

RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION

RECOGNITION

BY

TAN YONG MING

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2023

REPORT STATUS DECLARATION FORM

Title: RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION
RECOGNITION


Academic Session: MAY 2023

I TAN YONG MING


(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.



(Author's signature)

Verified by, 

(Supervisor's signature)

Address:

B-832, JALAN KUBUR
TANAH MERAH
17500, KELANTAN

____ Dr. Aun Yichiet _____
Supervisor's name

Date: 12/9/2023

Date: ____15/9/2023____

Universiti Tunku Abdul Rahman			
Form Title : Sample of Submission Sheet for FYP/Dissertation/Thesis			
Form Number: FM-IAD-004	Rev No.: 0	Effective Date: 21 JUNE 2011	Page No.: 1 of 1

FACULTY/INSTITUTE* OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN

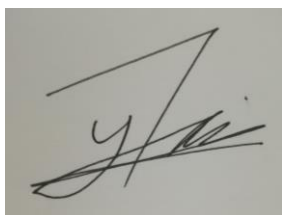
Date: 12/9/2023

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that TAN YONG MING (ID No: 20ACB05117) has completed this final year project/ dissertation/ thesis* entitled "RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION" under the supervision of Dr. Aun Yichiet (Supervisor) from the Department of Computer and Communication Technology (DCCT), Faculty/Institute* of Information and Communication Technology .

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,




Tan Yong Ming
(Student Name)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled “**METHODOLOGY, CONCEPT AND DESIGN OF A 2-MICRON CMOS DIGITAL BASED TEACHING CHIP USING FULL-CUSTOM DESIGN STYLE**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :  _____

Name : _____ Tan Yong Ming _____

Date : _____ 12/9/2023 _____

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to the following people for their invaluable support and contributions to this project:

First and foremost, I would like to thank my supervisor, Dr. Aun Yichiet, for his constant guidance, support, and encouragement throughout the project. Without his expertise and insights, this project would not have been possible. Dr. Aun Yichiet has given me this bright opportunity to engage in a Computer Vision project, and I am grateful for his unwavering support and encouragement. A million thanks to you, Dr. Aun Yichiet, for your invaluable contributions to this project.

To a very special person in my life, Carolyn Tan, for her remarkable patience, unwavering support, and love for being a constant source of strength and inspiration for me. Finally, I am deeply thankful for the love, support and continuous encouragement that my parents and family have shown me throughout the course of my project. Their belief in me and their constant motivation have been instrumental in my success.

ABSTRACT

The challenge of navigating complex road systems, especially in urban environments, necessitates innovative solutions to enhance driver safety and confidence. This project explores the development of a computer vision system aimed at detecting road junctions in real-time, providing drivers with timely and accurate guidance. The system comprises two key components: junction detection and image similarity comparison. Traditional object detection metrics, such as Intersection over Union (IOU), are ill-suited for the intangible nature of junctions. As a solution, we propose the use of accuracy as an alternative evaluation metric to assess the model's ability to classify frames as 'junction present' or 'no junction.' Ground truth labeling of test images as '1' or '0' is performed, facilitating accuracy evaluation. This project's computer vision model demonstrated significant progress in junction detection accuracy, enhancing driver safety and navigation. Challenges encountered provide valuable insights for future refinement, particularly in optimizing cloud-based processing efficiency. The findings contribute to the advancement of intelligent navigation systems in complex urban environments.

TABLE OF CONTENTS

TITLE PAGE	i
REPORT STATUS DECLARATION FORM	ii
FYP THESIS SUBMISSION FORM	iii
DECLARATION OF ORIGINALITY	iv
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	x
LIST OF TABLES	xi
LIST OF ABBREVIATIONS	xii
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement and Motivation	1
1.2 Objectives	1
1.3 Project Scope and Direction	2
1.4 Contributions	2
1.5 Report Organization	3
CHAPTER 2 LITERATURE REVIEW	4
2.1 Previous Works using Tensorflow Object Detection Models	4
2.1.1 Weakness	5
2.2 Previous Works using Intangible Object Detections	6
2.2.1 Intersection Detection	6
2.3 Previous Works on Raspberry Pi	9
2.4 Previous Works using Cloud Computing on Object Detection Models	9
2.4.1 Amazon Cloud Service	10
2.4.2 Weakness	11

CHAPTER 3 SYSTEM METHODOLOGY/APPROACH	13
3.1 Methodologies and General Work Procedures	13
3.1.1 Image Acquisition	13
3.1.2 Junction Detection	13
3.1.3 Audible Feedback	14
3.1.4 Cloud Computing	14
3.2 Study Area and Data	15
CHAPTER 4 SYSTEM DESIGN	17
4.1 System Block Diagram	17
4.2 System Components Specifications	17
4.2.1 Camera + Raspberry Pi	17
4.2.2 Amazon S3	17
4.2.3 Amazon EC2	18
4.2.3.1 Amazon EC2	18
4.2.3.2 Image Similarity Detection Model	19
4.2.3.3 Google Directions API	19
4.2.3.4 Google Street View API	19
4.2.4 Raspi Audio	19
CHAPTER 5 SYSTEM IMPLEMENTATION	20
5.1 Hardware Setup	20
5.2 Software Setup	21
5.2.1 Jupyter Notebook	21
5.2.2 TensorFlow	21
5.2.3 Operating system of Raspberry Pi	22
5.2.4 Amazon EC2	22
5.3 Setting and Configuration	22
5.3.1 Jupyter Notebook	22
5.3.2 TensorFlow	22
5.3.3 Raspberry Pi OS	23
5.3.4 Amazon S3	23

5.3.5	Amazon EC2	23
5.4	System Operation (with Screenshot)	25
5.4.1	Video Acquisition	25
5.4.2	Video Frame Extraction	25
5.4.3	Sending Frames to Amazon S3	26
5.4.4	Connect to Amazon EC2 to run Python Script	26
5.4.5	Handle Output to get Comparison Result	27
5.4.6	Trigger Raspi Audio	27
5.5	Implementation Issues and Challenges	28
5.5.1	Migration of code to Raspberry Pi	28
5.5.3	Computation Limitation of Raspberry Pi	28
5.5.3	Internet Connectivity Issues	29
5.6	Concluding Remark	29
CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION		30
6.1	System Testing and Performance Metrics	30
6.2	Testing Setup and Result	30
6.3	Project Challenges	31
6.4	Objectives Evaluation	31
6.5	Concluding Remark	32
CHAPTER 7 CONCLUSION AND RECOMMENDATION		33
7.1	Conclusion	33
7.2	Recommendation	33
REFERENCES		35
APPENDIX		37
WEEKLY LOG		52
POSTER		58
PLAGIARISM CHECK RESULT		152
FYP2 CHECKLIST		160

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1.1	Framework for building and training a deep learning model to detect street signs using Tensorflow [6]	5
Figure 2.2.1.1	Rotated road bounding box detection results; (a) nonintersection case, (b) intersection case with rotated bounding box, and (c) unrotated bounding box.[4]	7
Figure 2.2.1.2	Intersection Detection Algorithm in [5]	8
Figure 2.2.1.3	The roundabout intersection. A top-view. The red arrows in (b) and (c) point to intersections.	8
Figure 2.4.1.1	Cloud vs Local Processing Time [12]	11
Figure 3.2.1	Study Area	15
Figure 4.1.1	System Block Diagram	17
Figure 5.1.1	The setup of the Raspberry Pi 4 Model B	21
Figure 5.3.5.1	The configuration of AMI in EC2	23
Figure 5.3.5.2	The configuration of instance type in EC2	24
Figure 5.3.5.3	The configuration of EBS in EC2	24
Figure 5.4.2.1	Input Video	25
Figure 5.4.2.2	Output Frames	25
Figure 5.4.3.1	Code Snippet to send images to Amazon S3	26
Figure 5.4.3.2	Amazon S3 Console	26
Figure 5.4.4.1	Amazon EC2 Console	27
Figure 5.4.5.1	Output from the EC2 python script	27
Figure 5.4.6.1	Code snippet for Handling Output	27

LIST OF TABLES

Table Number	Title	Page
Table 3.2.1	Summarisation of Dataset	16
Table 5.1.1	Specifications of laptop	20
Table 5.1.2	Specifications of Raspberry Pi	20
Table 5.1.3	Specification of Camera	21
Table 5.2.1.1	Specifications of Jupyter Notebook	21
Table 5.2.2.2	Specifications of TensorFlow	21
Table 5.2.3.1	Specification of Operating System of Raspberry Pi	22
Table 5.2.4.1	Specification of Amazon EC2	22
Table 6.2.1	Results of Model	31

LIST OF ABBREVIATIONS

<i>API</i>	Application Programming Interface
<i>CPU</i>	Central Processing Unit
<i>OS</i>	Operating System
<i>RAM</i>	Random Access Memory
<i>AWS</i>	Amazon Web Service
<i>Amazon S3</i>	Amazon Simple Storage Service
<i>Amazon EC2</i>	Amazon Elastic Compute Cloud
<i>EBS</i>	Elastic Block Store
<i>FPS</i>	Frame Per Second
<i>SSH</i>	Secure Socket Shell

Chapter 1

Introduction

Over the years, the domain of computer vision has invested significant efforts in the analysis and assessment of road condition. Given that majority of society heavily relies on road networks for their daily activities, continuous research endeavours aim to improve the convenience and efficiency of road usage. Within this extensive body of research, road navigation stands out as a crucial domain to investigate and enhance, given its indispensable role in our lives. In this chapter, I will present the background and motivation driving this research, elucidate its contributions to the field, clarify the rationale for pursuing this study, and provide an overview of the proposal's structure.

1.1 Problem Statement and Motivation

The motivation behind this research is to enhance road navigation for users by harnessing computer vision technology to accurately identify junctions and intersections in real-time. As reliance on GPS navigation continues to grow, drivers often encounter difficulties in precisely recognizing and navigating these crucial points, resulting in missed turns, incorrect routes, and potential safety risks. This study seeks to address these challenges by developing a real-time computer vision model using TensorFlow and RetinaNet. These models are designed to detect junctions and intersections from live street images captured by a camera connected to a Raspberry Pi device. The primary goal is to provide drivers with instantaneous navigation support, thereby improving their driving experience and reducing the likelihood of navigation errors. Furthermore, we envision future enhancements, including integration with an audible feedback system, utilization of the Google Map API, and the implementation of an image comparison model, to create a more comprehensive and refined navigation experience. However, it is important to note that these enhancements are beyond the scope of the current project.

1.2 Objectives

The primary objective of this project is to develop a computer vision model capable of detecting junctions in frames captured by the camera before they are sent to the subsequent block, which is the image similarity comparison model. The model can be divided into two parts. The first

part will focus on detecting junctions in the frames obtained from the camera on the Raspberry Pi. The second part of the model will be responsible for comparing the captured images with images of correct intersections from Google Street View images, but only under the condition that a junction has been detected in the previous block of the model. These operations will be done on the cloud for reasons that will be clarified later in the report.

1.3 Project Scope and Direction

The project involves several critical components and processes aimed at creating an effective intersection detection system. At its core, we've developed a computer vision model that accurately detects intersections. This model is hosted in the AWS cloud for efficient computation.

Within the AWS cloud environment, we have two key models: the Junction Detection Model and Image Similarity Detection Model. These models utilize Amazon's Elastic Compute Cloud (EC2) for complex calculations. The Raspberry Pi acts as an essential interface, triggering audio alerts based on similarity results from EC2.

The project's deliverables include a system that captures street scene images at one frame per second. These frames are securely stored in AWS Simple Storage Service (S3). Amazon EC2 retrieves the frames from S3 and performs junction detection and image matching. The Image Similarity Model compares captured images with Google Street View images, referencing the driver's route. Together, these components form a comprehensive system for intersection detection and driver assistance.

1.4 Contributions

In this project, several significant contributions have been made to develop an effective intersection detection system. Firstly, we meticulously configured the camera to capture images at a consistent rate of one frame per second, ensuring a continuous stream of visual data. Secondly, we successfully set up the Raspberry Pi, establishing seamless communication between the camera and the device to facilitate the transfer of frames.

