REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING

BY

TAN YI XUAN

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

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING

Academic Session: MAY 2023

I TAN YI XUAN

(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.

[illegible]

Verified by, [illegible]

________________________	________________________
(Author's signature)	(Supervisor's signature)

Address:
26, JALAN FLORA 3F/16,
TELOK PANGLIMA GARANG
42500, SELANGOR

_____Dr. Aun Yichiet______
Supervisor's name

Date: 12/9/2023

Date: __15 Sep 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 Yi Xuan***______________________________ (ID No: __***21ACB01124***) has completed this final year project/ dissertation/ thesis* entitled "____________*REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING* _______" 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,

[illegible]

Tan Yi Xuan
(*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	:	[illegible]
Name	:	Tan Yi Xuan
Date	:	12/9/2023

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr. Aun Yichiet who has given me this bright opportunity to engage in a junction recognition image matching task.

And my project group mate, Tan Yong Ming for collaborating seamlessly.

ABSTRACT

In an age of advanced navigation technology, urban road complexity still poses challenges, leading to missed turns and potential hazards. This research endeavors to mitigate these issues by developing a computer vision-based intersection detection system. The project leverages the synergy of Google Directions API and Google Street View (GSV) API to automate route and junction coordination retrieval. The study focuses on a neighborhood in Westlake, Kampar, UTAR, where real-world urban complexity is scaled down for testing. The objective is to instill driver confidence and safety by providing timely and accurate junction notifications. The deliverables include automated route retrieval, junction coordinates, and a mechanism to filter and compare video frames, thereby improving navigation in complex urban environments. By integrating Google Directions API and GSV, our project aims to revolutionize the way drivers navigate unfamiliar roads. Our system not only enhances safety but also streamlines the navigation process by reducing cognitive load. The intelligent frame filtering mechanism represents a significant contribution to the efficiency of the system, ensuring that drivers receive timely and accurate notifications. Overall, this research stands as a testament to our commitment to improving urban navigation and safety through innovative computer vision solutions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

φ	Phi (representing latitude)
λ	Lambda (representing longitude)

LIST OF ABBREVIATIONS

GSV	Google Street View
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
XML	Extensible Markup Language
VBA	Visual Basic for Applications
OSM	Open Street Map
GCP	Google Cloud Platform

Chapter 1

Introduction

In this chapter, I will present the background, problem statement and motivation of our research. My contribution to this field through this paper is also highlighted in this chapter.

1.1 Problem Statement and Motivation

In today's age of advanced road navigation systems and applications like Google Maps and Waze, finding our way around cities has become significantly more manageable. However, the intricacies of urban road networks often present challenges that these navigation tools struggle to overcome. Taking a wrong turn in a bustling city is a common occurrence, and the need to frequently glance at a mobile screen for route updates can not only be inconvenient but also hazardous. Despite the progress made in navigation technology, drivers still grapple with unfamiliar road layouts, resulting in numerous accidents and disruptions.

To address these challenges, our research endeavors to develop a computer vision model capable of accurately predicting the appropriate turns for drivers. One specific problem we aim to tackle involves situations there are flyovers and the two roads will be sharing the same space from a top-down perspective like in Figure 1.1.1. Conventional navigation apps might fail to recognize that a driver has taken the wrong turn in such scenarios, delaying route corrections. Our objective is to enhance driver confidence and safety by promptly identifying intersection points and notifying drivers of incorrect turns.

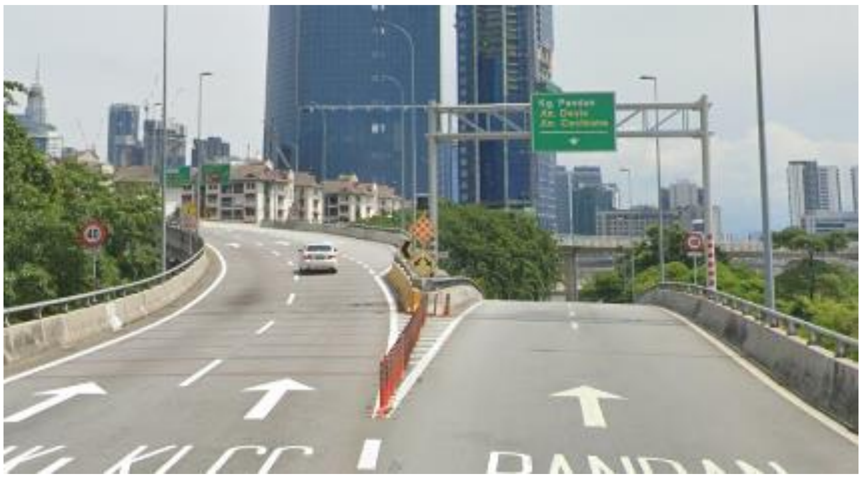


Figure 1.1.1 Flyover with two roads sharing same space

1.2 Objectives

The objective of this research is to develop and evaluate a computer vision model that utilizes image comparison techniques to accurately recognise the appropriate turn for drivers based on digital images captured by a camera installed in front of a vehicle. The model will compare the captured images with images of correct intersections from Google Street View API and will be integrated into a Raspberry Pi device that will provide audible alerts to the driver, signaling the correct turn or warning against incorrect turns.

1.3 Project Scope and Direction

It's essential to note that our research scope focuses on a specific geographical area within Westlake, Kampar, UTAR. Here, we have scaled down our study to neighborhood-sized junctions, as obtaining appropriate test images for complex flyover scenarios proved challenging. By providing drivers with accurate, real-time guidance in these neighborhood settings, we aim to demonstrate the potential of our approach and set the stage for future advancements in urban navigation systems. Ultimately, our research strives to make city navigation more efficient and secure for all road users.

I will be using Google APIs in this project, namely Google Direction API and Google Street View API.

The deliverables of this project will include automation of route and junction coordinates retrieval and an junction recognition algorithm that filters video frames that depict the same junction and compares distinct junctions with the correct junctions.

1.4 Contributions

In this research project, I have made several contributions towards the development of a computer vision model for turn prediction using street images captured by a camera installed in front of a vehicle. My contributions are as follows:

Proposing a novel approach: the innovative integration of the Google Directions API and Google Street View (GSV) API to provide a holistic and real-time view of the driver's journey. By seamlessly combining these two powerful tools, I introduced a novel approach to address the challenges of road navigation in complex urban environments.

These contributions advance the field of computer vision and have implications for the development of practical solutions for turn recognition in real-world driving scenarios.

1.5 Report Organization

The details of this research are shown in the following chapters. In Chapter 2, some related backgrounds are reviewed. Then, the system mythology and general work procedures is discussed in Chapter 3. Chapter 4 explains the system design diagram and its components. Chapter 5 shows the settings, configurations and system operation. And then, Chapter 6 reports the result and feasibility of the proposed method. Lastly, Chapter 7 details the conclusion of this project.

Chapter 2
Literature Review

2.1 Previous Works on using Google Directions API

The Google Directions API is a fundamental tool in modern navigation systems, enabling users to get precise directions, plan routes, and access geospatial data. This API taps into Google's vast geographical database and real-time traffic information to provide turn-by-turn directions. It takes addresses of origin and destination as input and factors in variables like traffic congestion and road closures to find the quickest path.

In addition to directions, this API supports route planning, allowing users to customize trips based on transportation mode (e.g., driving, walking, or public transit), and it offers data on travel times, distances, and expected arrival times. It's also flexible, accommodating waypoints or stops, which is valuable for logistics and delivery businesses.

Most importantly, the Google Directions API offers geospatial data retrieval. It provides access to location-related data, including coordinates, place details, and geographic boundaries. This feature is crucial for applications like geofencing, location-based marketing, and spatial analysis. This aspect is utilized in my project as will be discussed later in the subsequent chapters. In essence, the Google Directions API serves as a foundational component of modern navigation and location-based services.

The Google Directions API [6] functions through a straightforward process. When implementing it, an HTTP request is dispatched to a specific address, typically "http://maps.googleapis.com/maps/api/directions," with various parameters. These parameters include:

- **Origin Point**: This specifies the starting location of the trip.
- **Waypoints**: These are optional waypoints or intermediate locations that the route should pass through during the journey.

- **Destination Point**: This marks the final endpoint where the trip concludes.

Upon receiving this request, the Google Directions API processes the provided information to calculate the optimal route, considering factors like distance, traffic conditions, and available roadways. It then generates a response, usually in JSON or XML format, containing detailed navigation instructions, waypoints, distance, estimated travel time, and other relevant data.

The authors in [5] have utilize these features to develop a navigation system for visually impaired individuals to generate the optimal route. However, their evaluation highlights limitations in the Google Directions API when applied to outdoor navigation for the visually impaired. Another study [7] aimed to find an optimal vehicle routing scheme for delivering goods and services to multiple convenience stores using web scraping and Excel VBA. They utilized the Google Directions API to develop a computer program for solving the optimal networking problem, enhancing the efficiency of goods delivery. This showcases Google Directions API's versatility in solving complex logistical challenges and its potential to enhance cost-effective and efficient supply chain operations.

This study [8] introduces a different use case for the Google Directions API, specifically in the context of measuring public transit accessibility. It demonstrates that the API can provide shorter access distances in areas with more public access points. This finding suggests that open cloud services like the Google Directions API can serve as valuable alternatives for measuring public transit accessibility, benefiting transit researchers and agencies.

2.1.1 Weakness

Even though the Google Directions API is a powerful tool for navigation, it may not consistently provide precise destination points, leading to deviations from the intended final locations. In certain scenarios, especially those involving irregular paths or densely spaced waypoints, the API can struggle to accurately pinpoint the exact destination. This limitation results in users reaching their destinations at points slightly away from the intended endpoint a shown in Figure 2.1.1.1 [5]. Consequently, addressing this issue becomes crucial for improving the accuracy and reliability of navigation applications that rely on the Google Directions API.

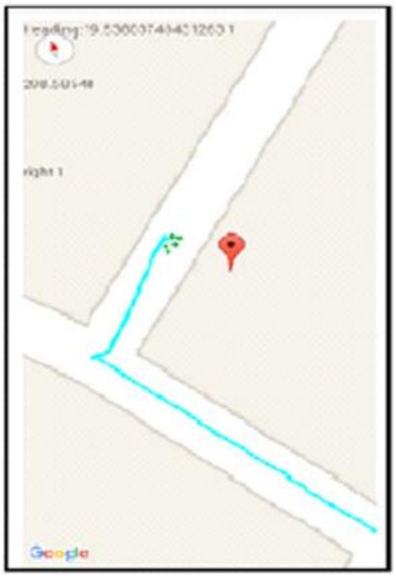


Figure 2.1.1.1 Destination that is slightly apart from the endpoint

2.2 Previous Works on Using Google Street View Image API

Google Street View (GSV) is a technology integrated into Google Maps and Google Earth, offering panoramic street views from various locations worldwide. Initially launched in 2007 with limited coverage, GSV has expanded to encompass an extensive image dataset across 48 countries. Equipped with nine directional cameras, GPS units, laser range scanners, and connectivity options, GSV captures detailed street-level imagery. Data collection involves driving vehicles to photograph locations, with factors like weather and population density influencing timing and location choices. Advanced sensor data allows precise geographic mapping of images, enabling the creation of seamless 360-degree views through image processing algorithms. This resource presents a powerful alternative to traditional asset data collection methods, potentially enhancing or even replacing existing practices for certain applications. [9]

The Google Street View API is a crucial component of modern location-based applications, allowing developers to seamlessly integrate street-level imagery into their projects. This API grants straightforward access to Google's vast repository of visual data, enabling developers to incorporate panoramic views of locations from any location effortlessly.

