Faculty of Engineering and Applied Science

SOFE 4630U Cloud Computing

QUANTIFYING CRITICAL SCENARIOS FOR AUTOMATED DRIVING SYSTEMS

R2: Project Implementation

Group 06 Members

Ashley Tyrone	100786450
Darren Constantine	100782701
Laksumi Somaskanthamoorthy	100782723
Lauryne Zachary	100797920
Ram Baldeo	100701013

Course Professor: Dr. M. El-Darieby

QUANTIFYING CRITICAL SCENARIOS FOR AUTOMATED DRIVING SYSTEMS

Ashley Tyrone, Darren Constantine, Laksumi Somaskanthamoorthy, Lauryne Zachary, Ram Baldeo

Github Repository: https://github.com/Darren-C26/inD-trajectory-classification

Executive Summary

The focus of this project is to develop a criticality metric in order to provide safety validation for an Autonomous Drive System (ADS) during scenario-based testing. Processing a large dataset in a fast and efficient manner is required and a crucial aspect when considering safety validation. To compute the criticality metric, the inD dataset will be utilized along with multiple technologies, including Google Pub/Sub, Apache Beam, and Dataflow. Functional scenarios are to be extracted from the dataset and the appropriate parameter trajectories will be utilized for criticality assessment.

1. Introduction

With the rise of autonomous vehicles in recent years comes rapid developments made very frequently, both in industry and on a provincial scale – in 2016, Ontario had announced it invested $2.95 million for a 10-year pilot project surrounding self-driving vehicles [1]. However, this is only the surface as more and more initiatives are emerging from industries that are seeing the potential in autonomous vehicles, resulting in a stark increase in the research, design, and development in this area of interest. Autonomous vehicles, also known as "AVs", refer to vehicles that use different components such as sensors, actuators, as well as various rules, models, and algorithms, to sense changes in their surrounding environment and operate without a driver in the vehicle [2]. At their core, AVs strive to perform the operational functionalities of a traditional car, while not requiring a human to be controlling the vehicle by intricately navigating through their environment autonomously.

To determine the level of autonomy these vehicles have, there are 6 levels of automation present: Level 0 refers to no automation being present at all, while Level 5 indicates an AV as fully autonomous [2]. An important aspect of AVs stems from the collection of data from various sensors that is processed and analyzed so that the vehicle can understand its environment and predict what actions it must take based on the changes it sees. As the data processed is often in large volumes and arriving at high velocities, the need for an architecture, platform, and infrastructure that supports this data is vital to the overall process. For this reason, a cloud computing approach becomes especially important to ensure that the data is stored and processed at the rate it arrives and allows the vehicle to access the data it requires to function in a safe and controlled manner.

Safety validation is integral to ensuring that an AV behaves safely when it senses an event, object, or overall change in its environment. In the situation it senses an object it has not seen before, performing an unsafe action can result in dire consequences and a loss of life, in extreme cases. To reduce such risks, it becomes necessary to identify and compute the possible trajectories it can take by collecting and processing the obtained data; this information can be used by the AV to make safe decisions when faced with such situations [3]. Through the use of trajectory computation, safety validation vastly improves the performance and makes certain that AVs make use of the data and perform safe actions when operating.

2. Objectives

The objectives for this project have been revised to further reflect our implementation. They are as follows:

- Extracting and identifying objects present in the dataset
- Implementing a pipeline through Apache Beam and Google Dataflow to process, analyze, and classify the data based on the computed turning behavior, which is the vehicle trajectory of the object
- Store the computed trajectories using Big Query, in which analytics can be extracted and visualized through Looker Studio
- Utilize Google Pub/Sub to allow for messages to be published to Pub/Sub topics based on the vehicle's trajectory, which can be received by their respective subscriptions

3. Project Overview

The proposed project will utilize the paper "Criticality Metric for the Safety Validation of Automated Driving using Model Predictive Trajectory Optimization" [3] to implement a cloud native solution through the use of concepts and tools learned in the course. The data retrieved by AVs and their sensors display the core characteristics of "Big Data" as they can be quite large in volume, arrive at high velocities, and are in different varieties, from structured to unstructured data [4]. However, traditional architectures may not be able to support the needs of this data which makes a cloud native architecture vital to such applications. These architectures have the flexibility, scalability, and availability required of them to perform processing and analysis in close to real-time.