One of the pivotal contributions lies in the development of a robust junction detection model. Leveraging a pre-trained model from TensorFlow as a foundation, we meticulously fine-tuned and tailored the model to our specific needs. Notably, we made crucial modifications to the classification head while preserving the box regression head, enabling the model to accurately detect junctions, a critical element in our intersection detection system.

Recognizing the computational limitations of the Raspberry Pi, we took a strategic step by relocating the trained junction detection model to the cloud. This migration ensures that the model can harness the immense computational power and resources available in the cloud environment, ultimately enhancing the efficiency and effectiveness of our intersection detection system. These contributions collectively form the cornerstone of our project's success, paving the way for further refinement and evaluation in the subsequent phases.

1.5 Report Organization

The details of this research are shown in the following chapters. In Chapter 2, some related backgrounds are reviewed. Then, Chapter 3 details the system methodology and general work procedures. And then, Chapter 4 highlights the system diagram and its components. Chapter 5 is about the system operation. Chapter 6 reports the result and feasibility of the proposed method. Lastly, Chapter 7 concludes the project.

Chapter 2

Literature Review

2.1 Previous Works using Tensorflow Object Detection Models

TensorFlow Object Detection models are a groundbreaking solution in the field of computer vision, offering a versatile and efficient approach to detecting objects within images and videos. These models are pre-trained on large datasets containing a wide variety of objects, enabling them to recognize and locate objects in real-world scenes accurately. The TensorFlow framework, developed by Google, serves as the foundation for these models, ensuring robust and reliable performance.

The Object Detection API represents an open-source toolkit constructed upon TensorFlow, designed for the training and deployment of object detection models. TensorFlow, an open-source library, is employed for a wide range of dataflow programming activities. The underlying TensorFlow platform is primarily written in C++, and it supports a Python or C++ API layer for ease of use and integration. [7]

One of the notable features of TensorFlow Object Detection models is their adaptability. They provide a starting point for tackling object detection tasks, allowing researchers and developers to fine-tune and customize the models to suit specific needs as fully utilised by [6] where the authors extract traffic sign data using Tensorflow Object Detection model. Figure 2.1.1 below displays their extensive use of Tensorflow. This flexibility is invaluable when addressing real-world applications, as different scenarios may require tailored solutions. Whether it's identifying pedestrians for autonomous vehicles, detecting defects in manufacturing processes, or monitoring wildlife, TensorFlow Object Detection models can be adapted to handle a wide range of challenges.

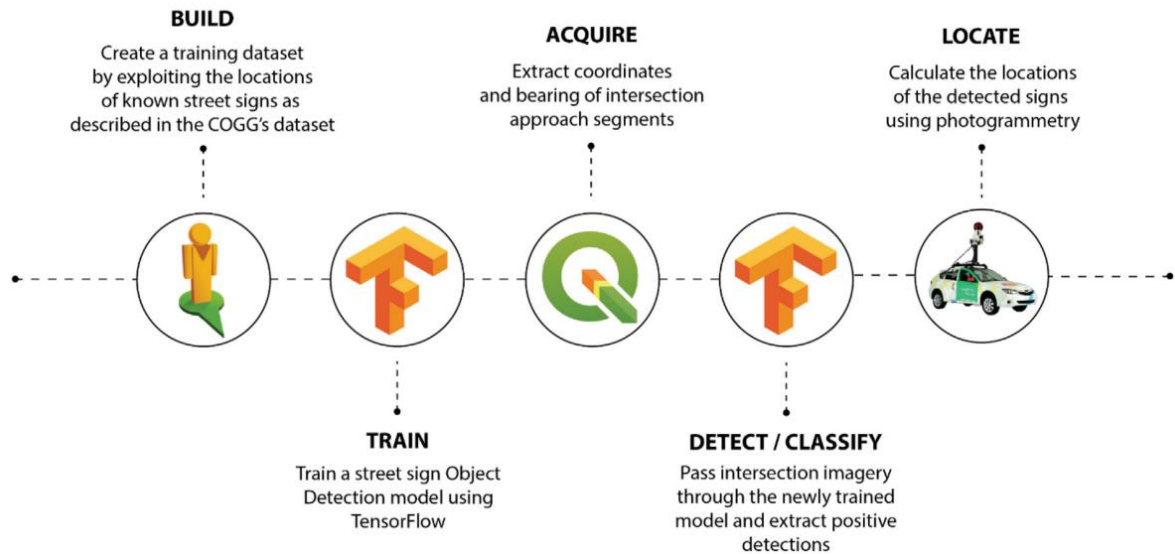


Figure 2.1.1 Framework for building and training a deep learning model to detect street signs using Tensorflow [6]

Furthermore, TensorFlow Object Detection models are compatible with various platforms and frameworks, making them accessible for deployment in diverse environments. This versatility has contributed to their widespread adoption in both research and industry, where they are applied to tasks such as object tracking, instance segmentation, and even more complex problems like scene understanding. Take [7] for example, they were able to identify the location of traffic lights with a moderate number of samples when using the TensorFlow approach. In summary, TensorFlow Object Detection models stand as a testament to the power of deep learning and its transformative impact on computer vision, empowering developers to create solutions that enhance safety, efficiency, and automation across various domains.

2.1.1 Weakness

As highlighted in [9], TensorFlow has several drawbacks as well, the first is: Although the COCO dataset offers a diverse collection of objects, it is not all-encompassing, and certain objects are absent from it. For instance, the dataset lacks images of junctions, so we manually obtained and labelled the junction dataset to be able to train the object detection model. Additionally, the developed assistive tool can identify humans as individuals but does not possess the capability to differentiate between specific individuals using facial recognition techniques.

2.2 Previous Works using Intangible Object Detections

2.2.1 Intersection Detection

Intersection detection plays a pivotal role in modern transportation and urban planning, addressing critical challenges that impact both traffic efficiency and road safety. In congested urban areas, efficient traffic management is essential for reducing gridlock and minimizing commute times. Intersection detection systems provide real-time insights into traffic conditions, enabling adaptive traffic signal control and efficient traffic management. By dynamically adjusting signal timing based on current traffic patterns, these systems help alleviate congestion and enhance the overall flow of vehicles, making urban transportation more efficient and eco-friendly.

Moreover, intersection detection contributes significantly to road safety. Intersections are notorious hotspots for accidents and collisions due to complex traffic interactions. Detection systems equipped with advanced sensors and computer vision technologies can provide drivers with warnings about potential conflicts and unsafe conditions. Additionally, they can aid in the development of collision avoidance systems, helping vehicles make split-second decisions to prevent accidents. As cities grow and traffic volumes increase, the importance of intersection detection becomes increasingly evident, offering not only convenience but also a crucial layer of safety in our daily commutes.

This domain is quite sufficiently explored by multiple researchers and there are quite a few techniques when it comes to intersection detection.

A related study is presented in [4] where they introduce a novel approach for intersection detection using deep learning and computer vision techniques. The system relies on a multi-task deep neural network that simultaneously performs two critical tasks: drivable area segmentation and rotated road bounding box detection as shown in Figure 2.2.1.1. The drivable area segmentation identifies regions where the vehicle can safely drive, while the rotated road bounding box detection precisely identifies branch roads within intersections. This method's effectiveness is demonstrated through experiments in real-world parking lot environments, where it outperforms traditional model-based techniques and facilitates successful intersection navigation by guiding the vehicle through the detected branch roads while avoiding obstacles.

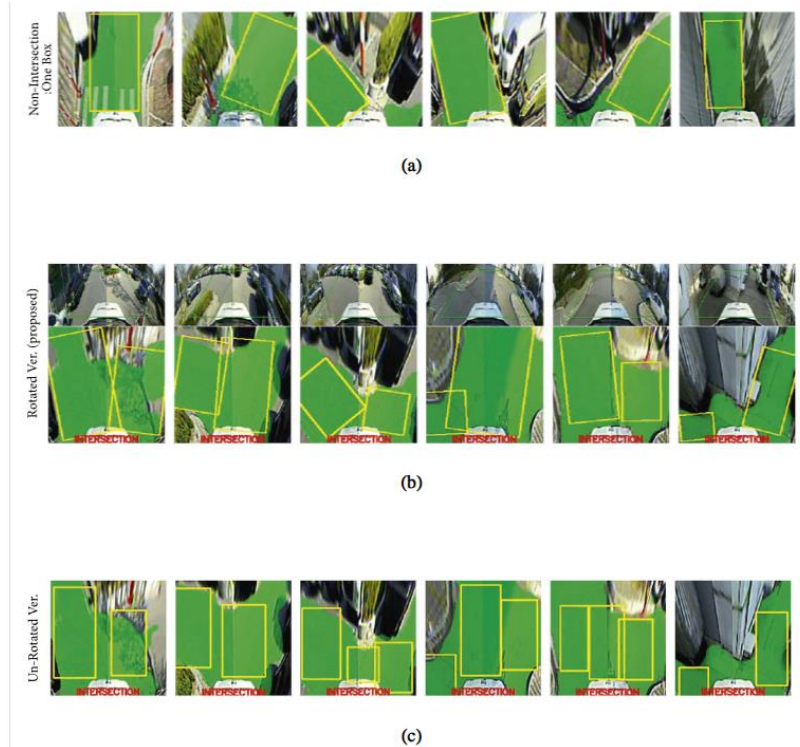


Figure 2.2.1.1 Rotated road bounding box detection results; (a) nonintersection case, (b) intersection case with rotated bounding box, and (c) unrotated bounding box.[4]

The authors in [5] employed scene semantic segmentation on images, utilizing convolutional neural networks with an encoder-decoder architecture, to generate an image where each pixel is classified into specific classes like "road," "building," and "sidewalk." This segmentation image is then used to identify road pixels and the shapes they form, which aids in distinguishing between different types of intersections. Additionally, the paper emphasizes the importance of traffic light and sign detection around intersections, focusing on the precise position and type of these landmarks rather than comprehensive attribute recognition. These components collectively contribute to the paper's intersection detection methodology as illustrated clearly in Figure 2.2.1.2.

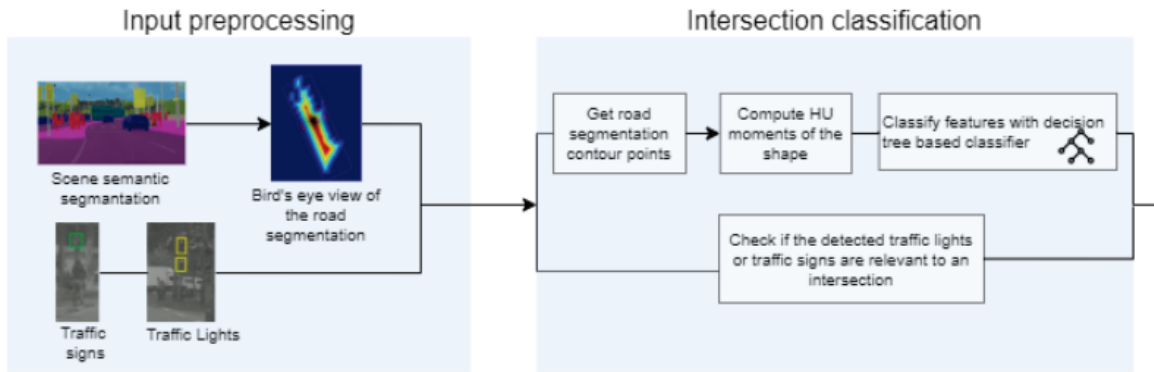


Figure 2.2.1.2 Intersection Detection Algorithm in [5]

In contrast to the aforementioned papers, our approach diverges by focusing on a localization-centric strategy within object detection. Specifically, we initiated a TensorFlow object detection model with a custom classification head and conducted training exclusively on junction images that were meticulously annotated with bounding boxes. This distinctive emphasis on localization enables us to precisely pinpoint intersection-related features, potentially enhancing the accuracy and specificity of our detection system.