The Google Street View API offers various features, including the retrieval of static images, dynamic panoramas, and 360-degree photos. These visual assets can be easily integrated into various applications such as mapping services, tourism guides, and real estate platforms, offering users a virtual tour of places as if they were there. Additionally, the imagery is also used in machine learning projects that require the use of street-level images. This versatility makes the GSV a valuable resource for developers looking to enhance user experiences and leverage street-level visual data in a wide range of applications.

Google Street View (GSV) provides a virtually continuous tour of city streets, offering users a sensation akin to being present in that location. It closely resembles the experience of exploring a city through traditional means like driving, cycling, or walking. This is why it is used in projects that required street-level images, like in [2] where they perform urban greenery assessment using static GSV images. To achieve their project objective, they captured the GSV images in six directions and three vertical view angles as they intend to get the 360° perspective of a pedestrian as shown in Figure 2.2.1. In our case, we aim to capture the view of a driver so we will be only capturing the GSV image in 1 single direction. In [10], the authors employed a similar approach as their aim is to capture images of traffic signs, given that these signs typically face the road in a single direction.

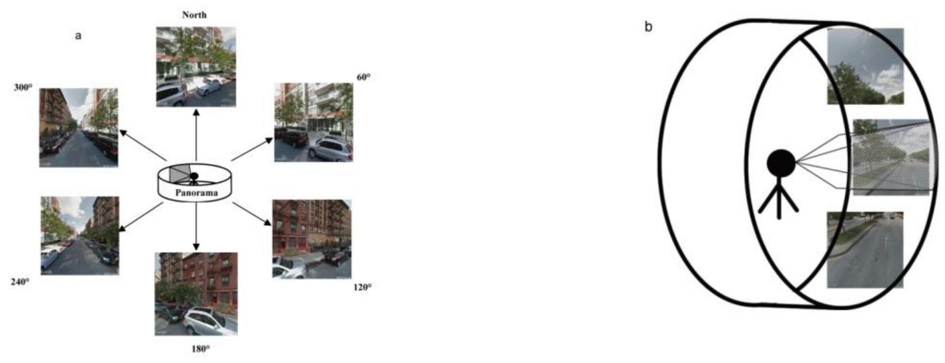


Figure 2.2.1 GSV images captured in six directions at a sample site in the study area (a) and GSV images captured at three vertical view angles at a sample site (b)

Authors is [10] used GSV for an entirely different use case, lane detection. They assessed numerous works on lane detection that used Open-Street-Map (OSM) [11] for acquiring test images based on geographical data. However, OSM lacks the capability to provide the current system location, as indicated in [12], rendering it unable to obtain test images at the system's present location. This limitation can be overcome by leveraging GSV APIs, which enable the retrieval of the system's current location.

Although I couldn't locate any prior research utilizing GSV images for junction similarity matching in the existing literature, I chose to investigate whether employing GSV images for calculating junction similarity would be viable. This decision was influenced by studies hinting at the potential effectiveness and efficiency of this approach.

2.2.1 Strength and Weaknesses

[3] effectively collected data related to the visual appeal of an environment, the availability of infrastructure for active transportation, and the quality of sidewalk amenities. However, they encountered difficulties in obtaining meaningful results concerning motorized traffic and parking. Similarly, [4] faced limitations when trying to assess the locations of a seasonal tree-nesting species. The variability in dates of Google Street View (GSV) imagery presents a significant challenge when extracting such data.

2.3 Previous Works on Route Validation

Route validation, in the context of confirming that a driver is adhering to the intended path, holds a crucial role in modern navigation and driver assistance systems. It serves as a real-time feedback mechanism that ensures drivers stay on course and make the correct turns, reducing the likelihood of errors and enhancing road safety. By cross-referencing the driver's progress with the intended route, these systems can provide timely alerts and corrective instructions if deviations occur. The importance of route validation becomes particularly evident in scenarios where precise navigation is critical, such as emergency response vehicles, logistics, and public transportation. It not only aids in preventing wrong turns and missed destinations but also contributes to smoother traffic flow and optimized travel times. In essence, route validation plays a vital role in enhancing driver confidence, reducing navigational stress, and ultimately improving the efficiency and reliability of road journeys.

2.4 Previous Works on Image Similarity Models in Traffic Analysis

The importance of image similarity models in the realm of traffic analysis, especially when it comes to recognizing junctions and assessing road conditions, cannot be overstated. These models represent a critical technological advancement that leverages computer vision and machine learning techniques to process vast amounts of visual data generated by traffic cameras, dashcams, and other imaging sources. In doing so, they provide invaluable insights into the dynamic and often complex nature of roadways, profoundly impacting road safety, transportation efficiency, and overall traffic management.

One of the primary functions of image similarity models is the precise identification of junctions, a task that holds immense significance in route planning and navigation systems. Recognizing junctions accurately ensures that drivers receive timely and reliable directions, reducing the risk of missed turns and navigational errors. This is particularly crucial in urban environments, where intricate networks of intersections and roadways can pose challenges for both human drivers and automated navigation systems. Image similarity models excel at this task by comparing real-time visual data with a database of reference images, allowing for seamless junction recognition and guidance.

Chapter 3

System Methodology/Approach

3.1 Study Area and data

Our research was conducted in the Westlake, a neighborhood just beside UTAR, Kampar. Figure 3.2.1 below shows the location where we obtained our sample test cases.

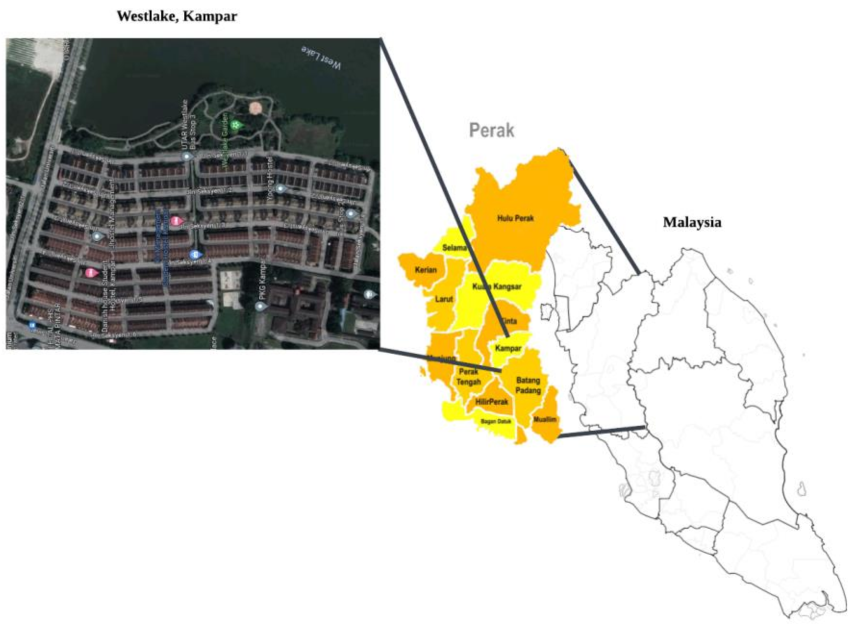


Figure 3.1.1 Location of Test Case

3.2 Methodology and General Work Procedures

3.2.1 Google Directions API Call

In this section, we outline the procedures and steps involved in obtaining route information using the Google Directions API. The Google Directions API serves as a pivotal component in our intersection detection project, allowing us to retrieve precise navigation instructions and route details.

3.2.1.1 API Key Acquisition

To access the Google Directions API, we initiated the process by obtaining a unique API key from the Google Cloud Platform (GCP). This API key serves as the authentication mechanism for our requests, enabling us to utilize the API's functionalities securely.

3.2.1.2 API Endpoint

Our API calls were directed to the specific Google Directions API endpoint, which comprises the base URL and a designated path for route-related requests. The endpoint URL was structured to target the Directions API accurately.

3.2.1.3 Request Parameters

We configured our API requests with a set of parameters to tailor the route information to our requirements. These parameters included:

Table 3.2.1.3.1 Parameters of Google Directions API

Parameters	Description
Origin	The starting location of the trip.
Destination	The ending point of the route
Waypoints (Optional)	Optional waypoints or intermediate locations that the route should pass through during the journey.
Mode (Optional)	The preferred method of transportation, like driving or walking.

3.2.1.4 HTTP Request Method

Our API requests were made using the HTTP GET method, which is the standard practice for retrieving directions and route-related data from the Google Directions API.

3.2.1.5 Handling Response

The obtained JSON result from the Google Directions API was processed to extract specific data related to junctions and their associated cardinal directions. The process involved iterating through the JSON structure, which contained route information.

For each route, the script traversed through its legs and steps, focusing on the provided instructions. It identified headings and turns in the instructions by checking for specific keywords like "Head" and "Turn." When a heading was encountered, it extracted the cardinal direction associated with it, and when a turn instruction was found, it extracted the turn direction.

To ensure accurate cardinal direction assignment, the script considered the context of each turn instruction. If a "Turn" instruction immediately followed a "Head" instruction, it was treated differently from subsequent turn instructions. The script utilized a function called get_new_heading to calculate the new heading based on the current heading and the turn direction.

Additionally, the latitude and longitude coordinates of the start location for each step were extracted. These coordinates, along with the associated cardinal directions, were stored in a structured format as dictionaries. These dictionaries, representing location data with directions, were then appended to a list called "junctions."

The script also maintained a record of the previous turn direction to facilitate the calculation of new headings as the instructions were processed sequentially. This methodology allowed for the extraction and organization of data regarding junctions and their respective directions, which was crucial for subsequent analysis and processing in the project.

3.2.2 GSV API Call

This section outlines the methodology and procedures employed to access street-level imagery through the Google Street View API, a fundamental component of our intersection detection project. The Google Street View API enables us to retrieve panoramic images of specific locations, which are vital for our image-based intersection analysis.

3.2.2.1 API Key Acquisition

We initiated our interaction with the Google Street View API by acquiring an API key from the Google Cloud Platform (GCP). This API key served as the authentication mechanism for our requests, ensuring secure access to the API's services.

3.2.2.2 API Endpoint

Our API calls were directed to the designated Google Street View API endpoint. This endpoint was constructed with precision to target the specific functionalities of the Street View API, allowing us to fetch street-level images.

3.2.2.3 HTTP Request Method

API requests to the Google Street View API were made using the standard HTTP GET method. This method facilitated the retrieval of street view images for specified geographical coordinates. An example of requesting a GSV static image is shown below.

```
https://maps.googleapis.com/maps/api/streetview?location={latitude},{longitude}&size={size}&fov={fov}&heading={heading}&pitch={pitch}&key={api_key}
```

Figure 3.2.2.3.1 shows an example of a requested GSV static image.