To perform safety validation, the classification of various vehicle trajectories will be implemented. The dataset to be used is the inD dataset, which involves data from drones that have been collected over German intersections [5]. This data will be stored in buckets, and reformatted to a form where each track record is processed, which will be used by a function to compute the turning behavior of a vehicle, based on the average heading difference and average of the centroids. This is then used in an overall Apache Beam pipeline that involves processing the data, performing classification of the vehicle's behavior, and then both storing this data in Big Query to be visualized by Looker Studio, as well as publishing the data into Google Pub/Sub topics depending on the vehicle's trajectory.

4. Dataset Specification

The Intersection Drone (inD) dataset by leveLXData involves a collection of data surrounding vehicle trajectories that have been captured using drones hovering over German intersections [5]. It consists of recordings from four locations, each with different road users such as cars, trucks, as well as cyclists and pedestrians [5]. Because of this, the inD dataset is especially useful when handling scenario-based applications that involve the safety and prediction of vehicle behavior. The dataset includes data which is georeferenced with aspects such as longitude, latitude, as well as position, heading, acceleration and velocity [5]. It also includes metadata for the recordings and tracks, containing the type of road user, the class type, as well as size [5].

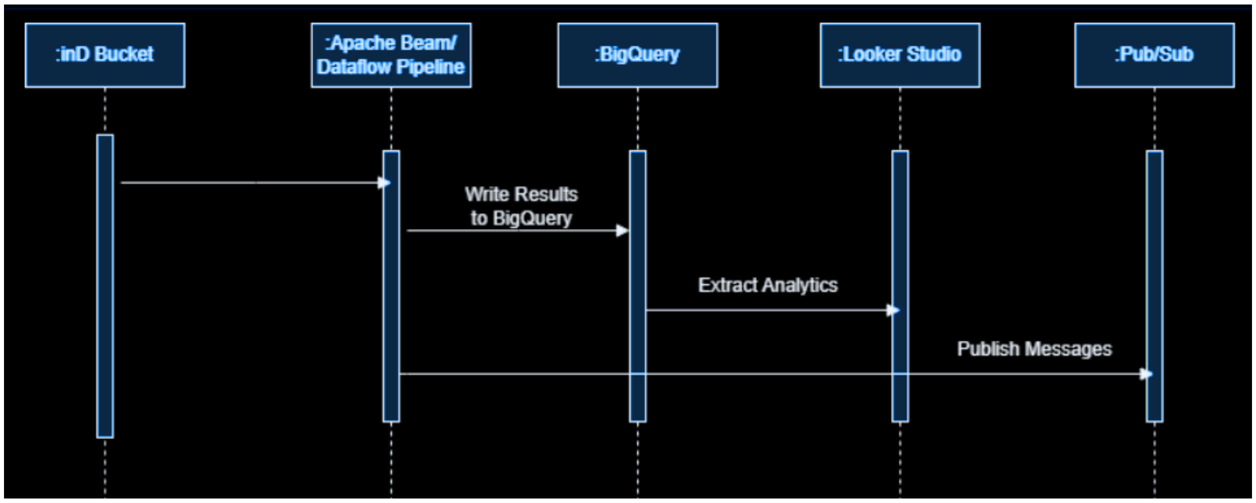
For our implementation, we will be making use of Google Cloud Storage, specifically "buckets" to store the CSV files of the track records. They are then read, parsed, grouped by a composite key that uses the same recording ID and track ID of a record, and processed in order to be used in the computation of the vehicle trajectories; this data is then reformatted into a JSON output to be used for publishing messages.