It is worth mentioning that most other junction detection models capture and detect junctions from aerial-view. [3]

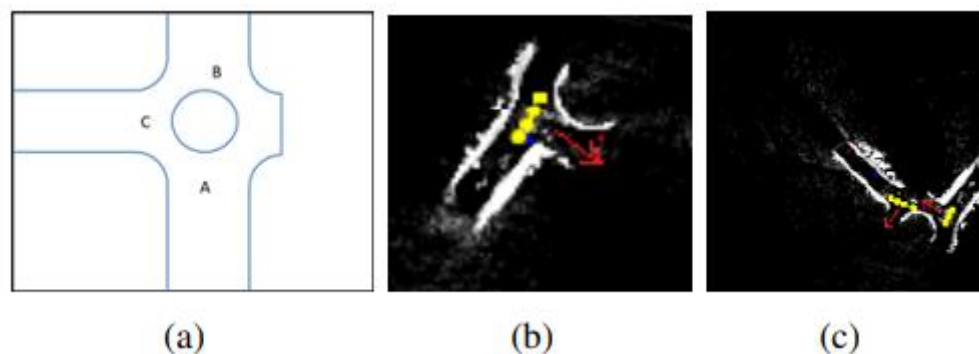


Figure 2.2.1.3 The roundabout intersection. A top-view. The red arrows in (b) and (c) point to intersections.

2.3 Previous Works on Raspberry Pi

The Raspberry Pi, a credit-card-sized single-board computer, has emerged as a powerful and cost-effective tool in the realm of object detection tasks. Its role in object detection is particularly significant due to its portability, affordability, and versatility. Raspberry Pi devices are equipped with various models, each differing in computational power and capabilities, but even the most modest Raspberry Pi models can be harnessed effectively for object detection tasks.

One of the key features that make Raspberry Pi suitable for object detection is its ability to run lightweight machine learning models efficiently. Object detection involves the identification and localization of objects within images or video frames, and this process demands computational resources. Raspberry Pi can leverage frameworks like TensorFlow Lite and OpenCV to deploy pre-trained object detection models, enabling real-time inference on images or video streams. Its compact form factor makes it suitable for embedded systems, surveillance cameras, and even robotics applications where object detection is crucial.

Moreover, the Raspberry Pi's GPIO (General Purpose Input/Output) pins allow for seamless integration with various sensors, cameras, and peripherals, making it adaptable to a wide range of object detection scenarios. Whether it's monitoring home security, wildlife tracking, or , the Raspberry Pi offers a flexible platform to implement object detection solutions. In summary, the Raspberry Pi's affordability, accessibility, and compatibility with machine learning frameworks make it a valuable asset for object detection tasks.

2.4 Previous Works using Cloud Computing on Object Detection Models

Leveraging cloud computing for object detection models has become a game-changer in the field of computer vision and artificial intelligence. Cloud computing offers several advantages for deploying, managing, and scaling object detection models, making it a preferred choice for businesses, researchers, and developers.

One of the primary benefits of using cloud computing for object detection is the immense computational power and resources available in the cloud. This concept is exemplified in a

study by [10], where cloud computing was harnessed to augment the computational and storage capabilities of local robots. Training complex deep learning models for object detection often requires significant computational resources, including high-performance GPUs and TPUs. Cloud providers like AWS, Azure, and Google Cloud offer access to these resources on-demand, allowing developers to train and fine-tune their models efficiently. This eliminates the need for expensive on-premises hardware investments and ensures that even resource-intensive tasks can be accomplished swiftly.

Scalability is another key advantage of cloud computing. Object detection models deployed in the cloud can easily scale to handle varying workloads. Whether it's processing a few images or analyzing a massive video stream, cloud-based object detection systems can dynamically allocate resources to meet demand, ensuring low latency and consistent performance. This flexibility is invaluable for applications such as surveillance, autonomous vehicles, and real-time monitoring.

Security and reliability are paramount when dealing with sensitive object detection tasks, such as in autonomous vehicles or healthcare applications. Leading cloud providers invest heavily in security measures, including data encryption, access controls, and compliance certifications. This ensures that object detection models deployed in the cloud meet rigorous security standards and can be trusted for mission-critical applications.

2.4.1 Amazon Cloud Service

In this project, Amazon Web Services (AWS) is chosen to be the cloud service provider due to its comprehensive suite of cloud computing services, extensive scalability options, and robust infrastructure, which align with the requirements and demands of the object detection tasks and computational needs of the project. Specifically, we chose to use Amazon Elastic Compute Cloud (EC2) to handle the computation. This cloud service offers machine images that encapsulate all the necessary components for a server, including the operating system. This simplifies the setup process and ensures a streamlined and efficient environment for running our object detection models. Additional settings can be customized to tailor CPU, memory, and networking capacity to meet the specific requirements of our object detection tasks. [12]

When compared to local processing, cloud-based object detection using AWS EC2 typically exhibits faster processing times. This advantage is attributed to the immense computational power and resources available in the cloud, enabling rapid execution of object detection tasks. This can be particularly beneficial for scenarios where real-time or low-latency responses are crucial, as cloud-based processing can deliver results more quickly and efficiently compared to relying solely on local resources. The speed between local and cloud is visualised in Figure 2.4.1.1.



Figure 2.4.1.1 Cloud vs Local Processing Time [12]

In summary, cloud computing has revolutionized object detection by providing access to vast computational resources, scalability, and simplified deployment and management. It has democratized the development and deployment of object detection models, making them accessible to a broader audience while ensuring reliability and security, ultimately driving innovation in various industries.

2.4.2 Weakness

Although cloud computing presents numerous advantages for various tasks, it is not without its inherent drawbacks. Chief among these concerns is network dependence, where factors such as network outages or congestion can disrupt access to the cloud-based model, potentially

rendering critical services inaccessible. Additionally, the reliance on remote cloud resources may introduce latency into the system, resulting in delays when making predictions compared to running inference locally. This is discussed in detail in [11]. This latency issue is particularly relevant in applications necessitating real-time or low-latency responses, highlighting the need for a careful consideration of the trade-offs between cloud-based computational power and the imperative of timely decision-making in certain scenarios.

Chapter 3

Proposed Method/Approach

3.1 Methodologies and General Work Procedures

The overall system consists of four parts: image acquisition, junction detection and image comparison (to be reported by my groupmate) and audible feedback.

3.1.1 Image Acquisition

The camera would need to be positioned at the front of the car, oriented toward the road. It will capture a video and a python program will handle video splitting, obtaining a frame for every second. These images will be fed into a junction detection model that uses RetinaNet and ResNet as the pre-trained backbone network.

3.1.2 Junction Detection

The object detection architecture employed is RetinaNet, a single-stage object detection network that employs a feature pyramid network to identify objects across a range of scales and aspect ratios. However, I utilized a particular pre-trained checkpoint designed for the ResNet50_v1 architecture. ResNet50_v1 is a frequently employed backbone network in object detection models. The code fetches the pre-trained checkpoint tailored for ResNet50_V1, which has been trained on the COCO dataset. Subsequently, it restores all layers except the top classification layer within the network. This enables the smooth transfer of weights to the RetinaNet architecture.

The reason for using the ResNet50_v1 checkpoint stems from its widespread adoption as a backbone network in object detection models and its proven ability to extract essential features from images effectively. The RetinaNet architecture implemented in the code necessitates a backbone network for feature extraction, and ResNet50_v1 is a popular selection owing to its robust performance track record.

Beyond the previously mentioned architecture and pre-trained checkpoint specifications, the refinement of the junction detection model included additional fine-tuning using a dedicated dataset meticulously annotated for the purpose of junction detection. This training

dataset comprised Google Street View images, painstakingly annotated using the LabelImg tool to outline bounding boxes around the junction present in the images. Leveraging this annotated dataset, the RetinaNet model undergo training to ensure that it is precisely customised for the specialised task of junction detection.

In the training phase, the RetinaNet model that was initially configured with a single class, “junction” to detect junction as per the previous approach. In an effort to enhance the model’s performance, I acquired new training data and labelled it using the LabelImg tool before employing it to train the RetinaNet model. The hyperparameter of the model during the training remained consistent with those utilised in the previous work, with the sole adjustment being the integration of a larger training dataset. After training, the model’s state was saved for future use in detecting junction in new images.

3.1.3 Audible Feedback

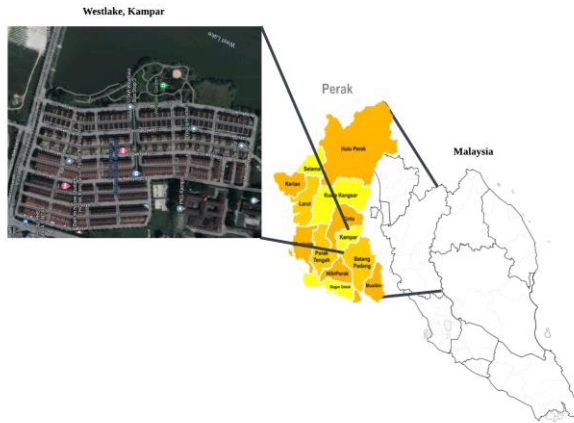
As aforementioned the system will provide audible feedback to the driver, signalling the correct turn or warning against incorrect turns. There will be 2 types of audible feedback; correct and incorrect audible sounds. When the driver is approaching the correct junction, the system will trigger the correct audible sound to inform the driver that is the correct turn to take. On the other hand, the system will trigger the correct audible sound to inform the drive that it is the correct turn to take. On the other hand, the system will trigger the incorrect audible sound as soon as it detects that the driver has taken the wrong turn. By listening to the feedback sound, drivers are able to identify whether they have taken the correct or incorrect turns. The decision of what types of sound to make in specific situation is determined by a Python program code, which also relies on the output from the model.

3.1.4 Cloud Computing

Due to the limitation in computing power in Raspberry Pi, I will be a virtual server, Amazon EC2 to host and run the junction detection model and other codes such as image similarity detection and so on. The input for the junction detection model, which are the frames mentioned in Section 3.1.1 will be uploaded to Amazon S3 to provide easy access to the EC2 instance.

3.2 Study Area and data

Our research was conducted in Westlake, a neighbourhood just beside UTAR, Kampar. The figure below shows the exact location of our test case.



The training data can be accessed here:

<https://drive.google.com/drive/folders/1HdFo0P83sj5dUjiYYEqPsgSdtE37ZkF8?usp=sharing>

The testing data can be accessed here:

https://drive.google.com/drive/folders/1E8YGYOqdUYT1wHcu__ry5poARVBUQWNs?usp=sharing

The table below shows the summarisation of the dataset:

Table 3.2.1 Summarisation of Dataset

Dataset	Training Data	Testing Data
Size	1085 images	41 images
Collection Method	Snipping Tool (Google Street View)	Captured using camera
Image Preprocessing	Cropped unnecessary parts, resized to 1024 x 1024	-
Annotation Tool	LabelImg (bounding box labeling)	Manually labelled
Annotation Format	XML file (exported from LabelImg)	.txt file
Annotation Purpose	Fine tune the pre-trained model	Calculate accuracy

Chapter 4

System Design

4.1 System Block Diagram

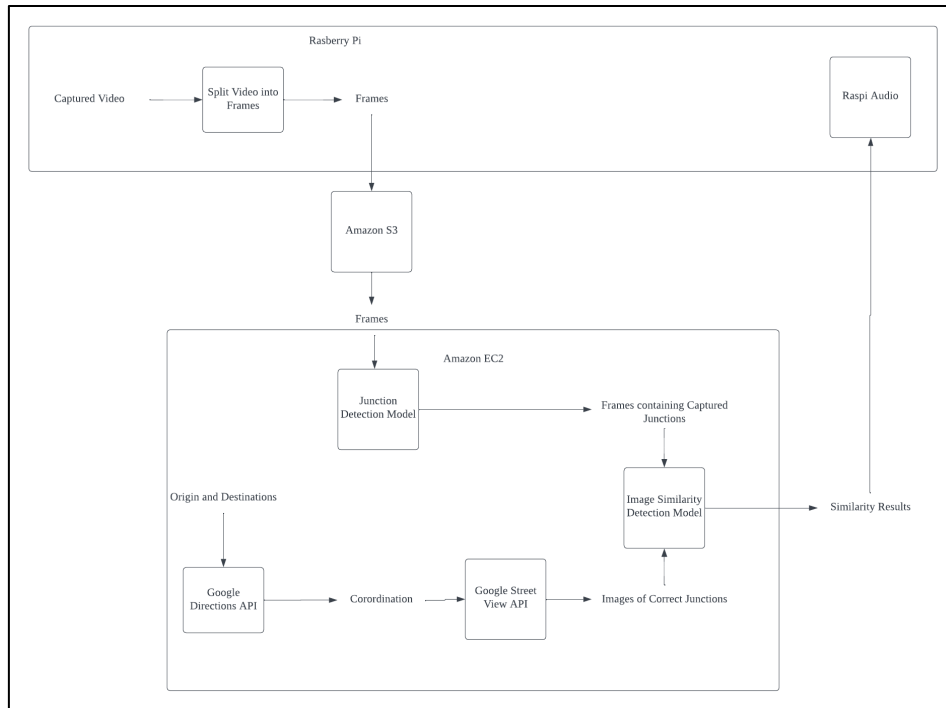


Figure 4.1 System Block Diagram

4.2 System Components Specifications

4.2.1 Camera + Raspberry Pi

The process commences with the front-facing camera capturing the street scene in video format. A Python program running on the Raspberry Pi then splits this video into individual frames at a rate of one frame per second. These freshly captured frames are the input for the subsequent stage of processing.