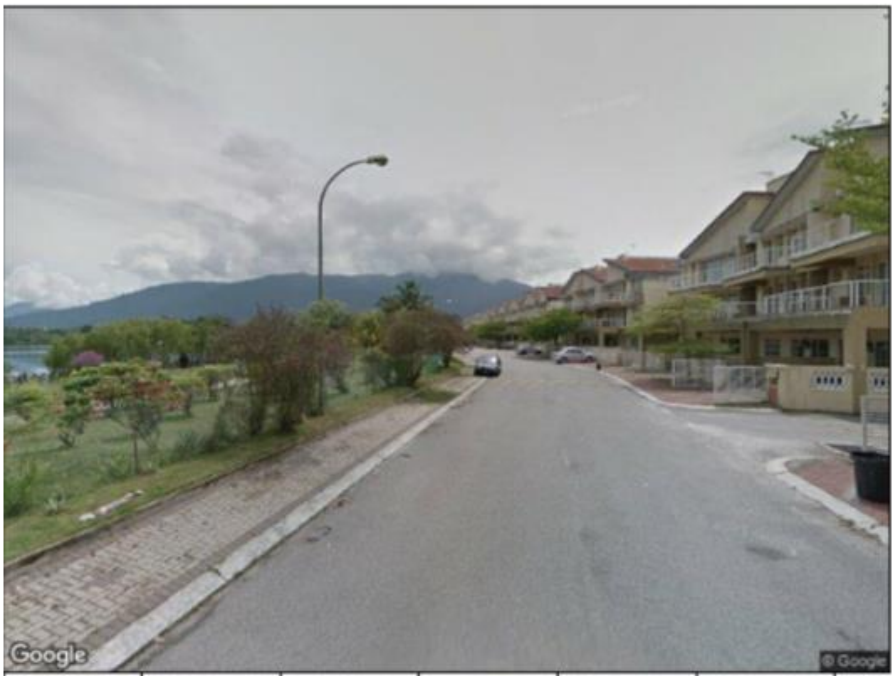


Figure 3.2.2.3.1 Example of a requested GSV static image

Table 3.2.2.3.1 Parameters of Google Street View API

Parameter	Description	Dimension
Location	Either text string or lat/long value	lat/long value
Size	Output size of the image in pixels	2048×2048
Heading	Horizontal field of view of the image	degrees
Pitch	Up/down angle of the camera relative to the Street View vehicle	0
Field of Vision (FOV)	Horizontal field of view of the image	degrees

Table 3.2.2.3.2 Cardinal Directions and Heading Mapping

Cardinal Directions	Heading
North	0
East	90
South	180
West	270

3.2.2.4 Handling Response

3.2.2.4.1 Shift Coordinates

Since the coordinates obtained from the Google Directions API are directly centered on the user's intended junctions (Figure 3.2.2.4.1.1), it becomes necessary to modify these coordinates. This modification entails shifting them to a position that allows for a better view of the junction within the frame of the Google Street View (GSV) image, ensuring that the junction is adequately captured.

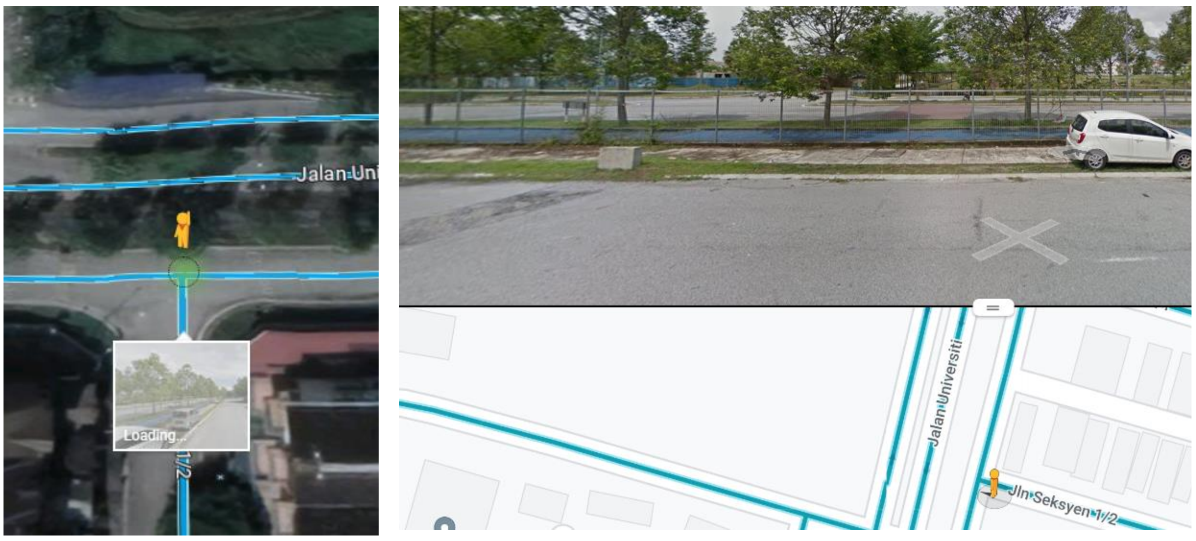


Figure 3.2.2.4.1.1 Street View of Coordinates obtained from Direction API (before shifting)

Figure 3.2.2.3.1.2 below shows that the junction can be captured in the image after shifting.

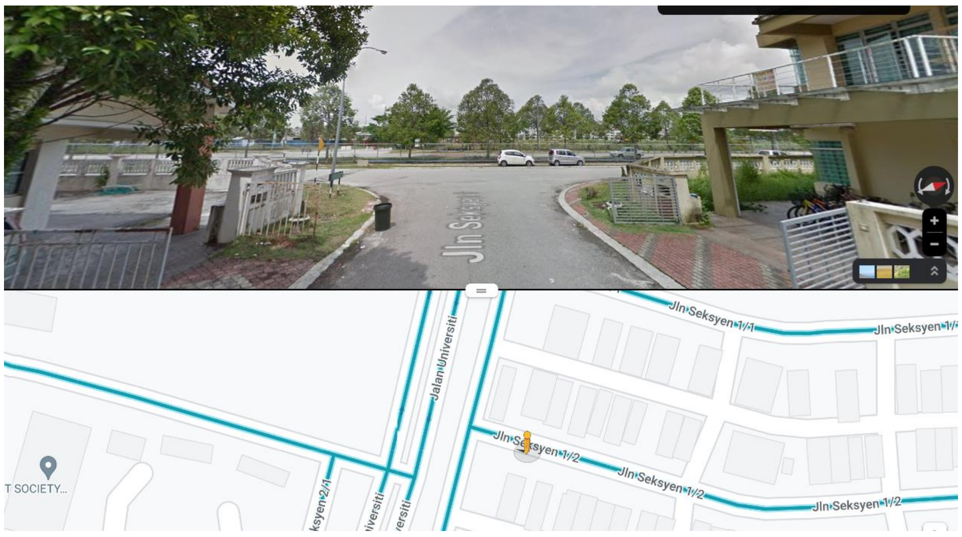


Figure 3.2.2.3.1.2 Street View of Coordinates obtained from Direction API (after shifting)

3.2.2.4.2 Handling Response

Utilizing the adjusted coordinates and heading values described in Chapter 3.2.1.5, the GSV API will retrieve street images and subsequently add them to a list known as 'junction_images.' This list will serve a crucial role in Chapter 3.2.3. The feature vectors of the images will be added to a separate list called ‘junction_feature_vectors’ for later use when calculating image similarity.

The adjustment of the new latitude and longitude coordinates is determined by the current heading of the route. If the current heading is 'north,' the offset is applied slightly in the southern direction. Similarly, for other headings, such as 'south,' 'east,' or 'west,' the offset is calculated accordingly to align with the intended route direction.

3.2.3 Image Similarity Model

Upon obtaining images with detected junctions, it's possible to encounter multiple images capturing the same junction as shown in Figure 3.2.2.4.1. To address this, an image similarity model is employed to assess the similarity between the current frame and the previous frame. This comparison helps determine whether the junction remains the same or if it's a different one.

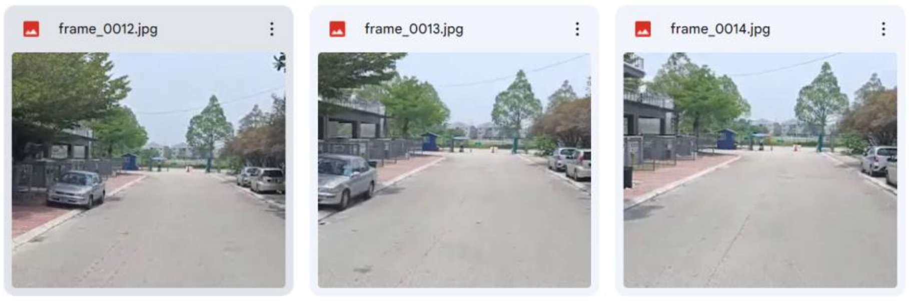


Figure 3.2.2.4.1 Different Frames Capturing the Same Junction

For distinct junctions, a comparison is made against specific ‘junction images' that have been predefined. These images represent specific junctions that the driver must navigate according to the planned route. Consequently, this process helps identify and confirm whether the detected junction corresponds to one of these predefined junctions.

3.3 System Design Equation

3.3.1 Equation of Coordinates Shifting

Let:

- $\varphi_{current}$ to be the current latitude in radians.
- $\lambda_{current}$ to be the current longitude in radians.
- Shift to be the offset value. Shift = 0.00017

The new latitude($\varphi_{current}$) and longitude ($\lambda_{current}$) [13] can be calculated based on the current heading (current_heading) as follows:

If current_heading = "north":
$$\varphi_{new} = \varphi_{current} - shift$$
$$\lambda_{new} = \lambda_{current}$$

If current_heading = "south":
$$\varphi_{new} = \varphi_{current} + shift$$
$$\lambda_{new} = \lambda_{current}$$

If current_heading = "east":
$$\varphi_{new} = \varphi_{current}$$
$$\lambda_{new} = \lambda_{current} - shift$$

If current_heading = "west":
$$\varphi_{new} = \varphi_{current}$$
$$\lambda_{new} = \lambda_{current} + shift$$

(1)

Chapter 4
System Design

4.1 System Component Specifications

4.1.1 System Design Diagram

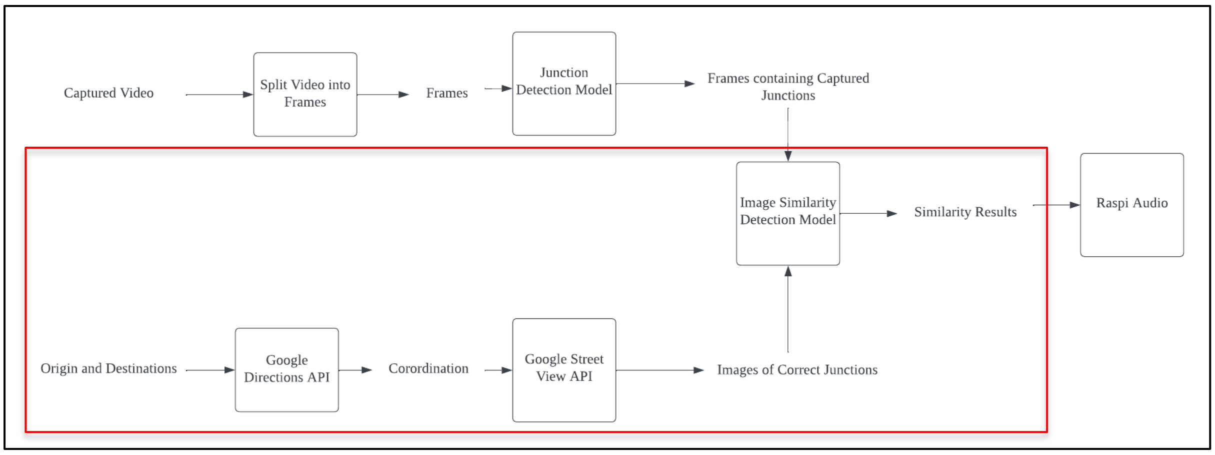


Figure 3.3.1 System Design Diagram

4.1.2 System design Diagram Component Specifications

4.1.2.1 Google Directions API

This block is used to provide route information. This component is responsible for making API requests to retrieve detailed route data, including navigation instructions and coordinates of any intersections to be taken. The API endpoint, request parameters, and the handling of API responses are all meticulously defined to facilitate effective communication with the Google Directions API.