5. System Architecture

5.1 Architecture Overview

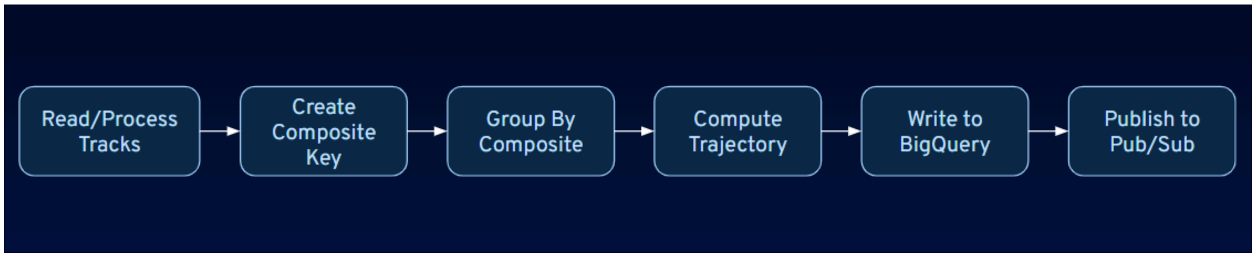
The data flow for this projects begins with CSV files that are stored in a Google Cloud Storage (GCS) bucket named '**ind_tracks**.' Apache Beam is used to process these files, extract some relevant information from them, and store the process results to BigQuery for further analysis. Additionally, the pipeline publishes messages to Pub/Sub topics based on trajectory classifications. Finally, Looker Studio is utilized to visualize and analyze data stored in BigQuery. This provides some general insights into object trajectories and traffic patterns.

A sequence diagram representing the scenario described above is shown below.



5.2 Architectural Components

The snippet below illustrates the pipeline sequence of the project.



The following provides a breakdown of the data pipeline employed in the project.

Read/Process Tracks: This component is responsible for ingesting the data from the inD dataset. The CSV files of the track records are stored in the '**ind_tracks**' bucket, which are read and parsed.

Create Composite Key: The composite key is created using the recording ID and track ID of the records; this is used in the next step of the pipeline.

Group By Composite: Records that have the same recording ID and track ID are grouped together by the composite key, which are then used to compute the trajectory.

Compute Trajectory: This is primarily performed by the '**compute_trajectory**' function. Arrays containing the centroid information and the heading rotations are used to calculate the average of centroids and average differences in heading between consecutive frames. If the average heading difference is 0, this indicates that the vehicle is not moving and is parked. In the situation that the x_center metric bears a resemblance to a linear plot, which is checked by a '**is_linear**' function, the trajectory is classified as 'straight'. If the average heading difference is less than 0, and the x_center metric of the last frame is greater than initial, the trajectory is classified as turning left and otherwise, it is classified as turning right. In cases where the average heading difference is greater than 0, the trajectory is classified as driving straight.

Write to BigQuery: The results are written to BigQuery; each message has a format that aligns with the schema in the BigQuery table. The processed records are temporarily stored in a bucket called '**track_trajectories**' while the write process occurs. Afterwards, the processed data is then sent to the Google Pub/Sub topics to be published.

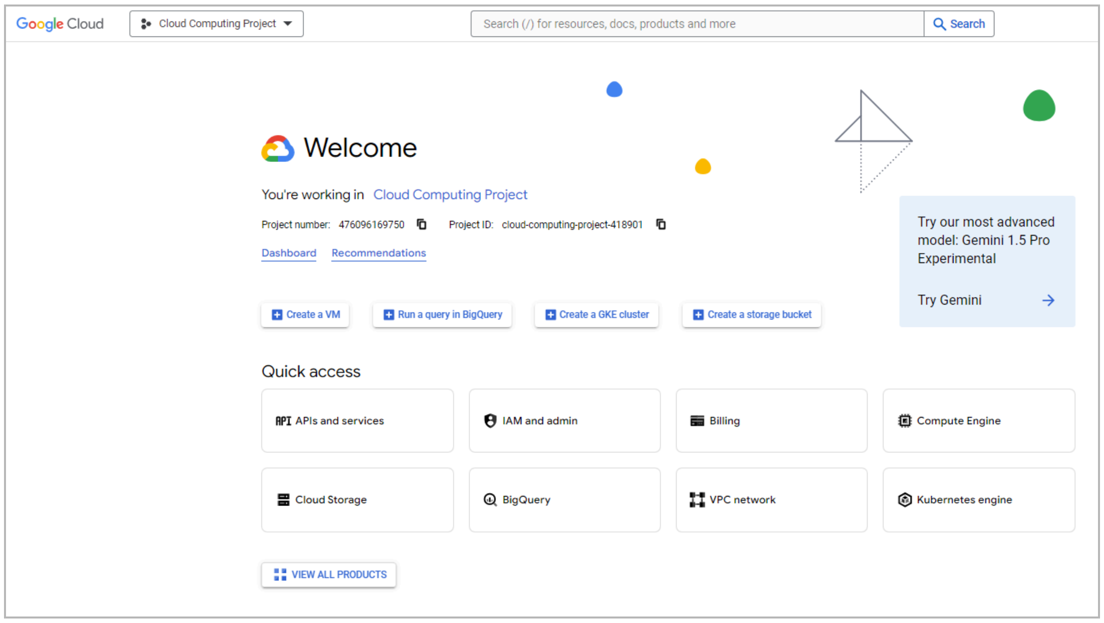
Publish to Pub/Sub: Once the trajectories are computed through the reading, parsing and processing the data, it can then be sent to Pub/Sub. On Pub/Sub the data is published on 4 different topics. Each topic is based on the trajectory paths "Left", "Straight", "Left Turn", "Right Turn". Once published, the subscribers are able to subscribe to the relevant topics.

