved several key steps and stages. I believe that it was sufficient as the old robot successfully completed the goal of the project. It was but the hardware issues that hampered with the newer robot. I believe that given more time the new robot hardware difficulties could be fixed and the new robot will function just as well as the older robot.

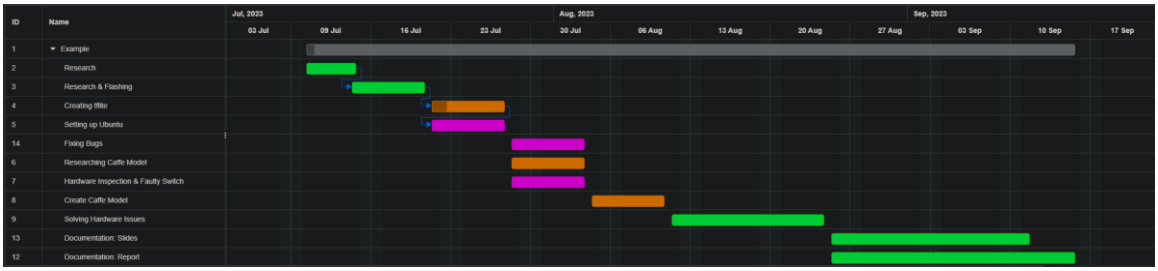
For the project, I began by setting clear objectives, which were to set up the base for the robot to perform image recognition using a deep learning. While working on the project the decision and process (Finding and confirming errors via GUI or Physical Inspection) to raise the issue was also made early which I think saved us quite some hassle at the end of the project. Thus, the process I took (to me) was sufficient in part of completing the project.

In evaluating the effectiveness of the process, it's think that it allowed my team to achieve our project's primary goals. We successfully programmed the robot to recognize the specified signs, and we gained valuable insights into the fields of robotics, deep learning, and hardware/software integration. However, due to multiple hardware issues and delays, the robot as mentioned above was not able to move. Without the available parts and approval to change or swap the parts we were not able to fix the robot in time.

Despite this setback, throughout the project, I have learned and improved my Linux proficiency and familiarized myself with the ROS framework, all of which contributed to my growth and skill development. In terms of meeting deadlines and milestones, the process was generally effective, although I did encounter some delays caused by the aforementioned challenges. Another area where I think my team did well was the communication. When moving ahead with plans we would always share our findings and suggest approaches to our problems.

In summary, with reference to the newer robot the final deliverable did not meet our overall project's objectives and requirements, but the older robot did and as such I think that the process, we took was sufficient. The robot successfully recognized and responded to the specified signs, demonstrating the practical applications of deep learning and robotics but was unable to move due to the Accelerometer Board. The process I followed to produce the final deliverable was generally effective (I did not expect delays due to hardware issues), but not without its challenges as mentioned above. With lessons learned from this project, I am better equipped to improve future endeavours in the field of robotics and deep learning.

Include a workload distribution among the team members.



Maximilian – Hardware integration and setup. While the robot was getting fixed also took the time to try Re enforcement Learning.

Ryu – Model Building & Creation

Chapter 5 | Conclusion

In this section write up about your reflections. We are looking for evidence and reflection of learnings achieved.

Our major project provided valuable insights and lessons that shaped our understanding of Linux, system constraints, hardware inspections, ROS, and software flashing. We encountered challenges, but they ultimately contributed to our growth and skill development. The project illuminated the importance of meticulous hardware inspections, efficient software management methods in a limited environment, and the significance of communication within a team. While the project did not fully meet our objectives due to hardware issues, it highlighted the need for constant adaptation and troubleshooting in robotics and deep learning projects.

Overall lessons learnt and insights from doing your major project.

Expect delays and prepare for them.