The input for this block is Origin and Destination as the start and end point of a given route. After retrieving the JSON data returned by the API call, I process the data to extract coordination information and the cardinal direction of each intersection.

4.1.2.2 Google Street View API

In this block, the input consists of coordinates and cardinal directions for each intersection. These parameters are supplied to the API endpoint, and the outcome is a set of Google Street View (GSV) images depicting the respective intersections. Additionally, within this block, coordinate shifting is performed to guarantee that the entire junction is captured within the GSV images.

4.1.2.3 Image Similarity Detection Model

The input to this block comprises frames from captured images and a predefined set of correct junctions along the specified route. Before comparing the captured frames with the correct junctions, the block conducts similarity matching among frames to eliminate duplicates depicting the same junctions. After computing image similarity between distinct junctions and the correct junctions, the model will generate a "comparison_result" for each unique junction. This output will be utilized in the subsequent block to generate auditory feedback.

Chapter 5
System Implementation

5.1 Hardware Setup

The hardware involved in this project is a laptop. The laptop is used to call the GSV and Google Directions API and for hosting the code that will compare the results of the junction detection model and the obtained static GSV images.

Table 5.1 Specifications of laptop

Description	Specifications
Model	HUAWEI MATEBOOK 16S
Processor	INTEL I7-12700H
Operating System	Windows 11 Home (Version 21H2)
Graphic	Intel Iris Xe Graphics
Memory	16GB LPDDR5
Storage	1TB NVMe PCIe SSD

5.2 Software Setup

In the development of this project, I mainly used Google Colabotary

1. Google Colaboratory

   Version: 0.0.1a2
2. OpenCV

   Version: 4.6.0

   Release Date: 12/6/2022
3. Tensorflow (comes with Keras API)

   Version: 2.10.1
4. Numpy

   Version: 1.21.5
5. Pandas

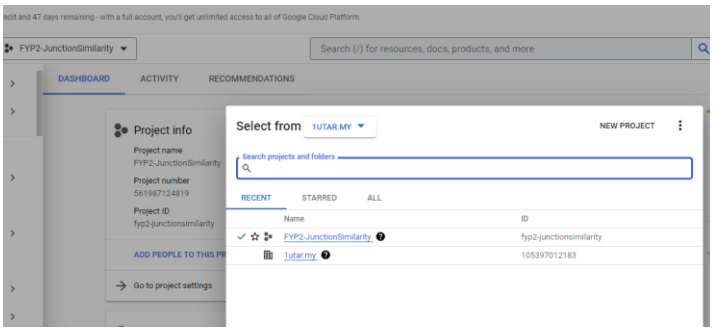
   Version: 1.4.4
6. Matplotlib

   Version: 3.5.2

5.3 Settings and Configuration

5.3.1 Google Cloud API Key

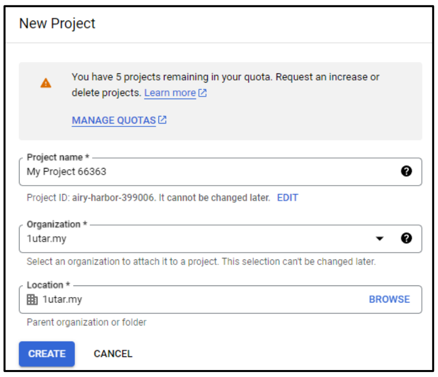
Step 1: Create Google Cloud account.
Step 2: Create a New Project


edit and 47 days remaining - with a full account, you'll get unlimited access to all of Google Cloud Platform.

Project info
Project name: FYP2-JunctionSimilarity
Project number: 561987124819
Project ID: fyp2-junctionsimilarity

Select from 1UTAR.MY

Name	ID
FYP2-JunctionSimilarity	fyp2-junctionsimilarity
1utar.my	105397012183

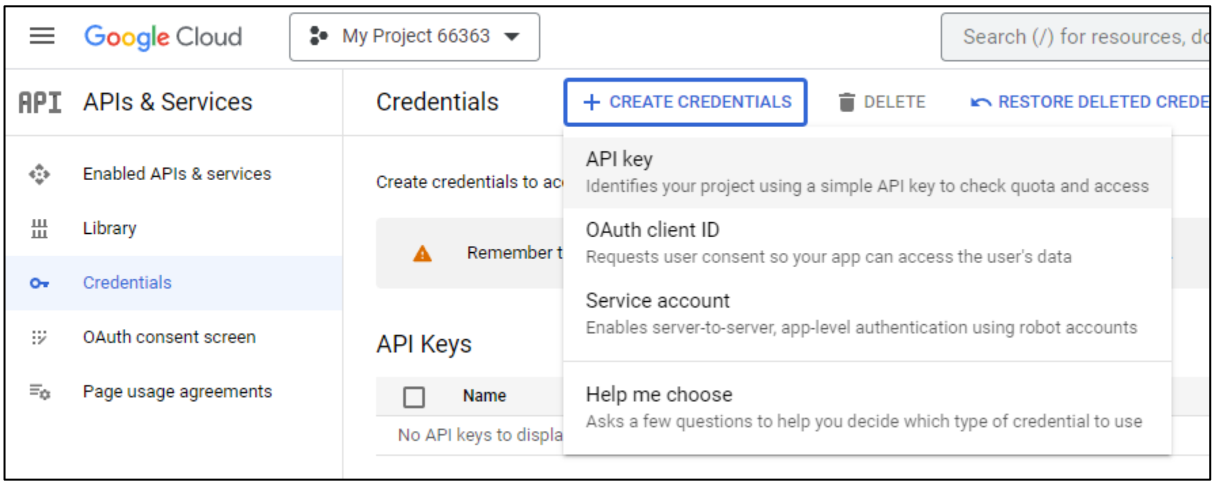

New Project

You have 5 projects remaining in your quota. Request an increase or delete projects. Learn more

Project name *: My Project 66363
Project ID: airy-harbor-399006. It cannot be changed later.
Organization *: 1utar.my
Select an organization to attach it to a project. This selection can't be changed later.
Location *: 1utar.my
Parent organization or folder

Figure 5.3.1.1 Creating a New Project in Google Cloud

Step 3: Select the project and click Create Credentials > API key



API key
Identifies your project using a simple API key to check quota and access

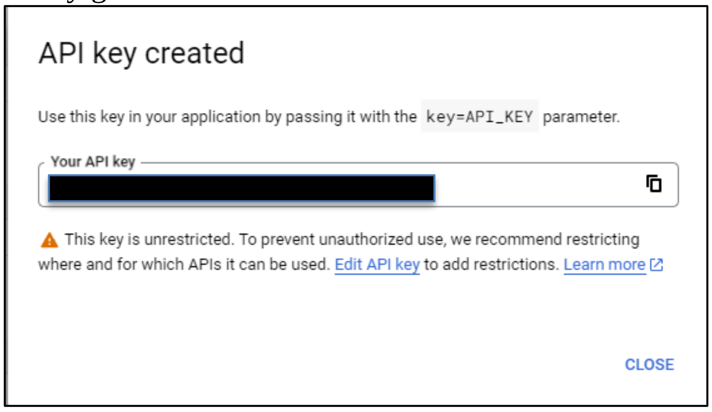
OAuth client ID
Requests user consent so your app can access the user's data

Service account
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

Figure 5.3.1.2 Generating an API key in Google Cloud

Step 4: Copy the API key generated.


API key created

Use this key in your application by passing it with the `key=API_KEY` parameter.

Your API key: [redacted]

This key is unrestricted. To prevent unauthorized use, we recommend restricting where and for which APIs it can be used. Edit API key to add restrictions. Learn more

Figure 5.3.1.3 Generated API key

5.4 System Operation

In this section, I will demonstrate a test case using the steps specified below.

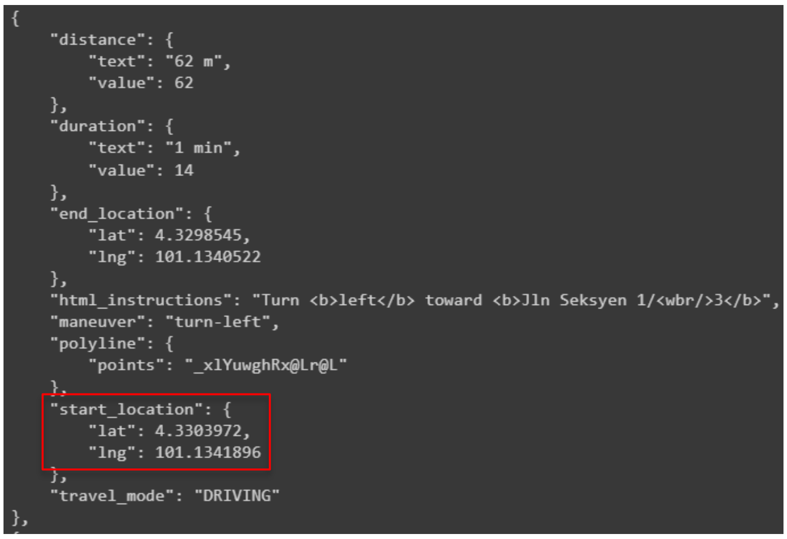
5.4.1 Google Directions API

The table below displays the parameters passed to the Google Directions API endpoint:

Table 5.4.1.1 Parameters Passed to Google Directions API endpoint

Parameters	Value
origin	'1066, Jln Seksyen 1/2, Taman Bandar Barat, 31900 Kampar, Perak'
destination	'1337, Jln Seksyen 1/3, Taman Bandar Barat, 31900 Kampar, Perak'
API key	***
mode	'driving'

Figure 5.4.1.1 shows a snippet of the JSON response from Google Directions API.