4.2.2 Amazon S3 (Simple Storage Service) (made some changes – newly added)

The Amazon S3 (Simple Storage Service) is a cloud storage service offered by one of the pioneers of cloud services providers in the market, Amazon Web Services (AWS). AWS is known for its scalability and security. It enables users to conveniently store and retrieve various types of data, including files and objects via the internet. The purpose of S3 in this project is that it will be used to store the frames that are uploaded from the Raspberry Pi which was initially capture by the camera. The reason of uploading the frames into Amazon S3 will be explained in Section 4.2.3.

4.2.3 Amazon EC2

The Amazon EC2 (Elastic Computer Cloud) is a web service provided by Amazon Web Services (AWS) that offers flexible and scalable computing resources to handle any computation that is needed in the cloud. Users can easily start and oversee virtual servers, referred to as ‘instances’, on-demand.

The Raspberry Pi 4 Model B, with its 1.8 GHz quad-core Cortex-A72 processor and 2GB SDRAM, faces processing power limitations when tasked with running both the Junction Detection Model and Image Similarity Detection Model. These models demand substantial computing resources, which can strain the Raspberry Pi's capabilities. To address this challenge, we've opted for an alternative approach—leveraging Amazon EC2 for cloud computing. By migrating both models to Amazon EC2, we can harness the cloud's computational prowess, relieving the Raspberry Pi of this heavy load. Additionally, to further optimize system performance, we've implemented a process where the camera-captured frames are uploaded to Amazon S3. This strategy significantly reduces the processing burden on the Raspberry Pi when interfacing with EC2. Configuring Amazon EC2 instances to retrieve input images directly from Amazon S3 proves to be a more efficient approach compared to transmitting images as payloads directly to EC2. Following cloud-based computations, the results are then relayed back to the Raspberry Pi for further processing.

4.2.3.1 Junction Detection Model

In this report, the junction detection model undergoes fine-tuning with an expanded dataset comprising a larger number of annotated junction images obtained from Google Street View. The hyperparameters of the model remain consistent with those utilized in the prior work. Following fine-tuning, we will save the model's state for future use in detecting junctions within new images. Then the trained model will be uploaded to Amazon EC2. The model in EC2 will get the frames from Amazon S3 and then perform junction detection on said frames. If there is existence of junction in the frames, the model will then send that frame to the following block which is Image Similarity Detection Model for further processing.

4.2.3.2 Image Similarity Detection Model

This part is done by my groupmate, Tan Yi Xuan.

4.2.3.3 Google Directions API

This part is done by my groupmate, Tan Yi Xuan.

4.2.3.4 Google Street View API

This part is done by my groupmate, Tan Yi Xuan.

4.2.4 Raspi Audio

As mentioned in the earlier part of the paper, the system will provide 2 types of audible feedback to the driver. The python code of Raspi Audio is employed in the Raspberry Pi device which and the outcome of the python code will be determined by the similarity result obtained from the preceding block. In essence, if the detected junction is the correct junction, the system will initiate the corresponding correct auditory signal; conversely when user has taken the wrong junction, the system will prompt the system to emit the corresponding feedback.

Chapter 5

System Implementation

5.1 Hardware Setup

The hardware that will be used in this project includes a laptop, a Raspberry Pi device, and a camera. The laptop will serve the purpose of configuring the Raspberry Pi as well as training the junction detection model. Then the Raspberry Pi will be used as an edge device to perform the computer vision tasks. The variant of the Raspberry Pi is Raspberry Pi 4 Model B. The camera will capture the input, which consist of street images. An audio system will be employed to provide output, informing the user whether the turn is correct or if they have taken the wrong turn.

Table 5.1 Specifications of laptop

Description	Specifications
Model	ILLEGEAR ROGUE
OS	Window 10 Pro (Version 21H2)
CPU	Ryzen 4800H
RAM	16GB
Graphic	Nvidia GTX1650 TI
Storage	512GB M.2 PCIe NVMe SSD

Table 5.2 Specifications of Raspberry Pi

Description	Specifications
Model	Raspberry Pi 4 Model B
OS	Debian GNU/Linux 11 (bulleye)
CPU	Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
RAM	2GB LPDDR4-3200 SDRAM
Hard Disk	SanDisk EDGE 16GB

Table 5.3 Specifications of Camera

Description	Specifications
Model	Rapoo XW180
Frame Rate	Up to VGA 30FPS
Video Resolution	FHD 1080P/HD 720P
Connector Type	USB 2.0

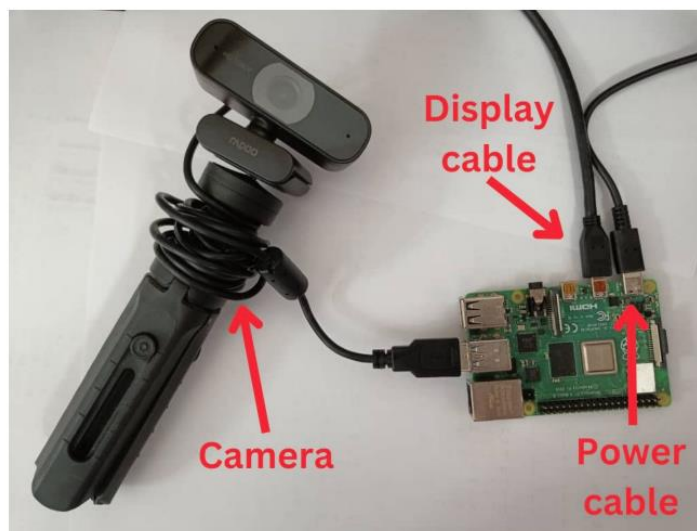


Figure 5.1 The setup of the Raspberry Pi 4 Model B

5.2 Software Setup

The software that will be used in this project includes Jupyter notebook, TensorFlow, Raspberry Pi OS, Amazon EC2, Amazon S3.

5.2.1 Jupyter notebook

Table 5.4 Specifications of Jupyter Notebook

Description	Specifications
Version	7.0.3

5.2.2 TensorFlow

Table 5.5 Specifications of TensorFlow

Description	Specifications
Version	2.13.0

5.2.3 Operating system of Raspberry Pi

Table 5.6 Specification of Operating System of Raspberry Pi

Description	Specifications
Version	Debian 11 (Bullseye) 64-bit

5.2.4 Amazon EC2

Table 5.7 Specification of Amazon EC2

Description	Specifications
CPU	2 virtual CPU
RAM	8GB
OS	Amazon Linux 2023

5.3 Settings and Configuration

5.3.1 Jupyter notebook

The settings and configuration that was done on Jupyter notebook is not much because it is only the platform to train the Junction Detection Model.

5.3.2 TensorFlow

The pre-trained model that was used to train the Junction Detection Model is TensorFlow. Initially TensorFlow's Object Detection has 2 heads which are object detection head and box regression head. Each of these head is responsible for a specific task or output. For instance, classification head will determine what objects are present in an image and assigning class labels to those objects. On the other hand, box regression head is responsible for refining the bounding box coordinates of the detected objects. Our approach in this project is that we will preserve the weight of the box regression head and then make changes to the classification head. In a simpler terms, we train the classification head with manually labelled training data to be able to classify between junction and non-junction images while restoring the weight of the box regression head.

5.3.3 Raspberry Pi OS

The Raspberry Pi device was configured with a 32-bit Debian 10 initially. After that we upgraded it to 64bit of Debian 11 as the device is equipped with a 64-bit CPU, which can significantly enhance the system’s overall performance. We observed that the execution of python to be slightly faster than the previous version.

5.3.4 Amazon S3

The configuration of Amazon S3 is done through the Amazon S3 console. After that we also created a bucket in the Amazon S3 to store the frames that will be uploaded from Raspberry Pi. After that we need to make sure that Amazon S3 is configured to be able to connect to Amazon EC2 as the frames that stored in the S3 will be used by the EC2. The connection between them requires a few steps. First of all, we need to run “AWS configure” in EC2 and insert access key and secret access key to verify the credentials. With the help of “boto3” library, EC2 can then establish client connection to S3.

5.3.5 Amazon EC2 (Elastic Computing Cloud)

The configuration of Amazon EC2 is done by launching the instance also known as virtual server via AWS (Amazon Web Service) console.

The initial consideration when setting up an instance in EC2 is that we need to select the appropriate AMI (Amazon Machine Image) according to our usage. Think of AMI as similar to an ISO image, as it will contains components like operating system, applications and other additional libraries that will be installed on the instance. For our case we are going to choose Linux x86_64 HVM kernel-6.1 as the AMI of our virtual server.

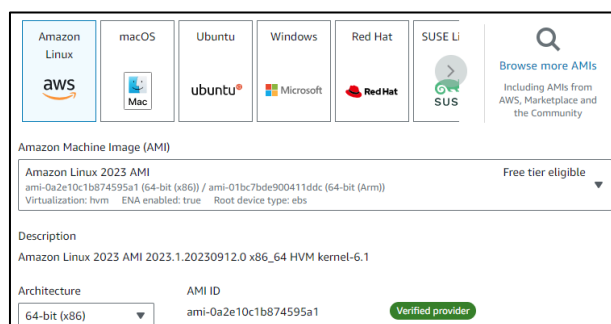


Figure 5.2 The configuration of AMI in EC2

The next configuration is the instance type of the EC2. Instance Types refer to virtual servers capable of running applications. AWS offer different combination of CPU, memory, storage, and networking capabilities allow the users to tailor their resource allocation accordingly. In our case, we selected instance type of t2.large as it is aligned to our model's need and is more cost efficient.

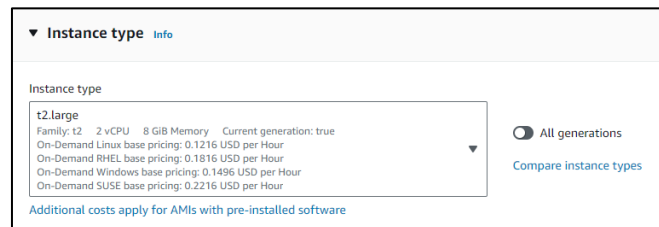


Figure 5.2 The configuration of instance type in EC2

By default, an Elastic Block store (EBS) is configured with volume size of 8GB of SSD. In our case, we configure the EBS to 25GB.