When starting this project, I started with the assumption that the hardware as well as software were working. This did not age well as there were multiple hardware and software issues. As there was an assumption that it was going to be smooth sailing, preparation for hardware or software issues was placed at the lowest priority when preparing and reading about the project. This meant that once the hardware and software issues arrived, I was confused. Some Issues were hardware based and some software based. This resulted in a lot of restarting or re-flashing the TX1 or Arduino, checking for loose sockets, checking the log files of the robot to see if certain parts are receiving power and checking the wiring for the Arduino Board. Thus, for future projects it's better to read up on the past errors or problems one might encounter from forums, subreddits or Nvidia Documentation instead of preparing or learn to implement more complicated features.

Expect things out of your control and learn to make the most out of your time.

Time is not a commodity and making the most out of your time approaching the problem in a different way or learning a tool might pay off. While the robot was under Dr Yap servicing, I found it was better to help with the modeling. Instead of using traditional CNN methods I decided to use Re-enforcement learning. Although it was not as good as a traditional CNN method was, I learnt was that RL learning for image recognition was not ideal as it was longer and more hardware intensive, which was not ideal for our edge device as it requires as much resource headroom as possible.

Raise Persistent Issues Even if not Confident.

It's wiser to voice concerns about an issue rather than remain silent. While configuring the robot, I took some time to inform the team about potential hardware issues. I hesitated because I was considering the possibility of it being a software problem or a mistake in the flashing or configuration process. Despite my lack of confidence in pinpointing the exact issue, I decided to share my concerns with the group rather than prolong my deliberation. This decision ultimately proved beneficial, as it led to a quicker response since more team members were aware of the situation. It turns out that my initial suspicions about the problems were mostly accurate, which saved the group a significant amount of time. This project taught me the importance of raising concerns promptly, even when I'm uncertain about the specifics.

How can things be done differently if you are to do it again?

With reference to the first point on the overall lesson learned, I would instead of focus on the more advance implementation of ROS or modelling, should focus instead on the problems or errors people have faced over the years while developing such robotics. Reading Similar projects to what was given to us, or simply running or doing a small ROS project would help tremendously in understanding and troubleshooting.

For software integration with ROS like our models one way to practice ROS or such implementation would be to use GAZEBO to simulate the robot. If I had to do it again, and do it faster for the project, I would have tried to find a similar project and implement it before the start of the project as this would allow me to better understand the architecture of ROS and how we could implement Deep Learning into it faster.

For the hardware, it is a different story as there is no hardware for us to test out before going into the project. What I would have done differently is first to do a check on all boards to see if they are working. It would just be a simple inspection or power on.

If I were to approach a similar situation again, I would choose not to doubt myself or engage in excessive analysis. Instead, whether the issue is minor or significant, I would opt to share such concerns along with my rationale for considering them problematic. I would involve my group members in a collaborative effort to identify and address these issues, rather than trying to tackle them in isolation.

For instance, instead of presenting an issue with a simple statement like, "This is the problem," I would take a more informative approach. I might say, "This could potentially be the problem because... or it might be related to this particular component..." By providing context, drawing from my experiences, and offering my insights into the possible causes, alongside considering input from others in the group, I believe it would not only enhance communication but also deepen our collective understanding of the issue and its underlying factors.

Chapter 6 | Future Enhancement

Hardware Improvements (Hardware)

- Implement an Enhanced Camera Setup by integrating additional cameras. This will significantly improve data capture capabilities for the robot, enhancing its perception and decision-making capabilities.

- Upgrading to Lidar Sensors. This upgrade will greatly enhance the robot's environmental mapping and awareness, contributing to safer and more efficient navigation. LiDAR technology will empower it with an expanded spatial understanding, allowing for more precise and efficient navigation through its surroundings.

- Utilizing the ESP 32 development board instead of the Arduino Shield. Compared to the Arduino board the ESP 32 board supports machine learning applications to a higher degree. With the use of the ESP 32 board the development of the product would be easier to maintain and attain higher performance at the same time. (Supports Wi-Fi and Bluetooth as well). It is viable as it is cheap at $12.