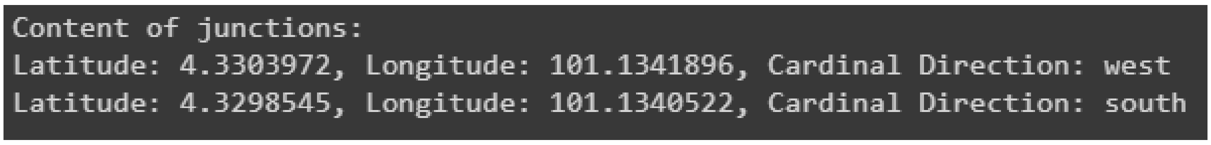


```
{
    "distance": {
        "text": "62 m",
        "value": 62
    },
    "duration": {
        "text": "1 min",
        "value": 14
    },
    "end_location": {
        "lat": 4.3298545,
        "lng": 101.1340522
    },
    "html_instructions": "Turn <b>left</b> toward <b>Jln Seksyen 1/<wbr/>3</b>",
    "maneuver": "turn-left",
    "polyline": {
        "points": "_xlYuwghRx@Lr@L"
    },
    "start_location": {
        "lat": 4.3303972,
        "lng": 101.1341896
    },
    "travel_mode": "DRIVING"
},
```

Figure 5.4.1.1 JSON response from Google Directions API

I implemented a "get_new_heading" function to get the cardinal directions solely from the JSON response. In our enclosed environment, when the vehicle is heading west and then makes a left turn, it ends up facing south. Therefore, we will focus exclusively on roads that follow this configuration. As we delve deeper into the project, we believe it's crucial to integrate a GPS system capable of determining the vehicle's cardinal direction. The code is available in the Appendix.

After that, I extracted the start_location in Figure 5.4.1.1 and the cardinal directions and stored them in a dictionary ('junctions') to be used when calling Google Street View API.

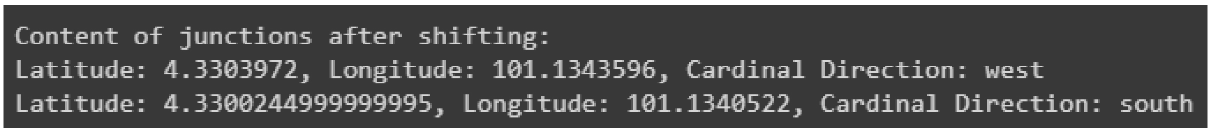


```
Content of junctions:
Latitude: 4.3303972, Longitude: 101.1341896, Cardinal Direction: west
Latitude: 4.3298545, Longitude: 101.1340522, Cardinal Direction: south
```

Figure 5.4.1.2 Content in Junction

5.4.2 Google Street View API

I performed coordinates shifting on the coordinates to get the desired junction Figure 5.4.1.2 and the figure below shows the shifted coordinates.



```
Content of junctions after shifting:
Latitude: 4.3303972, Longitude: 101.1343596, Cardinal Direction: west
Latitude: 4.3300244999999995, Longitude: 101.1340522, Cardinal Direction: south
```

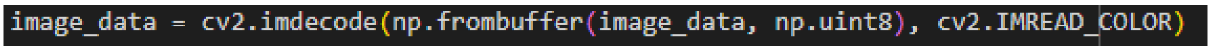
Figure 5.4.2.1 Content in Junction (after shifting)

Parameters passed into Google Street View API:

Table 5.4.2.1 Parameters Passed to Google Street View API

Parameters	Value
Latitude	(Latitude from junctions)
longitude	(Longitude from junctions)
size	'640x480'
fov	100
heading	(Cardinal direction from junctions)
pitch	0
API key	***

I decoded the response from the GSV API into a format that OpenCV (cv2) can work with using the code snippet in Figure 5.4.2.2. The flag “cv2.IMREAD_COLOR” indicates that the image should be loaded in color mode, preserving all color channels (Blue, Green, Red).



```python
image_data = cv2.imdecode(np.frombuffer(image_data, np.uint8), cv2.IMREAD_COLOR)
```

Figure 5.4.2.2 Code snippet to decode GSV API response

The images are appended to a list “junction_images” as shown in Figure 5.4.2.3

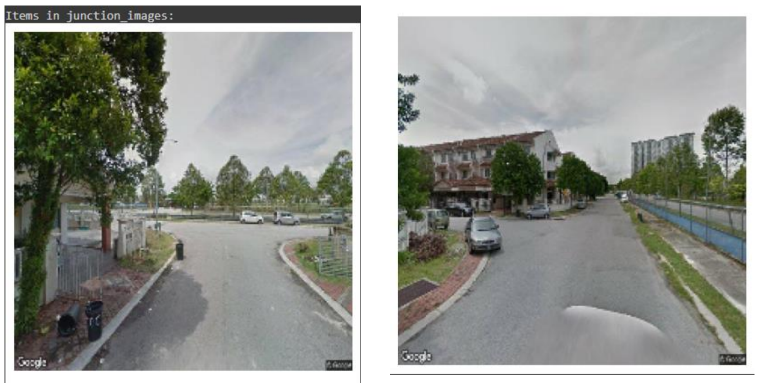


Figure 5.4.2.3 images in junction_images

5.4.3 Image Similarity Detection Model

I compared captured junction and the correct junction in “junction_images” and the results are further discussed in Chapter 6: System Evaluation and Discussion.

5.5 Concluding Remarks

In this section, we illustrated a comprehensive test case demonstrating the successful amalgamation of the Google Directions API and Google Street View API. Through precise parameter configuration, we efficiently acquired essential route information, junction coordinates, and cardinal directions. Notably, our custom "get_new_heading" function facilitated the extraction of cardinal directions from the API response. Following this, we executed coordinate shifting to optimize junction framing, and the resulting street view images were adeptly decoded for seamless integration into our system. This harmonious integration lays the groundwork for our intersection detection system, setting the stage for rigorous evaluation and refinement in the subsequent chapters.

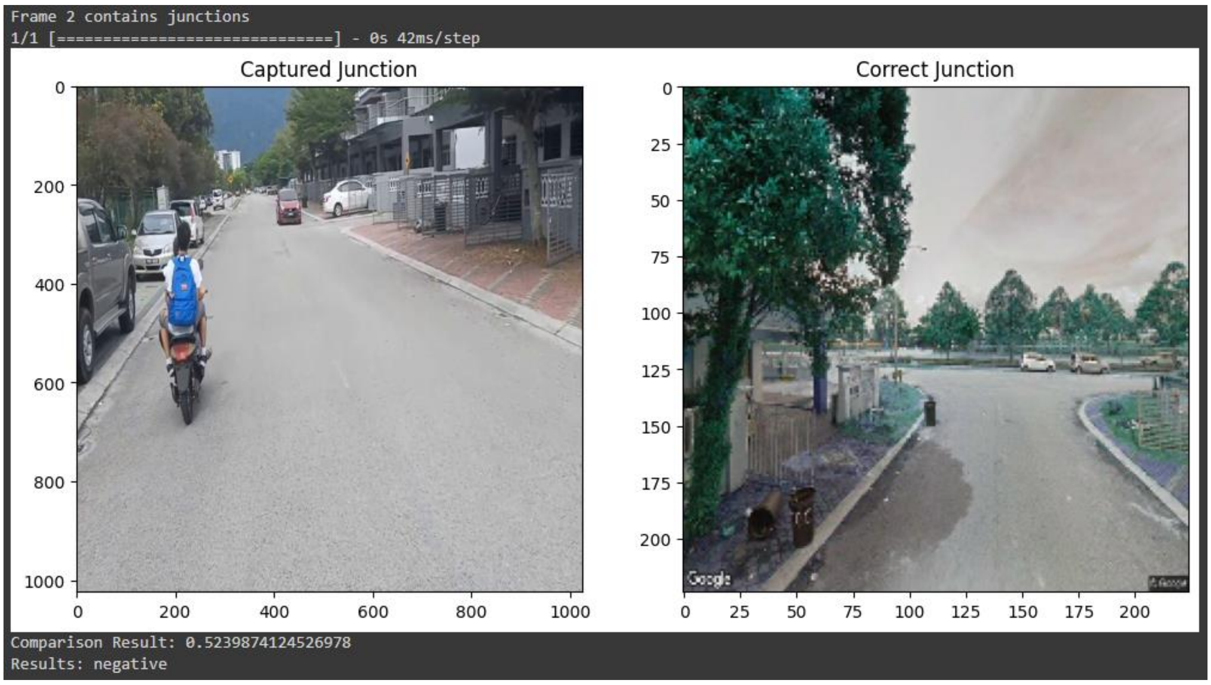
Chapter 6

System Evaluation and Discussion

6.1 Testing Setup and Result

In the evaluation of our junction recognition system, we concentrated our efforts on a specific test case within the Westlake area, where the system demonstrated results.

Figure 6.1 display two junctions that do not match. Under this situation, the result is negative.

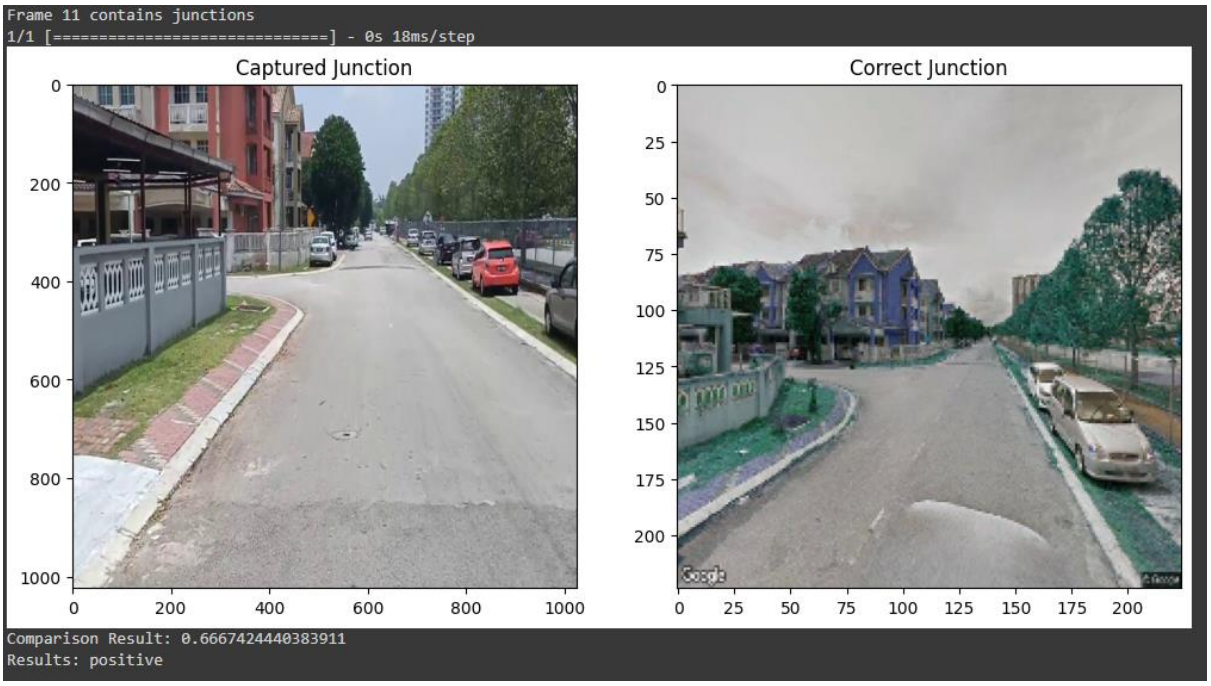


```
Frame 2 contains junctions
1/1 [==============================] - 0s 42ms/step
```

```
Comparison Result: 0.5239874124526978
Results: negative
```

Figure 6.1 Junction that do not match

Figure 6.2 display two images that matches. Under this condition, the results will be positive.