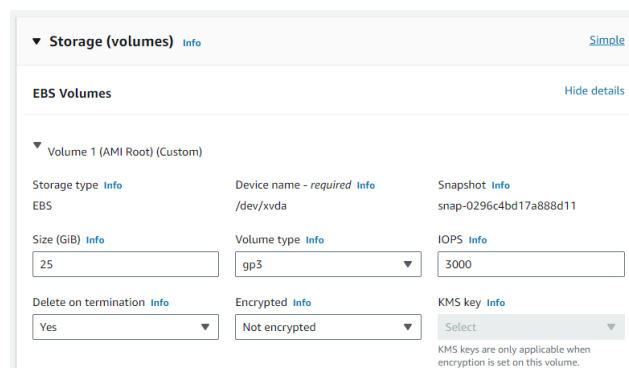


Figure 5.3 The configuration of EBS in EC2

Other than initial configuration of the EC2, the installation of libraries into the server is also required. As aforementioned, both of the Junction Detection Model and Image Similarity Detection Model are uploaded to the virtual server EC2 which means that EC2 also requires some libraries to be installed in order to perform detection as well as similarity calculation.

5.4 System Operation (screenshot)

5.4.1 Video Acquisition

The driving scene video is obtained by connecting a web camera to Raspberry Pi and capturing the camera input.

5.4.2 Video Frame Extraction

On Raspberry Pi, I performed video frame extraction using a frame rate of 1fps to get only 1 frame per second. Figure 5.4.2.1 shows the input video and Figure 5.4.2.2 shows the output frames of the video.

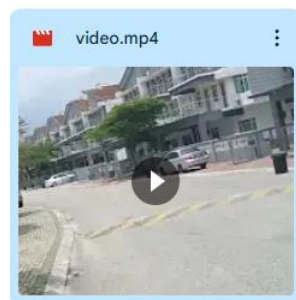


Figure 5.4.2.1 Input Video

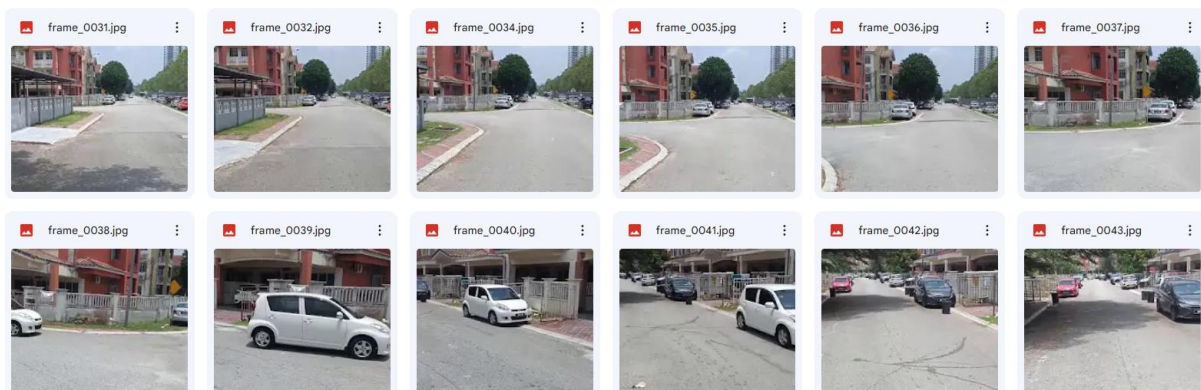


Figure 5.4.2.2 Output Frames

5.4.3 Sending Frames to Amazon S3

After authenticating credentials at Jupyter Notebook level, the Raspberry Pi is able to send the frames to an Amazon S3 bucket that we previously configured with the following code (Figure 5.4.3.1)

```
# Loop through the list of files and upload each image to S3
for local_file in local_files:
    if local_file.lower().endswith(('.jpg', '.jpeg', '.png', '.gif', '.bmp')):
        # Specify the destination object key (the path and filename in the bucket)
        object_key = f'frames/{local_file}' # Include the filename in the object_key

        # Specify the local path to the image file
        local_image_path = os.path.join(local_directory, local_file)

        # Upload the image to S3
        s3.upload_file(local_image_path, bucket_name, object_key)

    #print(f"Uploaded: {local_image_path} to S3 bucket: {bucket_name} with object key: {object_key}")
```

Figure 5.4.3.1 Code Snippet to send images to Amazon S3

The figure below shows the Amazon S3 console and the frames that have been uploaded.

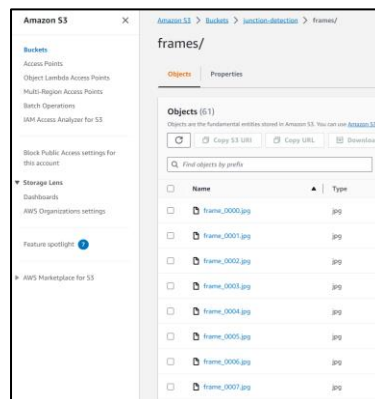


Figure 5.4.3.2 Amazon S3 Console

5.4.4 Connect to Amazon EC2 to run Python Script

After that, the Raspberry Pi need to use Secure Socket Shell (SSH) to connect to the EC2 instance to be able to run the python script that has been previously uploaded to the virtual server. The figure below shows the EC2 instance on AWS console.

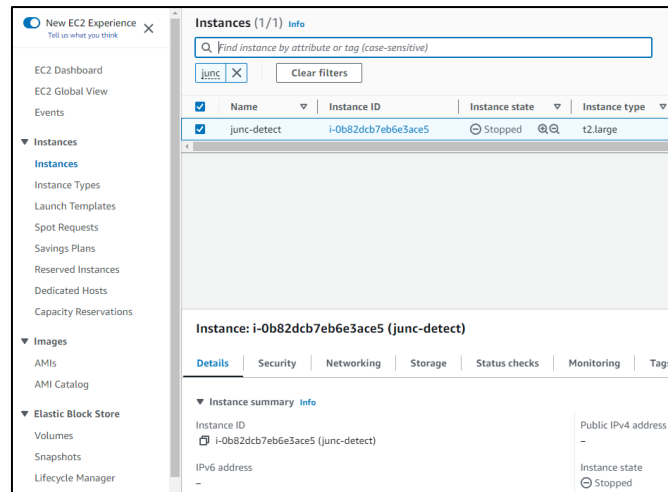


Figure 5.4.4.1 Amazon EC2 Console

5.4.5 Handle Output to get Comparison Result

I configured the SSH connection and command so that the EC2 will print the output to the Jupyter Notebook in real time. In this way, I can capture the result of each of the frames instead of getting the results only when the python script has finished running. Figure 5.4.5.1 shows the output of the EC2 python script in Jupyter Notebook.

```

Frame 1 contains junctions
1/1 [=====] - 1s 864ms/step
Comparison Result: 0.7220862507820129
Audio: 1

```

Figure 5.4.5.1 Output from the EC2 python script

Upon getting this data, I extracted “Audio: 1” or “Audio: 0” from the output.

5.4.6 Trigger Raspi Audio

The figure below shows the code snippet to extract the keywords and play raspi audios accordingly.

```

# Print the output in real-time
for line in iter(stdout.readline, ""):
    print(line, end='')
    if "Audio: 1" in line:
        print("Playing audio for positive result")
        play_beep_sound(pos)

    elif "Audio: 0" in line:
        print("Playing audio for negative result")
        play_beep_sound(neg)

```

Figure 5.4.6.1 Code snippet for Handling Output

5.5 Implementation Issues & Challenges

5.5.1 Migration of code to Raspberry Pi

Migrating code to Raspberry Pi from the laptop that was used to train the model can often encounter compatibility issues due to differences in operating system, and software dependencies. To solve this kind of problem, I must go through some research to download certain software dependencies and also finding the way of achieving the same output but on different operating systems. For example, making audible “beep” can be very easy on device that has Window Operating System as they just need to import “winsound” library to the python code and by using “winsound.Beep()” the system will make sound. On the other hand, we will need to install “beepy” libraries for Debian 11. Then the sound can be trigger by calling “beepy.beep()” in the code.

5.5.2 Computation Limitation of Raspberry Pi

The next challenge is the performance of both Junction Detection Model and Image Similarity Detection Model noticeably decreased after migrating the code from the laptop that trained the models to the Raspberry Pi device, primarily due to the computational limitations inherent to possesses significantly lower processing power and memory capacity compared to most laptops. As a result, computationally intensive task that run smoothly on a laptop will experience delays or execution time to complete the execution of the code is longer than expected on the Raspberry Pi device.

With that said, we looked for alternative ways and found that cloud service like Amazon Web Service (AWS) which is one of the pioneers of web service that one can find in the market nowadays can help us to overcome the issue of computational limitation in the Raspberry Pi. In simpler terms, both computationally intensive models will be uploaded to a virtual server in the cloud and most of the computation will be done in the cloud. Meanwhile Raspberry Pi will just handle in the input images which are the frames taken by the camera and upload those frames to the cloud to be executed by the model that is currently hosted in the virtual server in AWS. Furthermore, Raspberry Pi will also need to handle the output from the cloud which is the similarity result then trigger the corresponding audible feedback.

5.5.3 Internet Connectivity Issues

As mentioned earlier, both the Junction Detection Model and Image Similarity Detection Model are hosted in the cloud. This implies that the system requires a consistent and reliable Internet connection to execute the models. The Raspberry Pi device will rely on the internet to upload the frames captured by the camera so that these frames can be processed by the models, other than that the Raspberry Pi also need to handle the result that was returned by the model from the cloud. In the previous work, only a very small part of the system requires internet access which is the part where the system need to use Google Direction API to get the coordination as well as Google Street View API to get the image of correct junction. However, now even the Junction Detection Model and Image Similarity Detection Model required internet connectivity to perform detection.

5.6 Concluding Remarks

In conclusion, this project has showcased the potential of harnessing a combination of advanced technologies, from deep learning models to cloud-based solutions, to address real-world challenges in road safety and driver support. While the journey has been marked by its share of challenges, including compatibility issues during code migration and the inherent limitations of Raspberry Pi's computational power, we've successfully navigated these obstacles by leveraging the capabilities of Amazon Web Services (AWS) to offload intensive computations to the cloud. This shift to a cloud-based approach not only enhances the system's performance but also underscores the critical importance of a reliable internet connection in our setup.

Chapter 6

System Evaluation and Discussion

6.1 System Testing and Performance Metrics

Traditional evaluation metrics such as Intersection over Union (IOU) for object detection rely on precise object boundaries, which are well-defined for tangible objects like vehicles or pedestrians. However, the challenge with junction detection lies in the intangible nature of junctions; they don't have fixed boundaries, making it impractical to use bounding boxes and IOU as evaluation criteria.

As a solution, we propose using accuracy as an alternative metric for assessing the model's performance. In this context, accuracy will measure the model's ability to correctly classify images as either 'junction present' or 'no junction.' To establish ground truth, manual labelling of test images as '1' (indicating the presence of a junction) or '0' (indicating no junction) will be performed. The model's predictions will then be compared to this ground truth, allowing us to determine its accuracy in detecting junctions. This approach accounts for the unique characteristics of junctions and provides a meaningful measure of the model's real-world performance.

I will be using the metrics below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (2)$$

6.2 Testing Setup and Result

I used 41 test images to test the model's performance. They are frames that are split from a captured video in Westlake, Kampar. I manually labelled the test images with 1 or 0 depending on whether the image contain junction or not. The ground truth value is saved in an .txt file. The link to access the test data is provided in Section 3.2.

The result of the model is as follows:

Table 6.2.1 Results of Model

Metrics	Score
Accuracy	0.71
F1 Score	0.67

6.3 Project Challenges

One of the significant challenges faced during this project was the intangible nature of junctions as objects. Determining the precise boundaries for labeling training data, such as where the bounding box should start and end, proved to be a complex task. This inconsistency in labeling may have contributed to suboptimal model performance after training. Additionally, environmental factors like obstructions and varying weather conditions posed challenges in accurately detecting junctions.