Model Improvements (Software)

- Advanced Models include incorporating larger models like Efficient Net Models that although take up more space, uses lesser resources to infer. These advanced models will provide the robot with a higher degree of decision-making precision as well as speed, making it more adaptable to more complex signs.

- Adding Object Detection. Currently, the model we have trained can only do image recognition on the whole image. Hence, this can cause it to malfunction when there are no signs or if the sign is too far away. It will try to force a prediction out. Hence, with object detection included in models such as YOLO (You Only Look Once) it will help to differentiate between the many objects in its view for greater situational awareness for more complex inferences.

MP Supervisor Contribution

I would like to express our sincere gratitude for the exceptional support and mentorship extended to us by our supervisor, Mr. Tan Sio Poh. Throughout our journey with the robot, Mr. Tan's expertise and guidance have been invaluable. Not only did he provide us with the robot itself, but he also took the time to thoroughly demonstrate its functionality, ensuring that we were well-prepared to operate it effectively.

Mr. Tan's contributions went even further as he generously shared his insights into the robot's coding, proving to be instrumental in the success of our project.

During a critical moment when our robot's switch malfunctioned, Mr. Tan exhibited exceptional dedication by proactively seeking assistance from Dr. Yap to resolve the issue promptly. This action underscored his unwavering commitment to our success and the seamless progress of our work.

Furthermore, Mr. Tan's support extended beyond the hardware. He graciously provided us with the base code, which significantly expedited our project's development. This solid foundation allowed us to build upon existing knowledge and efficiently achieve our research objectives.

References/ Bibliography

262588213843476. (n.d.). *Install opencv 3.3 in ubuntu 18.04*. Gist. https://gist.github.com/luke-han/55047815e0dc2c446f5e184d9d0201cc

Bvlc. (n.d.). BVLC/Caffe: Caffe: A fast open framework for deep learning. GitHub. https://github.com/BVLC/caffe.git

Conversion from TensorFlow to caffe - creating the architectures - part 1 of 3. YouTube. (2018, April 4). https://youtu.be/9iJheyF7x4Y

Caffe. (n.d.). https://caffe.berkeleyvision.org/

Cloud computing services - amazon web services (AWS). (n.d.). https://aws.amazon.com/

Enterprise open source and linux. Ubuntu. (n.d.). https://ubuntu.com/

F., F. (n.d.). Av1: Unleashing a new era of streaming in the industry. LinkedIn. (2023, August 28). https://www.linkedin.com/pulse/av1-unleashing-new-era-streaming-ott-industry-francesco-farruggia/guest_blog

OpenCV. (2023, August 19). https://opencv.org/

Incremental and reinforced learning for image classification. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/02/incremental-and-reinforced-learning-for-image-classification/

Nvidia Drive Labs NVIDIA DRIVE Labs

Pytorch. PyTorch. (n.d.). https://pytorch.org/hub/ultralytics_yolov5/

Ros-Drivers. (n.d.). Ros-drivers/USB_CAM: A ROS driver for V4L USB cameras. GitHub. https://github.com/bosch-ros-pkg/usb_cam.git

ROS Robot operating system. ROS. (n.d.). https://www.ros.org/

Tan, M., & Le, Q. V. (2020, September 11). EfficientNet: Rethinking model scaling for Convolutional Neural Networks. arXiv.org. https://arxiv.org/abs/1905.11946

World leader in AI computing. NVIDIA. (n.d.). https://www.nvidia.com/en-sg/

Appendix A1: DEED OF ASSIGNMENT

In the Submissions

Appendix A2: Terms of Reference (TOR)

In the Submissions

Appendix A3: Project Plan

Clearer Version as attached in the Submissions folder.

Legend:

Green : Together

Pink : Max

Orange : Ryu

Appendix A4: Weekly Progress Reports

In the submissions folder