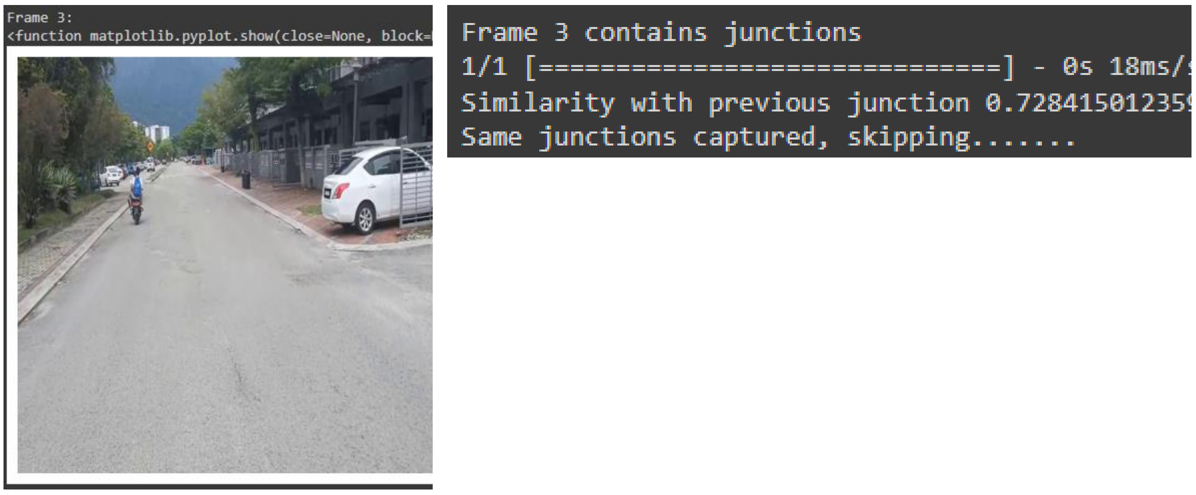


```
Frame 11 contains junctions
1/1 [==============================] - 0s 18ms/step
```

```
Comparison Result: 0.6667424440383911
Results: positive
```

Figure 6.2 Junction that matches

Figure 5.4.3.3 display an image that depicts the same junction as the previous frame. Consequently, it will be skipped and not compared with any items in “junction_images”



```
Frame 3:
<function matplotlib.pyplot.show(close=None, block=
```

```
Frame 3 contains junctions
1/1 [==============================] - 0s 18ms/s
Similarity with previous junction 0.72841501235
Same junctions captured, skipping.......
```

Figure 6.3 Duplicate Image of Junction

6.2 Project Challenges

In the implementation of our intersection detection system, we encountered several noteworthy challenges related to the use of Google Street View (GSV) imagery. One of the primary challenges arose from the fact that GSV imagery may capture junctions from different angles compared to our test cases. This variance in perspective posed difficulties for our similarity detection model, as it needed to match frames taken at potentially contrasting orientations. Consequently, achieving accurate similarity comparisons between our test cases and GSV imagery became a significant challenge.

Furthermore, we grappled with the issue of temporal disparities in GSV imagery. Over time, buildings, road infrastructure, and their surroundings can undergo substantial transformations. This temporal shift meant that GSV images often depicted junctions with different colors, structures, and environmental contexts compared to our current test cases. These alterations sometimes led to a noticeable discrepancy between the present-day scene and the older GSV imagery. Navigating these temporal differences and ensuring reliable similarity matching in such dynamic scenarios posed an intricate challenge that required innovative solutions.

6.3 Objectives Evaluation

Our approach successfully automated the process of route generation, coordination extraction, and junction detection. However, it's essential to acknowledge that while the outcomes were generally satisfactory, there exists a scope for improvement. Notably, we faced two significant challenges as discussed in Chapter 5.5: Implementation Issues & Challenges. The temporal differences between Google Street View (GSV) imagery and real-time scenarios can affect the system's accuracy, given that our model relies on GSV images as references. Furthermore, variations in image angles can result in deviations from the intended viewpoint, potentially impacting detection accuracy.

In light of these challenges, we adopted a conservative threshold of 0.65 for similarity scores in our similarity detection model. This threshold selection was a deliberate choice to mitigate the impact of these challenges on our results. Moving forward, we recognize the potential for further refining our approach and expanding its adaptability to diverse scenarios while maintaining a strong foundation built on the lessons learned from this specific test case. Our ongoing commitment is to enhance the system's robustness, making it even more effective and versatile.

6.4 The Feasibility of Using Google Street View API for Junction Image Capture

The utilization of the Google Street View (GSV) API as a tool for capturing images of junctions along predefined routes offers numerous advantages in the realm of intersection detection. It provides a readily available source of visual data that can be harnessed for real-time navigation and intersection recognition. However, the effectiveness of this approach is subject to certain limitations and considerations.

One primary consideration is the temporal aspect of GSV imagery. GSV provides a vast repository of street-level images, but the temporal gap between when these images were captured and the present moment can pose challenges. Street views may have changed over time due to construction, urban development, or simply the evolution of natural surroundings. Consequently, there is a possibility that the GSV images may not accurately represent the current state of junctions. This temporal misalignment introduces an element of uncertainty into the system, as the junctions depicted in the GSV imagery may not perfectly align with the physical junctions encountered by drivers.

Another crucial aspect is the variability in image angles and viewpoints captured by GSV. Street view imagery is collected using multiple cameras mounted on vehicles that traverse roadways, resulting in images captured from different angles and positions. While this diversity is valuable for general navigation purposes, it can become a challenge when specific viewpoints are required for intersection detection. Varied angles can lead to deviations from the intended perspective, potentially affecting the system's accuracy in recognizing junctions.

Despite these challenges, the GSV API remains a valuable tool for capturing images of junctions along routes. Its vast coverage and accessibility make it an attractive choice for automating intersection detection. To enhance its utility further, a potential avenue for improvement would be to ensure more frequent updates of GSV imagery. The integration of more recent images into the GSV database would enable a closer alignment between the captured imagery and real-time junctions, reducing the temporal discrepancies. Additionally, implementing techniques for selecting images with optimal viewpoints and angles could enhance the system's accuracy in identifying junctions.

In conclusion, while Google Street View API serves as a promising tool for capturing purposeful images of junctions along routes, it is essential to address the temporal disparities and angle variations inherent in the imagery. Overcoming these challenges through regular updates and viewpoint optimization could significantly enhance the system's effectiveness in intersection detection, further solidifying its role as a valuable resource for navigation and route guidance.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

I believe that my project represents a significant contribution to the field of intersection detection. By devising a novel approach that combines the Google Directions API and the Google Street View API, I aimed to streamline the process of obtaining route information while harnessing visual data critical for intersection identification. Through this implementation, I have demonstrated the potential for more efficient and user-friendly navigation systems.

One of the key strengths of my work lies in the automation of the entire process, encompassing route calculation and image comparison. This automation has the potential to reduce manual intervention, which is a substantial step forward in enhancing navigation systems. Additionally, my project addresses real-world challenges such as variations in image angles and temporal differences in Google Street View imagery, showcasing my commitment to overcoming practical obstacles.

Moreover, my project underscores the importance of adopting a multidisciplinary approach, bringing together geographic data, computer vision, and machine learning to tackle intersection recognition challenges. This interdisciplinary perspective not only adds depth to my work but also creates opportunities for future research and innovation in the realm of navigation and road safety.

In summary, I firmly believe that my project's unique blend of APIs, automation, and interdisciplinary thinking holds great promise for the improvement of intersection detection and navigation systems. While there is undoubtedly room for improvement, my work serves as a solid foundation upon which future developments can build, ultimately contributing to safer and more efficient road travel.

7.2 Recommendation

Firstly, considering the importance of user interaction and accessibility, we recommend the incorporation of a user-friendly interface that allows users to input their origin and destination coordinates conveniently. This feature would empower users to define their specific routes, enabling the system to fetch accurate directions and images tailored to their preferences. Such a user interface would promote greater engagement and customization, catering to a broader range of user requirements.

Furthermore, to augment the system's capabilities and user convenience, integration with widely used mapping services such as Google Maps could be considered. By seamlessly integrating with Google Maps, our system could leverage its extensive features, real-time traffic updates, and user-friendly interface. Users could effortlessly plan their routes, and our intersection detection system could seamlessly fetch the necessary data, providing a comprehensive and efficient navigation experience.

REFERENCES

[1] Google, Google Maps platform documentation | Street View Static API | google for developers, https://developers.google.com/maps/documentation/streetview/ (accessed Sep. 12, 2023).

[2] 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).

[3] A. G. Rundle, M. D. M. Bader, C. A. Richards, J. O. Teitler, and K. M. Neckerman, "Using google street view to audit neighborhood environments," American Journal of Preventive Medicine, https://www.sciencedirect.com/science/article/pii/S0749379710005623 (accessed Sep. 12, 2023).

[4] J. Rousselet et al., "Assessing species distribution using google street view: A pilot study with the pine processionary moth," PLOS ONE, https://journals.plos.org/plosone/article?id=10.1371%2Fjournal.pone.0074918 (accessed Sep. 12, 2023).

[5] S. F. Memon, M. A. Memon, S. Zardari, and S. Nizamani, "Blind's eye: Employing Google Directions API for outdoor navigation of visually impaired pedestrians," Mehran University Research Journal of Engineering and Technology, https://publications.muet.edu.pk/index.php/muetrj/article/view/282 (accessed Sep. 13, 2023).

[6] Google, "The Google Directions API - Google Maps API Web Services," Google Maps platform documentation | directions API | google for developers, https://developers.google.com/maps/documentation/directions/ (accessed Sep. 13, 2023).

[7] Q. T. Le and D. Pishva, "Application of web scraping and google API service to ... - IEEE xplore," 2015 17th International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea (South), https://ieeexplore.ieee.org/abstract/document/7224841/ (accessed Sep. 13, 2023).

[8] H. Jin, F. Jin, Q. Hao, H. Zhu, and X. Yang, "Measuring public transit accessibility based on Google Direction Api," The Open Transportation Journal, https://opentransportationjournal.com/VOLUME/13/PAGE/93/FULLTEXT/ (accessed Sep. 13, 2023).

[9] V. Balali, E. Depwe, and M. Golparvar-Fard, "Multi-class traffic sign detection and classification using Google ...," researchgate, https://www.researchgate.net/profile/Vahid_Balali/publication/271273346_Multi-class_Traffic_Sign_Detection_and_Classification_Using_Google_Street_View_Images/links/55f708f808ae07629dbcbfa1.pdf (accessed Sep. 13, 2023).

[10] R. S. Mamidala, U. Uthkota, M. B. Shankar, A. J. Antony, and A. V. Narasimhadhan, "IEEE Xplore Full-text PDF:," IEEEXplore, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7169508 (accessed Sep. 13, 2023).

[11] Y. Jiang, F. Gao, and G. Xu, "Computer Vision-based multiple-lane detection on straight ... - IEEE xplore," IEEE Xplore, https://ieeexplore.ieee.org/document/5476151/ (accessed Sep. 13, 2023).

[12] A. Buczkowski, "Why would you use openstreetmap if there is google maps?," Geoawesomeness, https://geoawesomeness.com/why-would-you-use-openstreetmap-if-there-is-google-maps/ (accessed Sep. 13, 2023).

[13] "Angular symbols for standard solar relations," Angular Symbols for Standard Solar Relations | EME 810: Solar Resource Assessment and Economics, https://www.e-education.psu.edu/eme810/node/575 (accessed Sep. 14, 2023).