Another noteworthy challenge was related to latency when utilizing cloud services. While the performance of using Amazon EC2 for computation improved significantly compared to the Raspberry Pi, there remained a noticeable delay in processing due to the utilization of cloud services. It is possible that the EC2 instance may not have been fully optimized to handle the extensive processing demands, leading to latency issues.

6.4 Objectives Evaluation

The primary goal of this project is to accurately detect junctions in frames captured by the camera. Junction detection accuracy is measured using the accuracy metric. This metric assesses the model's ability to correctly identify whether a junction is present in a given frame. The achieved accuracy score is 0.71, indicating a substantial success in meeting this primary objective.

Another critical objective is to assess the efficiency and latency of cloud-based processing since I are using cloud computing. This evaluation considers the time required for image processing and analysis on cloud infrastructure, specifically Amazon EC2. It helps determine whether the cloud-based approach aligns with project requirements and expectations. It is

essential to highlight that during the project's implementation and testing phases, a significant latency was observed in the processing pipeline. This latency, while expected in cloud-based processing, was particularly noticeable and merits consideration for future improvements.

6.5 Concluding Remarks

In conclusion, this project has achieved notable success in addressing the challenging task of junction detection. Traditional metrics like Intersection over Union (IOU), which are suited for well-defined tangible objects, were impractical for junctions due to their intangible nature. As a solution, we introduced accuracy as an alternative evaluation metric, which proved effective in assessing the model's real-world performance. With an accuracy score of 0.71, the model demonstrated its capability to distinguish between images with and without junctions, aligning with our primary objective.

However, it's crucial to acknowledge the project's significant challenges. The intangibility of junctions complicated the labeling process during training, potentially impacting the model's performance. Environmental factors, including obstructions and varying weather conditions, posed further challenges in achieving accurate junction detection.

Moreover, the integration of cloud-based processing, while enhancing computational capabilities with Amazon EC2, introduced noticeable latency into the system. This latency, although expected in cloud computing, calls for optimization and future improvements to streamline processing times.

In summary, this project has made substantial progress in achieving its primary objectives, notably in junction detection accuracy. Yet, the challenges encountered, such as the unique nature of junctions and latency issues, highlight areas for future refinement and optimization. These findings provide valuable insights for further enhancing the efficiency and effectiveness of our system.

Chapter 7

Conclusion and Recommendation

7.1 Conclusion

The decision to harness the power of cloud computing services, as opposed to local execution on the Raspberry Pi, offers a range of significant advantages. First and foremost, utilizing AWS EC2 enables us to seamlessly scale our computational resources in response to fluctuating workloads, ensuring optimal processing efficiency at all times. This scalability not only enhances performance but also promotes cost efficiency, as resource-intensive tasks are offloaded to EC2, reducing the computational load and translating into cost savings and improved energy efficiency for the Raspberry Pi. Moreover, AWS's robust and highly available infrastructure guarantees uninterrupted accessibility and the continuous operation of our models, boosting overall reliability. Additionally, the remote manageability afforded by EC2 empowers us to efficiently oversee and monitor our instances, simplifying maintenance, updates, and troubleshooting processes. Lastly, AWS's centralized data storage solutions, such as S3, enhance our data management practices, further streamlining our project's objectives. To summarise, the primary object of this project is to bolster driver confidence by delivering real-time audible feedback throughout their journeys and these cloud-based advantages serve as essential enablers in achieving that objective.

7.2 Recommendation

As we reflect on the overall satisfactory results achieved in the current project, it is worth considering avenues for further enhancement and exploration. One promising direction is to delve into additional AWS services, particularly Amazon SageMaker, for the implementation of machine learning solutions. SageMaker offers a comprehensive suite of tools and capabilities tailored for machine learning model development, training, and deployment. Exploring SageMaker's potential can open up new horizons in our project, enabling us to leverage its streamlined workflows, built-in algorithms, and scalable infrastructure. By incorporating SageMaker into our project's ecosystem, we can aim for even greater efficiency, scalability, and invocation in our machine learning endeavours.

The next future work that can be done is expanding the scope of our study area and increasing the number of training data. Due to the constraints of time and manpower in this

project, the model was trained primarily using data from the immediate vicinity of UTAR. However, we strongly believe that the potential of this system extends far beyond its current boundaries. With the implementation of an expanded training dataset, we firmly believe that the system can demonstrate remarkable performance improvements. By including data from a wider geographic area, we can enhance the model's ability to recognise various road conditions and driving scenarios. This expanded capability would position the system as a valuable support tool for drivers in diverse real-world situations.

REFERENCES

- [1] X. Li et al., "Assessing street-level urban greenery using google street view and a modified Green View index," *Urban Forestry & Urban Greening*, <https://www.sciencedirect.com/science/article/pii/S1618866715000874> (accessed Sep. 12, 2023).
- [2] A. Campbell, A. Both, and Q. (Chayn) Sun, "Detecting and mapping traffic signs from Google Street View images using Deep Learning and GIS," *Computers, Environment and Urban Systems*, <https://www.sciencedirect.com/science/article/pii/S0198971519300870> (accessed Sep. 13, 2023).
- [3] P. Mukhija, S. Tourani, and K. M. Krishna, "Outdoor Intersection Detection for Autonomous Exploration," *IEEE Xplore*, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7169508> (accessed Sep. 13, 2023).
- [4] J. Ahn, Y. Lee, M. Kim, and J. Park, "Vision-based branch road detection for intersection navigation in unstructured environment using Multi-Task Network," *Journal of Advanced Transportation*, <https://www.hindawi.com/journals/jat/2022/9328398/> (accessed Sep. 14, 2023).
- [5] A. G. Bălănescu, C. L. Sîrbu, and C. Orhei, "Intersection detection based on Mono-camera sensor | IEEE conference ...," *IEEE Xplore*, <https://ieeexplore.ieee.org/document/9851381/> (accessed Sep. 13, 2023).
- [6] A. Campbell, A. Both, and Q. (Chayn) Sun, "Detecting and mapping traffic signs from Google Street View images using Deep Learning and GIS," *Computers, Environment and Urban Systems*, <https://www.sciencedirect.com/science/article/pii/S0198971519300870> (accessed Sep. 14, 2023).
- [7] H. Eklund, "Diva," OBJECT DETECTION: MODEL COMPARISON ON AUTOMATED DOCUMENT CONTENT INTERPRETATION, <https://www.diva-portal.org/smash/get/diva2:994607/FULLTEXT01.pdf> (accessed Sep. 13, 2023).
- [8] T. V. Janahiraman and M. S. Mohamed Subuhan, "IEEE Xplore Full-text PDF:," *IEEE Xplore*, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7169508> (accessed Sep. 13, 2023).
- [9] R. B. Islam, S. Akhter, F. Iqbal, Md. S. U. Rahman, and R. Khan, "Deep learning based object detection and surrounding environment description for visually impaired people," *Heliyon*, <https://www.sciencedirect.com/science/article/pii/S2405844023041312> (accessed Sep. 14, 2023).

- [10] Y. Jian, W. Xin, Z. Xue, and Dai ZhenYou, "IEEE Xplore," Cloud computing and visual attention based object detection for power substation surveillance robots, <https://ieeexplore.ieee.org/abstract/document/7129299> (accessed Sep. 14, 2023).
- [11] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang , "IEEE Xplore," Distributed and Efficient Object Detection in Edge Computing: Challenges and Solutions, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8337810> (accessed Sep. 14, 2023).
- [12] M. Guillermo et al., "IEEE Xplore," Implementation of Automated Annotation through Mask RCNN Object Detection model in CVAT using AWS EC2 Instance, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9293906> (accessed Sep. 14, 2023).

Appendix

A.1 Raspi_code.ipynb

Split Video into Frames

```
import cv2 as cv
import os

# Open the video file
video_path = 'wrong1.mp4' # Replace with the path to your video file
cap = cv.VideoCapture(video_path)

# Check if the video file was opened successfully
if not cap.isOpened():
    print("Error: Could not open video file.")
    exit()

# Get the frames per second (FPS) of the video
fps = int(cap.get(cv.CAP_PROP_FPS))

# Define the output directory where frames will be saved
output_directory = 'output' # Change this to your desired output directory
os.makedirs(output_directory, exist_ok=True)

frame_count = 0

# Loop through the frames
while True:
    # Read the next frame
    ret, frame = cap.read()

    # Break the loop if we have reached the end of the video
    if not ret:
        break

    # Save the frame to the output directory
    frame_filename = os.path.join(output_directory, f'frame_{frame_count:04d}.jpg')
    cv.imwrite(frame_filename, frame)

    frame_count += 1

    # Skip frames to capture one frame per second
    frames_to_skip = fps - 1
    for _ in range(frames_to_skip):
        cap.read()

# Release the video capture object and close any open windows
cap.release()
cv.destroyAllWindows()

print(f"Saved {frame_count} frames to {output_directory}.")
```

Bachelor of Computer Science (Honours)
Faculty of Information and Communication Technology (Kampar Campus), UTAR

```

print(f"Video FPS: {fps}")

# Upload Frames to Amazon S3 Bucket
import os

#credentials
os.environ['AWS_ACCESS_KEY_ID'] = 'AKIAQVCXYEDYCERTQR4Y'
os.environ['AWS_SECRET_ACCESS_KEY'] = '*'
os.environ['AWS_DEFAULT_REGION'] = 'ap-northeast-1'

import os
import boto3

# Initialize an S3 client
s3 = boto3.client('s3')

# Specify your S3 bucket name
bucket_name = 'junction-detection'

# Specify the local directory path containing the images
local_directory = 'output'

# List all files in the local directory
local_files = os.listdir(local_directory)

# Loop through the list of files and upload each image to S3
for local_file in local_files:
    if local_file.lower().endswith(('.jpg', '.jpeg', '.png', '.gif', '.bmp')):
        # Specify the destination object key (the path and filename in the bucket)
        object_key = f'frames/{local_file}' # Include the filename in the object_key

        # Specify the local path to the image file
        local_image_path = os.path.join(local_directory, local_file)

        # Upload the image to S3
        s3.upload_file(local_image_path, bucket_name, object_key)

        #print(f"Uploaded: {local_image_path} to S3 bucket: {bucket_name} with object key:
        {object_key}")

# Audio function
import beepy
def play_beep_sound(value):
    pygame.mixer.init()
    if value > 0:
        beepy.beep("ping") #positive
    else:
        beepy.beep("error") #negative

```

```
# SSH to EC2 instance to Run Python Script containing junction detection model and
image similarity model
```

```
import subprocess
import paramiko
```

```
# Establish SSH connection
cert = paramiko.RSAKey.from_private_key_file("junc.pem")
c = paramiko.SSHClient()
c.set_missing_host_key_policy(paramiko.AutoAddPolicy())
print("connecting...")
c.connect(hostname="13.115.166.35", username="ec2-user", pkey=cert)
print("connected!!!")
```

```
# Command to execute your script
command = ['python3', '/home/ec2-user/Pipeline.py']
```

```
# Run the script and capture its output
process = c.exec_command(' '.join(command), get_pty=True)
stdout = process[1]
```

```
# Print the output in real-time
for line in iter(stdout.readline, ""):
    print(line, end="")
    if "Audio: 1" in line:
        print("Playing audio for positive result")
        play_beep_sound(1)

    elif "Audio: 0" in line:
        print("Playing audio for negative result")
        play_beep_sound(0)
```

```
# Wait for the process to finish
process_exit_code = process[0].wait()
print(f"Script exit code: {process_exit_code}")
```

```
# Close the SSH connection
c.close()
```