Appendix

A.1 api.ipynb

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

import os

import scipy.misc
import numpy as np
from six import BytesIO
from PIL import Image, ImageDraw, ImageFont

import tensorflow as tf


import pandas as pd
import numpy as np
import keras
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import h5py
import cv2
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
import tensorflow as tf
from scipy import spatial
#from tensorflow.keras.applications.vgg16 import VGG16


%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=0.45,
        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()

loaded_module = tf.saved_model.load('/content/gdrive/MyDrive/FYP2/current_best')
loaded_detection_fn = loaded_module.detect
loaded_fine_tuned_model = loaded_module.detection_model


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

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

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

def get_feature_vector(img):
 img = cv2.resize(img, (224, 224))
 feature_vector = basemodel.predict(img.reshape(1, 224, 224, 3))
 return feature_vector

def calculate_similarity(vector1, vector2):
    return 1 - spatial.distance.cosine(vector1, vector2)

    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': ‘*’,
        '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}")

    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


        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

        import requests
        # Function to fetch Google Street View image
        def fetch_street_view_image(latitude, longitude, heading):
          api_key = '*'
          size = '640x480'
          fov = 100
          heading = heading
          pitch = 0
          url =
    f'https://maps.googleapis.com/maps/api/streetview?location={latitude},{longitude}&size={si
    ze}&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')

        import cv2
        import numpy as np
        import matplotlib.pyplot as plt
        # 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)

        frames_dir = '/content/gdrive/MyDrive/FYP2/1fps'
        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))

        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("Similarity with previous junction", similarity_score)
                    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])

                # Display the images
                fig, axes = plt.subplots(1, 2, figsize=(12, 6))  # Create a subplot with two columns

                # Display image_np on the first subplot
                axes[0].imshow(image_np)
                axes[0].set_title('Captured Junction')

                # Display junction_images[junction_image_counter] on the second subplot
                axes[1].imshow(junction_images[junction_image_counter])
                axes[1].set_title('Correct Junction')

                plt.show()
```

```
    # 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("Results: positive")
      #trigger_raspi_audio(positive)
    else:
      print("Results: negative")
      #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...........')