A.2 ec2_code.py

```
import tensorflow as tf
import os
import numpy as np
from six import BytesIO
import numpy as np
import keras
import cv2
from PIL import Image, ImageDraw, ImageFont

from keras.layers import Flatten, Dense, Input, concatenate
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout
from keras.models import Model
from keras.models import Sequential
from scipy import spatial
#from tensorflow.keras.applications.vgg16 import VGG16
import boto3

def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.

    Puts image into numpy array to feed into tensorflow graph.
    Note that by convention we put it into a numpy array with shape
    (height, width, channels), where channels=3 for RGB.

    Args:
        path: a file path.

    Returns:
        uint8 numpy array with shape (img_height, img_width, 3)
    """
    img_data = tf.io.gfile.GFile(path, 'rb').read()
    image = Image.open(BytesIO(img_data))
    image = image.resize((1024, 1024))
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)

#for multiple boxes
def plot_detections(image_np,
                    boxes,
                    classes,
                    scores,
                    category_index,
                    figsize=(12, 16),
                    image_name=None,
                    title=None):
    """Wrapper function to visualize detections.

    Args:
        image_np: uint8 numpy array with shape (img_height, img_width, 3)
        boxes: a numpy array of shape [N, 4]
        classes: a numpy array of shape [N]. Note that class indices are 1-based,
    """
```

```

        and match the keys in the label map.
    scores: a numpy array of shape [N] or None. If scores=None, then
        this function assumes that the boxes to be plotted are groundtruth
        boxes and plot all boxes as black with no classes or scores.
    category_index: a dict containing category dictionaries (each holding
        category index `id` and category name `name`) keyed by category indices.
    figsize: size for the figure.
    image_name: a name for the image file.
    """
    image_np_with_annotations = image_np.copy()
    if scores is None:
        scores = np.ones_like(classes, dtype=np.float32)
    viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_annotations,
        boxes,
        classes,
        scores,
        category_index,
        use_normalized_coordinates=True,
        min_score_thresh=0.45,
        line_thickness=2)
    if image_name:
        plt.imsave(image_name, image_np_with_annotations)
    else:
        plt.figure(figsize=figsize)
        plt.imshow(image_np_with_annotations)
        if title:
            plt.title(title)
        plt.show()

## Load the Saved Junction Detection Model

import os
saved_model_path = r'/home/ec2-user/FYP2_retrain4'
pb_file_path = os.path.join(saved_model_path, 'saved_model.pb')
print(os.path.exists(pb_file_path))

loaded_module = tf.saved_model.load('/home/ec2-user/FYP2_retrain4')
loaded_detection_fn = loaded_module.detect
loaded_fine_tuned_model = loaded_module.detection_model

### Redefine Variables

junction_class_id = 1
num_classes = 1
category_index = {junction_class_id: {'id': junction_class_id, 'name': 'junction'}}

# Load Similarity Model
## VGG-16 model with pre-trained weights

```

```

vgg16 = keras.applications.VGG16(weights="imagenet", include_top=True, pooling="max",
input_shape=(224, 224, 3))

# Extract fc2 layer

basemodel = Model(inputs=vgg16.input, outputs=vgg16.get_layer("fc2").output)

# Obtain feature vectors function
def get_feature_vector(img):
    img = cv2.resize(img, (224, 224))
    feature_vector = basemodel.predict(img.reshape(1, 224, 224, 3))
    return feature_vector

# Calculate similarity using cosine similarity
def calculate_similarity(vector1, vector2):
    return 1 - spatial.distance.cosine(vector1, vector2)

# Get Coordinates of all Junctions in Desired Route using Google Directions API
import requests
import json

# Define the API endpoint
endpoint = 'https://maps.googleapis.com/maps/api/directions/json'

# Define a list of origins and destinations
origins = ['1066, Jln Seksyen 1/2, Taman Bandar Barat, 31900 Kampar, Perak']
destinations = ['1337, Jln Seksyen 1/3, Taman Bandar Barat, 31900 Kampar, Perak']

params = {
    'key': 'AIzaSyCRs4VDM3CDp4FBoZ8fZVJ6cg9XRd2Wxqc',
    'mode': 'driving'
}

# Iterate through origins and destinations
for origin in origins:
    for destination in destinations:
        # Set the origin and destination for this iteration
        params['origin'] = origin
        params['destination'] = destination

        # Make the API request
        response = requests.get(endpoint, params=params)
        # Check if the request was successful
        if response.status_code == 200:
            # Parse the JSON response
            routes = response.json()

        else:
            print(f"Error: {response.status_code}")

```

```

## Get the cardinal direction of the junction coordinates
def get_new_heading(current_heading, turn_direction):
    """
    Calculate the new heading based on the current heading and turn direction.

    Args:
        current_heading: Current heading (e.g., "north", "east", "south", "west").
        turn_direction: Turn direction (e.g., "left" or "right").

    Returns:
        New heading as a string.
    """
    cardinal_directions = ["north", "east", "south", "west"]

    # Define the order of cardinal directions
    if current_heading not in cardinal_directions:
        raise ValueError("Invalid current heading")

    index = cardinal_directions.index(current_heading)

    # Calculate the new index based on the turn direction
    if turn_direction == "left":
        new_index = (index - 1) % 4
    elif turn_direction == "right":
        new_index = (index + 1) % 4
    else:
        raise ValueError("Invalid turn direction")

    return cardinal_directions[new_index]

# List to store start locations with associated cardinal directions
junctions = []

# Initialize variables
current_heading = None
previous_turn_direction = None

for route in routes['routes']:
    for leg in route['legs']:
        for step in leg['steps']:
            instruction = step['html_instructions']

            # Check for headings
            if "Head <b>" in instruction:
                current_heading = instruction.split("<b>")[1].split("</b>")[0].lower()

            # Check for turns
            elif "Turn <b>" in instruction:
                turn_direction = instruction.split("<b>")[1].split("</b>")[0].lower()

```

```

# Check if this "Turn" instruction immediately follows a "Head" instruction
if current_heading is not None and not junctions:
    # If it's the first instruction, add it without applying get_new_heading
    current_heading = current_heading
# Calculate new heading based on current heading and turn direction
else:
    new_heading = get_new_heading(current_heading, previous_turn_direction)
    current_heading = new_heading # Update current heading

# Extract latitude and longitude
start_location = step['start_location']
latitude = start_location['lat']
longitude = start_location['lng']

# Create a dictionary with latitude, longitude, and associated directions
location_with_directions = {
    'latitude': latitude,
    'longitude': longitude,
    'current_heading': current_heading,
}

# Append the dictionary to the junctions list
junctions.append(location_with_directions)

# Update the previous turn direction
previous_turn_direction = turn_direction

## Shift Coordinates to capture entire junction in frame
def shift_coordinates(current_latitude, current_longitude, current_heading):
    """
    Shift coordinates based on current heading.

    Args:
        current_latitude (float): Current latitude.
        current_longitude (float): Current longitude.
        current_heading (str): Current heading ("north", "south", "east", or "west").

    Returns:
        Tuple (new_latitude, new_longitude): New coordinates.
    """
    shift = 0.00017

    if current_heading == "north":
        new_latitude = current_latitude - shift
        new_longitude = current_longitude
    elif current_heading == "south":
        new_latitude = current_latitude + shift
        new_longitude = current_longitude
    elif current_heading == "east":
        new_latitude = current_latitude
        new_longitude = current_longitude - shift
    elif current_heading == "west":
        new_latitude = current_latitude

```



```

        new_longitude = current_longitude + shift
    else:
        raise ValueError("Invalid current_heading")

    return new_latitude, new_longitude

# Update the latitude and longitude in the dictionary
for location in junctions:
    new_latitude, new_longitude = shift_coordinates(location['latitude'], location['longitude'],
    location['current_heading'])

    location['latitude'] = new_latitude
    location['longitude'] = new_longitude

# Get Google Street View image of all junctions in the specified route
import requests
# Function to fetch Google Street View image
def fetch_street_view_image(latitude, longitude, heading):
    api_key = 'AIzaSyCRs4VDM3CDp4FBoZ8fZVJ6cg9XRd2Wxqc'
    size = '640x480'
    fov = 100
    heading = heading
    pitch = 0
    url =
f'https://maps.googleapis.com/maps/api/streetview?location={latitude},{longitude}&size={size}&fov
={fov}&heading={heading}&pitch={pitch}&key={api_key}'
    response = requests.get(url)
    if response.status_code == 200:
        return response.content
    else:
        print('Error: Unable to fetch Street View image')

# List to store fetched images
junction_images = []
# Create a list to store the feature vectors
junction_feature_vectors = []

heading_mapping = {
    "north": 0,
    "south": 180,
    "east": 90,
    "west": 270
}

# Fetch images for each set of coordinates
for i, location in enumerate(junctions):
    heading_value = heading_mapping.get(location['current_heading'], None)
    image_data = fetch_street_view_image(location['latitude'], location['longitude'], heading_value)

    if image_data is not None:

```

```

image_data = cv2.imdecode(np.frombuffer(image_data, np.uint8), cv2.IMREAD_COLOR)
image_data = cv2.resize(image_data, (224, 224))
junction_images.append(image_data)
# Calculate the feature vector and store it separately
feature_vector = get_feature_vector(image_data).ravel()
junction_feature_vectors.append(feature_vector)

# Get Frames
boto3.setup_default_session(region_name='ap-northeast-1')
s3 = boto3.client('s3')

# Define the S3 bucket and directory
bucket_name = 'junction-detection'
directory_name = 'frames' # Change this to the specific directory within the bucket

# List objects in the S3 bucket
response = s3.list_objects_v2(Bucket=bucket_name, Prefix=directory_name)

# Download the frames to a local directory
local_directory = '/home/ec2-user/frames'

if not os.path.exists(local_directory):
    os.makedirs(local_directory)

for obj in response.get('Contents', []):
    file_name = obj['Key']
    local_path = os.path.join(local_directory, os.path.basename(file_name))

    # Download the file
    s3.download_file(bucket_name, file_name, local_path)

frames_dir = '/home/ec2-user/frames'
frames_np = []
# List all files in the directory
files = os.listdir(frames_dir)

# Filter for image files (e.g., '.jpg', '.png', etc.)
image_files = [f for f in files if f.lower().endswith((''.jpg', '.jpeg', '.png', '.gif', '.bmp'))]

# Loop through the image files
for i in image_files:
    image_path = os.path.join(frames_dir, i)
    frames_np.append(load_image_into_numpy_array(image_path))

# Calculate Similarity
# Loop through the frames, if there is junction detected, check the similarity with junction_images
(contain the correct junctions to be taken).

```

```

confidence_threshold = 0.4
similarity_threshold_prev = 0.5
similarity_threshold = 0.7

previous_junction_image = None
junction_image_counter = 0

for i in range(len(frames_np)):

    comparison_result = 0
    image_np = frames_np[i] # Extract the image numpy array
    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, axis=0), dtype=tf.float32)
    detections = loaded_detection_fn(input_tensor)

    # Filter out detections with confidence below the threshold
    filtered_indices = detections['detection_scores'][0].numpy() >= confidence_threshold
    filtered_boxes = detections['detection_boxes'][0].numpy()[filtered_indices]

    if len(filtered_boxes) > 0:
        # If there are detections above the threshold, process the image
        print('Frame', i+1, 'contains junctions')

        # Get the image containing junctions
        junction_frame = get_feature_vector(image_np).ravel()

        if previous_junction_image is None:
            # If it's the first detected junction frame, compare it with the first junction image
            comparison_result = calculate_similarity(junction_frame,
            junction_feature_vectors[junction_image_counter])

        else:
            # Compare the current frame with the most recently compared junction image
            similarity_score = calculate_similarity(junction_frame, previous_junction_image)

            # If the similarity score is above a certain threshold, consider it the same junction
            if similarity_score > similarity_threshold_prev:
                print("Same junctions captured, skipping.....")
                continue # Skip further processing for this frame

            # Now you can compare the frame with the "correct_junctions" data
            comparison_result = calculate_similarity(junction_frame,
            junction_feature_vectors[junction_image_counter])

        # Print similarity score
        print('Comparison Result:', comparison_result)

        # Check if the similarity score is above the threshold for triggering audio
        if comparison_result > similarity_threshold:
            print("Audio: 1")
            #trigger_raspi_audio(positive)
        else:
            print("Audio: 0")
            #trigger_raspi_audio(negative)

```

```
# Update the previous junction image with the current frame
previous_junction_image = junction_frame
junction_image_counter+=1

else:
# If no detections above the threshold, no junctions detected, discard the image
print('Frame', i+1, 'does not contain junctions, skipping.....')
```

A.3 Evaluation.ipynb

```
import tensorflow
import matplotlib
import matplotlib.pyplot as plt

import os
# import sys
# sys.path.append('/path/to/tf_slim')
import random
import io
import imageio
import glob
import scipy.misc
import numpy as np
from six import BytesIO
from PIL import Image, ImageDraw, ImageFont
from IPython.display import display, Javascript
from IPython.display import Image as IPyImage

import tensorflow as tf

from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
#from object_detection.utils import colab_utils
from object_detection.builders import model_builder

%matplotlib inline

def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.

    Puts image into numpy array to feed into tensorflow graph.
    Note that by convention we put it into a numpy array with shape
    (height, width, channels), where channels=3 for RGB.

    Args:
        path: a file path.

    Returns:
        uint8 numpy array with shape (img_height, img_width, 3)
    """
    img_data = tf.io.gfile.GFile(path, 'rb').read()
    image = Image.open(BytesIO(img_data))
    image = image.resize((1024, 1024))
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)

#for multiple boxes
def plot_detections(image_np,
                    boxes,
                    classes,
```

```

        scores,
        category_index,
        figsize=(12, 16),
        image_name=None,
        title=None):
"""Wrapper function to visualize detections.

Args:
image_np: uint8 numpy array with shape (img_height, img_width, 3)
boxes: a numpy array of shape [N, 4]
classes: a numpy array of shape [N]. Note that class indices are 1-based,
        and match the keys in the label map.
scores: a numpy array of shape [N] or None. If scores=None, then
        this function assumes that the boxes to be plotted are groundtruth
        boxes and plot all boxes as black with no classes or scores.
category_index: a dict containing category dictionaries (each holding
        category index `id` and category name `name`) keyed by category indices.
figsize: size for the figure.
image_name: a name for the image file.
"""
image_np_with_annotations = image_np.copy()
if scores is None:
    scores = np.ones_like(classes, dtype=np.float32)
viz_utils.visualize_boxes_and_labels_on_image_array(
    image_np_with_annotations,
    boxes,
    classes,
    scores,
    category_index,
    use_normalized_coordinates=True,
    min_score_thresh=0.35,
    line_thickness=2)
if image_name:
    plt.imsave(image_name, image_np_with_annotations)
else:
    plt.figure(figsize=figsize)
    plt.imshow(image_np_with_annotations)
    if title:
        plt.title(title)
    plt.show()

# Load the saved model
loaded_module = tf.saved_model.load('C:/Users/xiaoq/Downloads/FYP 1024 resnet/FYP2_retrain4')
loaded_detection_fn = loaded_module.detect
loaded_fine_tuned_model = loaded_module.detection_model

# Redefine variable
junction_class_id = 1
num_classes = 1
category_index = {junction_class_id: {'id': junction_class_id, 'name': 'junction'}}

# Test Images
frames_dir = 'output'
frames_np = []

```

```

# List all files in the directory
files = os.listdir(frames_dir)

# Filter for image files (e.g., '.jpg', '.png', etc.)
image_files = [f for f in files if f.lower().endswith(('.jpg', '.jpeg', '.png', '.gif', '.bmp'))]
print(len(image_files))
# Loop through the image files
for i in image_files:
    image_path = os.path.join(frames_dir, i)
    frames_np.append(np.expand_dims(load_image_into_numpy_array(image_path), axis=0))
binary_predictions = []
label_id_offset = 1

# Loop through the image files and perform detection
for i in range(len(frames_np)):
    input_tensor = tf.convert_to_tensor(frames_np[i], dtype=tf.float32)
    detections = loaded_detection_fn(input_tensor)

    found_positive = False # Flag to check if a positive detection was found

    for detection_score in detections['detection_scores'][0].numpy():
        if detection_score > 0.35:
            binary_predictions.append(1)
            found_positive = True
            break # Exit the inner loop if a positive detection is found

    if not found_positive:
        binary_predictions.append(0)

# Now, binary_predictions contains the predictions

# Evaluation Metric
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
labels_from_file = []
# Read the text file and extract labels
with open('output/ground_truth.txt', 'r') as file:
    for line in file:
        # Remove any leading/trailing whitespace and convert the line to an integer
        label = int(line.strip())
        labels_from_file.append(label)

# Calculate accuracy
accuracy = accuracy_score(labels_from_file, binary_predictions)

# Calculate F1 score
f1 = f1_score(labels_from_file, binary_predictions)

print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(confusion)
print(f'F1 Score: {f1:.2f}')

```

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	Study week no.:2
Student Name & ID: Tan Yong Ming , 2005117	
Supervisor: Dr. Aun Yichiet	
Project Title: RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION	

1. WORK DONE

- perform discussion with supervisor and groupmate

2. WORK TO BE DONE

- figure out the junction that will be used for the training as well as testing set

3. PROBLEMS ENCOUNTERED

- no

4. SELF EVALUATION OF THE PROGRESS

- On track



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	Study week no.:4
Student Name & ID: Tan Yong Ming, 2005117	
Supervisor: Dr. Aun Yichiet	
Project Title: RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION	

1. WORK DONE

- obtain more data on junction images

2. WORK TO BE DONE

- insert the new data and retrain the model

3. PROBLEMS ENCOUNTERED

- no

4. SELF EVALUATION OF THE PROGRESS

- On track



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	Study week no.:6
Student Name & ID: Tan Yong Ming, 2005117	
Supervisor: Dr. Aun Yichiet	
Project Title: RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION	

1. WORK DONE

- retrained model to get higher validation score

2. WORK TO BE DONE

- discuss with groupmate about the setup

3. PROBLEMS ENCOUNTERED

- no

4. SELF EVALUATION OF THE PROGRESS

- On track



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	Study week no.:8
Student Name & ID: Tan Yong Ming, 2005117	
Supervisor: Dr. Aun Yichiet	
Project Title: RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION	

1. WORK DONE

- configure raspi audio

2. WORK TO BE DONE

- transfer trained model

3. PROBLEMS ENCOUNTERED

- no

4. SELF EVALUATION OF THE PROGRESS

- On track



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	Study week no.:10
Student Name & ID: Tan Yong Ming, 2005117	
Supervisor: Dr. Aun Yichiet	
Project Title: RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION	

1. WORK DONE

- transfer trained model to raspberry pi, install library and configure new OS

2. WORK TO BE DONE

- finish up the report and the code

3. PROBLEMS ENCOUNTERED

- no

4. SELF EVALUATION OF THE PROGRESS

- On track



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	Study week no.:12
Student Name & ID: Tan Yong Ming , 2005117	
Supervisor: Dr. Aun Yichiet	
Project Title: RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION	

1. WORK DONE

- write report and finalise the code

2. WORK TO BE DONE

- recheck the report and make sure the code is working fine

3. PROBLEMS ENCOUNTERED

- no

4. SELF EVALUATION OF THE PROGRESS

- On track



Supervisor's signature



Student's signature

RIGHT TRACK- A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION

FACULTY OF INFORMATION
AND TECHNOLOGY



INTRODUCTION

Improve road navigation with real-time junction detection and clear audible feedback to boost driver confidence and reduce navigation errors.

OBJECTIVE

To develop a computer vision model that is capable of detecting junctions with the assistance of cloud resources.

METHODOLOGY

- 1** Provide audible feedback to assist the driver in making the correct turn
- 2** Utilising cloud resources to ensure performance of the junction detection model

Project Developer: Tan Yong Ming
Project Supervisor: Dr Aun Yi Chiet

CONTRIBUTIONS

Leveraging a pre-trained model from TensorFlow as a foundation then fine-tuned and tailored it to suit the objective of the project and then employed the model to the cloud environment to enable efficiency and effectiveness enhancing of junction detection system.

CONCLUSION

Proposed a system that utilises cloud resources with the ability to detect junctions and the correct turn while providing audible feedback to drivers.

PLAGIARISM CHECK RESULT

RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION

ORIGINALITY REPORT

6%

SIMILARITY INDEX

5%

INTERNET SOURCES

2%

PUBLICATIONS

3%

STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Universiti Tunku Abdul Rahman Student Paper	1%
2	eprints.utar.edu.my Internet Source	1%
3	downloads.hindawi.com Internet Source	<1%
4	robotics.iiit.ac.in Internet Source	<1%
5	fict.utar.edu.my Internet Source	<1%
6	Submitted to Sim University Student Paper	<1%
7	Andrew Campbell, Alan Both, Qian (Chayn) Sun. "Detecting and mapping traffic signs from Google Street View images using deep learning and GIS", Computers, Environment and Urban Systems, 2019 Publication	<1%

8	Nick Alteen, Jennifer Fisher, Casey Gerena, Wes Gruver, Asim Jalis, Heiwad Osman, Marife Pagan, Santosh Patlolla, Michael Roth. "AWS® Certified Developer Official Study Guide", Wiley, 2019 Publication	<1 %
9	Submitted to University of Ghana Student Paper	<1 %
10	projects-raspberry.com Internet Source	<1 %
11	bestcourses.org Internet Source	<1 %
12	www.ncbi.nlm.nih.gov Internet Source	<1 %
13	Jaspreet Kaur Bhamra, Shreyas Anantha Anantha Ramaprasad, Siddhant Baldota, Shane Luna et al. "Multimodal Wildland Fire Smoke Detection", Remote Sensing, 2023 Publication	<1 %
14	Joonwoo Ahn, Yangwoo Lee, Minsoo Kim, Jaeheung Park. "Vision-Based Branch Road Detection for Intersection Navigation in Unstructured Environment Using Multi-Task Network", Journal of Advanced Transportation, 2022 Publication	<1 %

15	Submitted to St Peter's College Student Paper	<1 %
16	Submitted to Submitted on 1687709076728 Student Paper	<1 %
17	Submitted to University of Greenwich Student Paper	<1 %
18	en.wikipedia.org Internet Source	<1 %
19	www.researchgate.net Internet Source	<1 %
20	Shenshen Liu, Shunfang Wang. "An Improved Face Recognition Fusion Algorithm Based on the Features extracted from Gabor, PCA and KPCA", Proceedings of the 2nd International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence, 2018 Publication	<1 %
21	ojs.uma.ac.id Internet Source	<1 %
22	Kyle Ong, Kok-Why Ng, Yih-Jian Yoong. "3D Shape Blending", Proceedings of the 2019 2nd International Conference on Computational Intelligence and Intelligent Systems, 2019 Publication	<1 %

23 Reagan L. Galvez, Argel A. Bandala, Elmer P. Dadios, Ryan Rhay P. Vicerra, Jose Martin Z. Maningo. "Object Detection Using Convolutional Neural Networks", TENCON 2018 - 2018 IEEE Region 10 Conference, 2018
Publication <1%

24 docplayer.net
Internet Source <1%

25 Sai Deepak Alapati, Muthukumar Arunachalam, Chandana Chennamsetty, Pujitha Dantam, Anusha Dabbara. "chapter 8 Real-Time Object Detection in Video for Traffic Monitoring", IGI Global, 2023
Publication <1%

Exclude quotes On
Exclude bibliography On

Exclude matches < 8 words

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	TAN YONG MING
ID Number(s)	20ACB05117
Programme / Course	BACHELOR OF COMPUTER SCIENCE (HONOURS)
Title of Final Year Project	RIGHT TRACK – A GOOGLE MAP COMPANION USING JUNCTION RECOGNITION

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)
Overall similarity index: <u> 6 </u> % Similarity by source Internet Sources: <u> 5 </u> % Publications: <u> 2 </u> % Student Papers: <u> 3 </u> %	
Number of individual sources listed of more than 3% similarity: <u> 0 </u>	
Parameters of originality required and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor

Name: Dr. Aun Yichiet

Date: 15/9/2023

Signature of Co-Supervisor

Name: _____

Date: _____



UNIVERSITI TUNKU ABDUL RAHMAN

**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)**

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB05117
Student Name	TAN YONG MING
Supervisor Name	DR. AUN YICHIE

TICK (√)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date:12/9/2023