print("\n\n")
```

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	**Study week no.:2**
Student Name & ID: Tan Yi Xuan , 2101124	
Supervisor: Dr. Aun Yichiet	
Project Title: REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING	

1. WORK DONE Pick up previous work done in FYP1 and discuss with supervisor.
2. WORK TO BE DONE Create Google developer account and get API key to be used in API calls in code.
3. PROBLEMS ENCOUNTERED - no
4. SELF EVALUATION OF THE PROGRESS - On track

[illegible]　　　　[illegible]

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 Yi Xuan , 2101124	
Supervisor: Dr. Aun Yichiet	
Project Title: REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING	

1. WORK DONE - generated API key - explored methods to use Google Street View API calls in python
2. WORK TO BE DONE - Get data for test case
3. PROBLEMS ENCOUNTERED - no
4. SELF EVALUATION OF THE PROGRESS - On track

[illegible]

[illegible]

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 Yi Xuan , 2101124	
Supervisor: Dr. Aun Yichiet	
Project Title: REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING	

1. WORK DONE - obtained videos that simulate cars dashcam when driving on the road within our testcase
2. WORK TO BE DONE - re-evaluate project flow
3. PROBLEMS ENCOUNTERED - no
4. SELF EVALUATION OF THE PROGRESS - On track

[illegible] [illegible]

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 Yi Xuan , 2101124	
Supervisor: Dr. Aun Yichiet	
Project Title: REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING	

1. WORK DONE - explored google directions api and implemented in code
2. WORK TO BE DONE - discuss with groupmate on integration
3. PROBLEMS ENCOUNTERED - no
4. SELF EVALUATION OF THE PROGRESS - On track

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	**Study week no.:10**
Student Name & ID: Tan Yi Xuan , 2101124	
Supervisor: Dr. Aun Yichiet	
Project Title: REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING	

1. WORK DONE - apply logic of integration with groupmate in code
2. WORK TO BE DONE - finalize code
3. PROBLEMS ENCOUNTERED - no
4. SELF EVALUATION OF THE PROGRESS - On track

[illegible]

[illegible]

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 Yi Xuan , 2101124	
Supervisor: Dr. Aun Yichiet	
Project Title: REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING	

1. WORK DONE - clean up code and write report
2. WORK TO BE DONE -
3. PROBLEMS ENCOUNTERED - no
4. SELF EVALUATION OF THE PROGRESS - On track

[illegible] [illegible]

Supervisor's signature   Student's signature

POSTER

REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING

Faculty of Information and Technology

UTAR UNIVERSITI TUNKU ABDUL RAHMAN

INTRODUCTION

The research aims to enhance traffic safety by developing a computer vision model that employs image matching to identify the correct intersection, addressing the challenge of helping divers navigate unfamiliar roadways more effectively.

OBJECTIVES

To develop and evaluate a computer vision model that utilise image comparison techniques to accurately recognise the appropriate turn for the drivers utilising Google APIs

METHODOLOGY

Streamlining the process of obtaining route information using various APIs

Harnessing visual data critical for intersection recognition

CONTRIBUTIONS

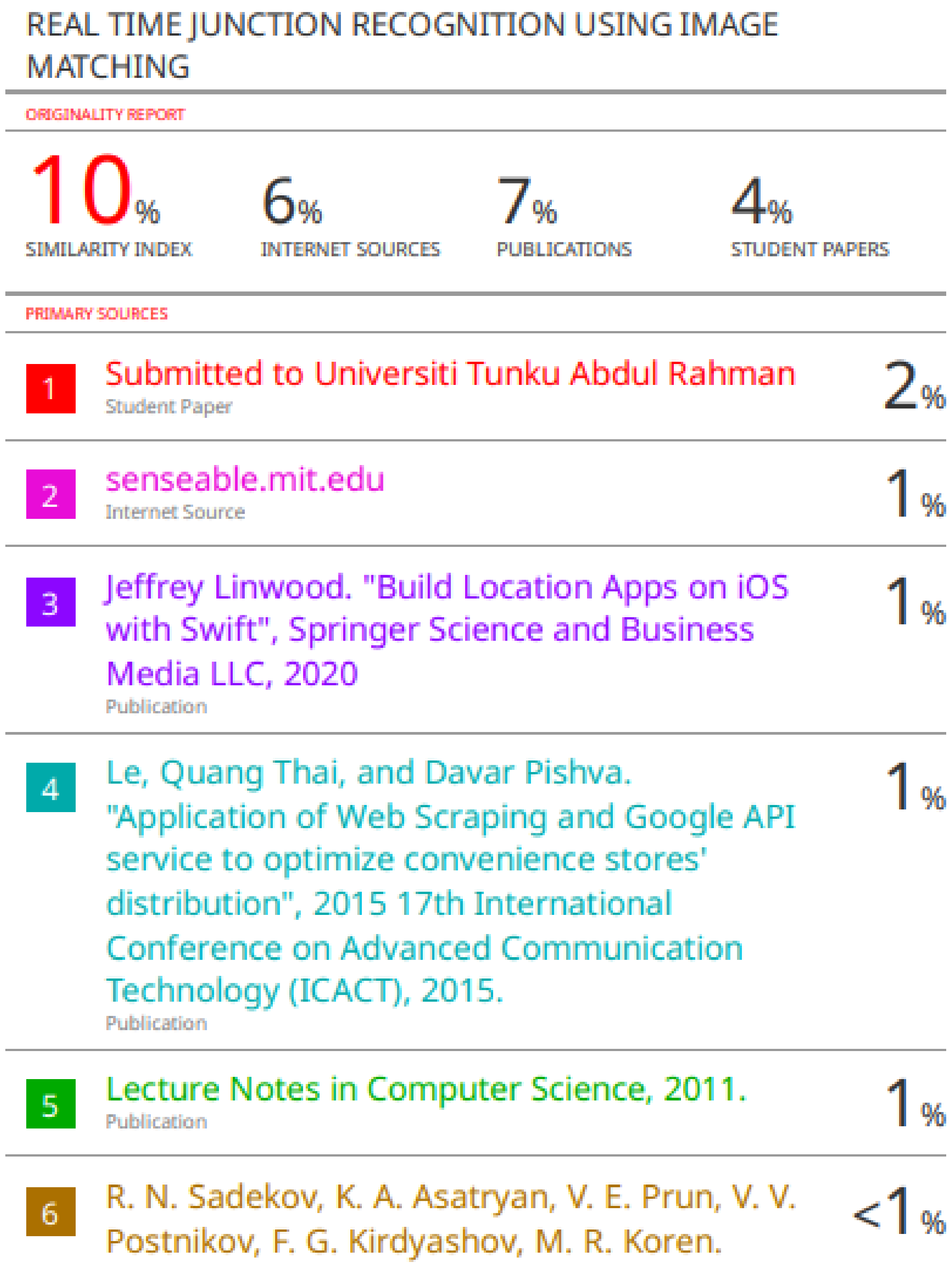
A novel approach that integrate Google Direction API and Google Street View API to provide a holistic and real-time view of the driver's journey

CONCLUSION

This research propose integration of Google Direction API and Google Street View API to simplify the retrieval of route details while utilising crucial visual data for identifying intersections

Project Developer: Tan Yi Xuan
Project Supervisor: Dr Aun Yi Chiet

PLAGIARISM CHECK RESULT



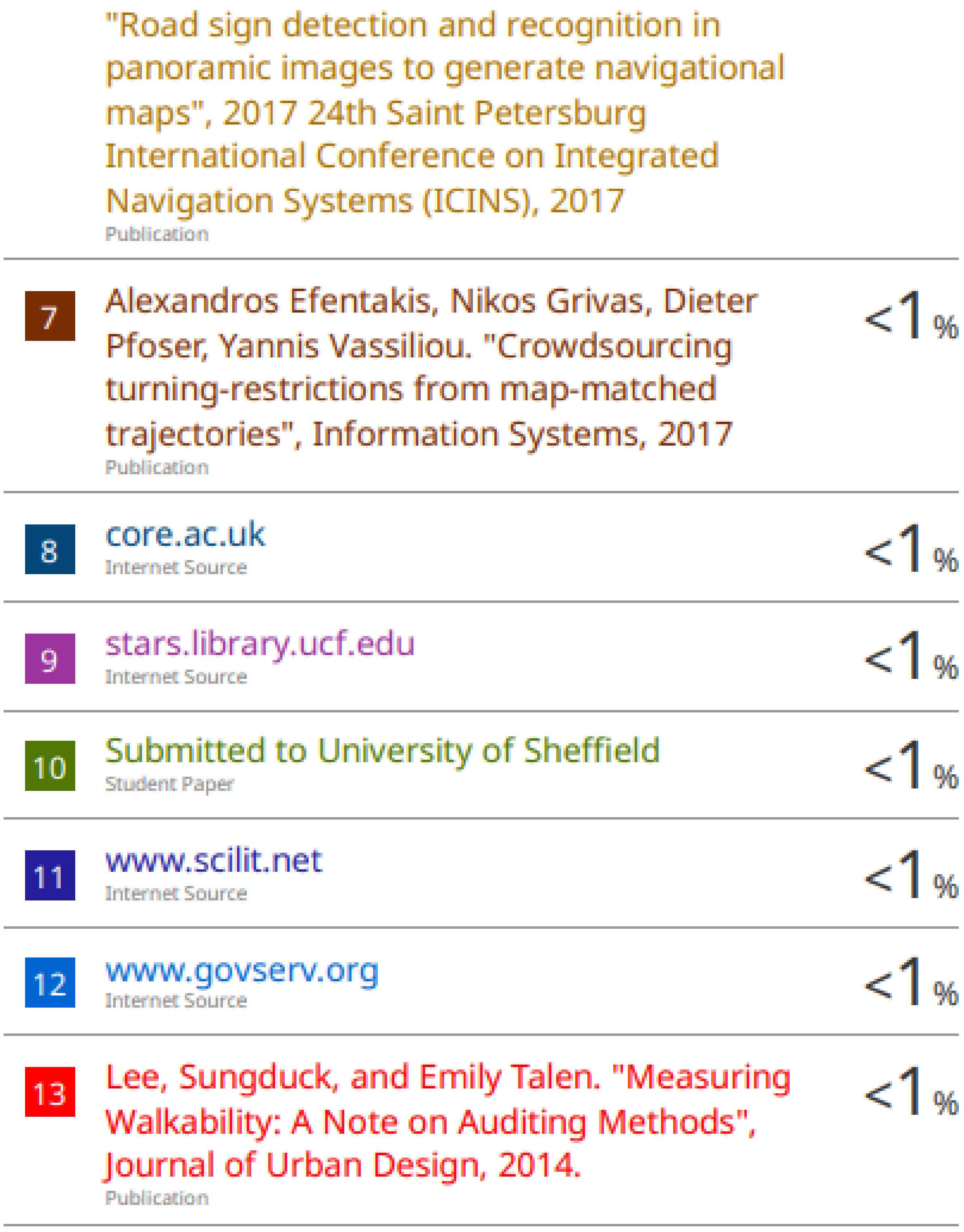
REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING

ORIGINALITY REPORT

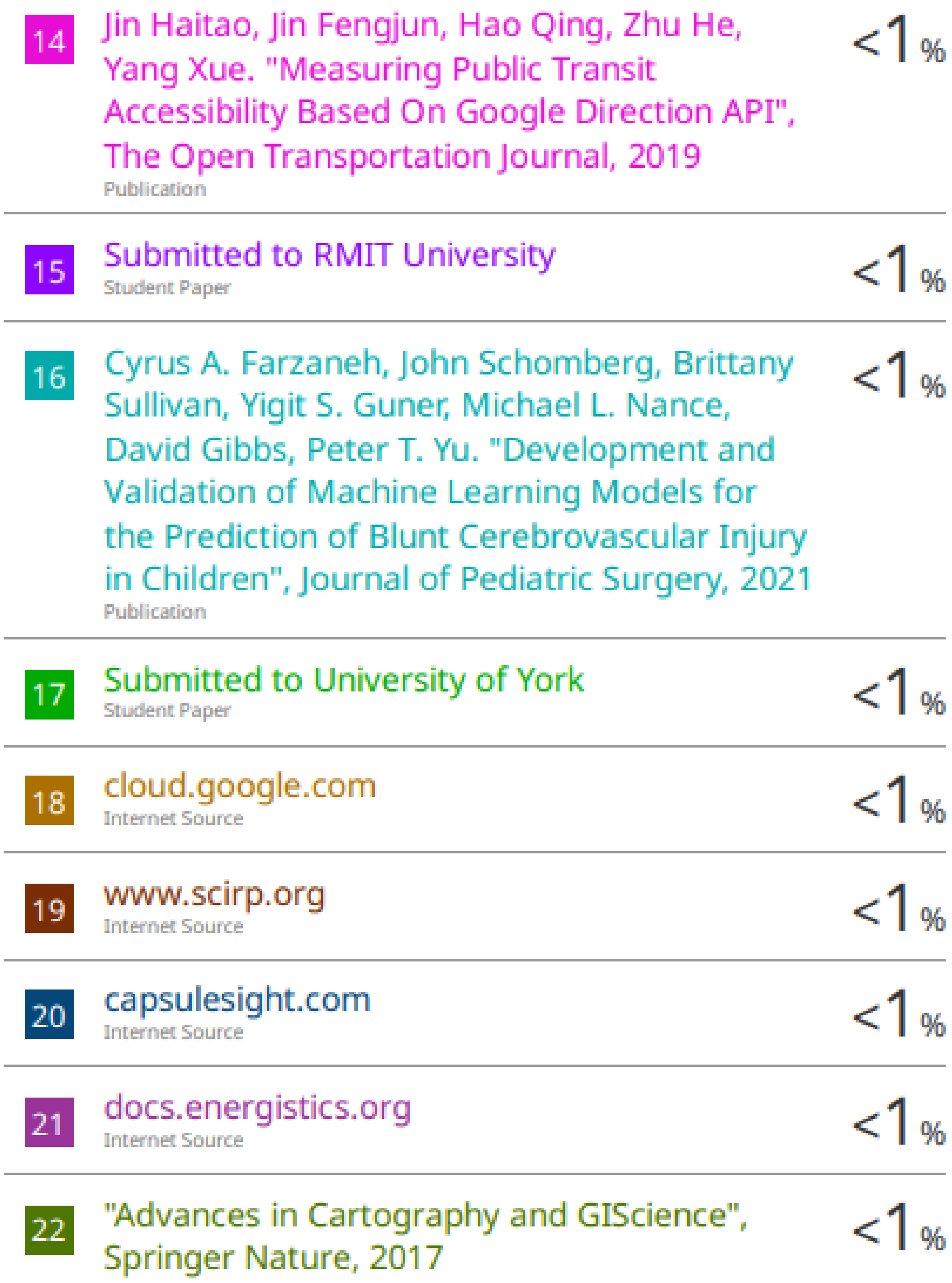
10%	6%	7%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

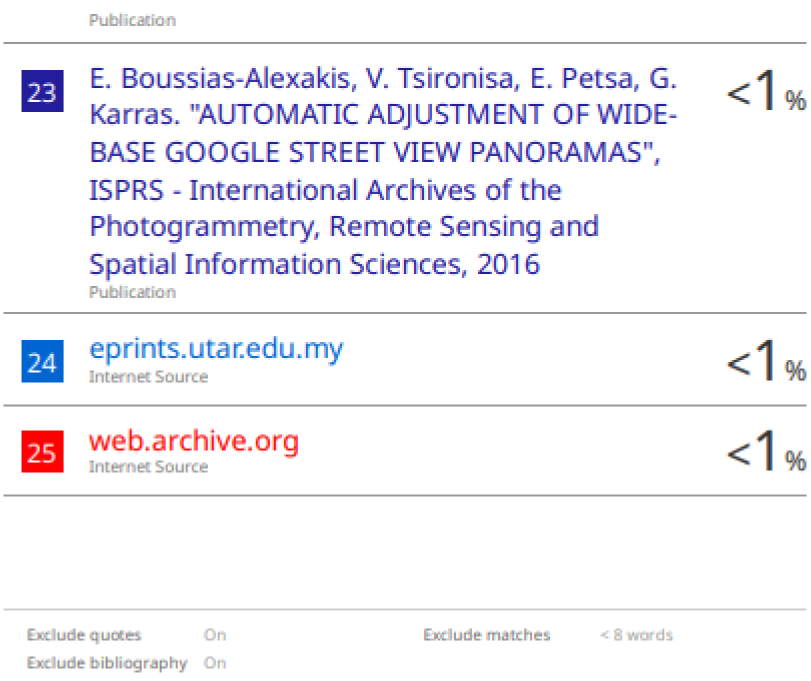
1	Submitted to Universiti Tunku Abdul Rahman Student Paper	2%
2	senseable.mit.edu Internet Source	1%
3	Jeffrey Linwood. "Build Location Apps on iOS with Swift", Springer Science and Business Media LLC, 2020 Publication	1%
4	Le, Quang Thai, and Davar Pishva. "Application of Web Scraping and Google API service to optimize convenience stores' distribution", 2015 17th International Conference on Advanced Communication Technology (ICACT), 2015. Publication	1%
5	Lecture Notes in Computer Science, 2011. Publication	1%
6	R. N. Sadekov, K. A. Asatryan, V. E. Prun, V. V. Postnikov, F. G. Kirdyashov, M. R. Koren.	<1%



	"Road sign detection and recognition in panoramic images to generate navigational maps", 2017 24th Saint Petersburg International Conference on Integrated Navigation Systems (ICINS), 2017 Publication	
7	Alexandros Efentakis, Nikos Grivas, Dieter Pfoser, Yannis Vassiliou. "Crowdsourcing turning-restrictions from map-matched trajectories", Information Systems, 2017 Publication	<1%
8	core.ac.uk Internet Source	<1%
9	stars.library.ucf.edu Internet Source	<1%
10	Submitted to University of Sheffield Student Paper	<1%
11	www.scilit.net Internet Source	<1%
12	www.govserv.org Internet Source	<1%
13	Lee, Sungduck, and Emily Talen. "Measuring Walkability: A Note on Auditing Methods", Journal of Urban Design, 2014. Publication	<1%



14	Jin Haitao, Jin Fengjun, Hao Qing, Zhu He, Yang Xue. "Measuring Public Transit Accessibility Based On Google Direction API", The Open Transportation Journal, 2019 Publication	<1 %
15	Submitted to RMIT University Student Paper	<1 %
16	Cyrus A. Farzaneh, John Schomberg, Brittany Sullivan, Yigit S. Guner, Michael L. Nance, David Gibbs, Peter T. Yu. "Development and Validation of Machine Learning Models for the Prediction of Blunt Cerebrovascular Injury in Children", Journal of Pediatric Surgery, 2021 Publication	<1 %
17	Submitted to University of York Student Paper	<1 %
18	cloud.google.com Internet Source	<1 %
19	www.scirp.org Internet Source	<1 %
20	capsulesight.com Internet Source	<1 %
21	docs.energistics.org Internet Source	<1 %
22	"Advances in Cartography and GIScience", Springer Nature, 2017	<1 %



Publication

23	E. Boussias-Alexakis, V. Tsironisa, E. Petsa, G. Karras. "AUTOMATIC ADJUSTMENT OF WIDE-BASE GOOGLE STREET VIEW PANORAMAS", ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2016 Publication	<1 %
24	eprints.utar.edu.my Internet Source	<1 %
25	web.archive.org Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		

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.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	TAN YI XUAN
ID Number(s)	21ACB01124
Programme / Course	BACHELOR OF COMPUTER SCIENCE (HONOURS)
Title of Final Year Project	REAL TIME JUNCTION RECOGNITION USING IMAGE MATCHING

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: __10__ **%** **Similarity by source** Internet Sources: __6__% Publications: __7__ % Student Papers: __4__ %	
Number of individual sources listed of more than 3% similarity: 0____	
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** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

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.

[illegible]

Signature of Supervisor	Signature of Co-Supervisor
Name: _Dr. Aun Yichiet____	Name: ________
Date: ___15 Sep 2023____	Date: ________

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	21ACB01124
Student Name	TAN YI XUAN
Supervisor Name	DR. AUN YICHIET

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.

[illegible]

____

(Signature of Student)
Date:12/9/2023