6. Processing Techniques/Tools/Packages

This project utilizes the inD dataset in order to access the large set of vehicle data to process. A low margin of error is allowed in terms of positioning and it also accommodates data for vehicles of various sizes and makes. Python was used due to its large range in terms of available libraries, and due to the powerful functionality it provides as a programming language. It also provides feasibility when accessing scripts in the inD dataset. Apache Beam was used to implement large-scale data processing [6]. A pipeline was created to read the data in the CSV files and identify objects. Users will be able to subscribe to the various types of objects that they prefer notifications for, as they appear. Additionally, Google Cloud Platform (GCP) is used as a cloud service provider for cluster creation and to implement Pub/Sub.

7. Configuration and Deployment of GCP environment

A cloud project called *Cloud Computing Project* is created on google cloud platform with a project number and a project ID. The dataflow API is enabled for data pipeline creation enabling the data to be read, processed and written. A Pub/Sub admin role to the service account is then granted. Once all these settings are done, dataflow is now configured.


Google Cloud
Cloud Computing Project
Search (/) for resources, docs, products and more
Search

Welcome

You're working in Cloud Computing Project

Project number: 476096169750 Project ID: cloud-computing-project-418901

Dashboard Recommendations

Create a VM | Run a query in BigQuery | Create a GKE cluster | Create a storage bucket

Try our most advanced model: Gemini 1.5 Pro Experimental

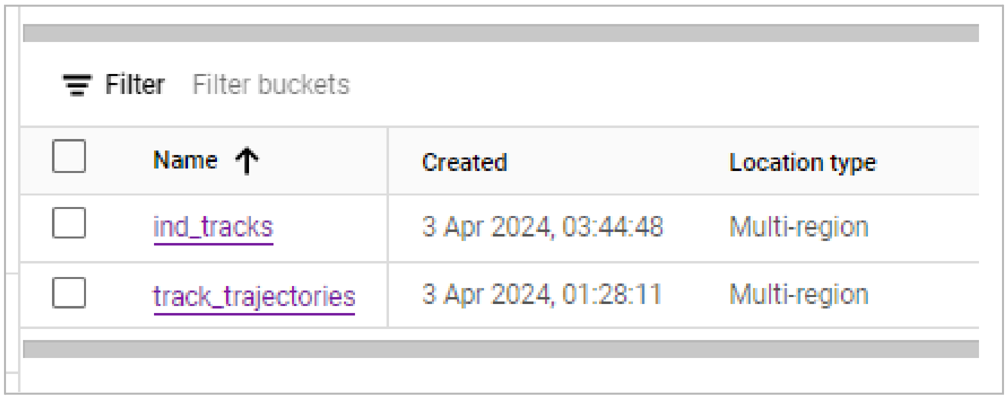
Try Gemini

Quick access

APIs and services | IAM and admin | Billing | Compute Engine
Cloud Storage | BigQuery | VPC network | Kubernetes engine

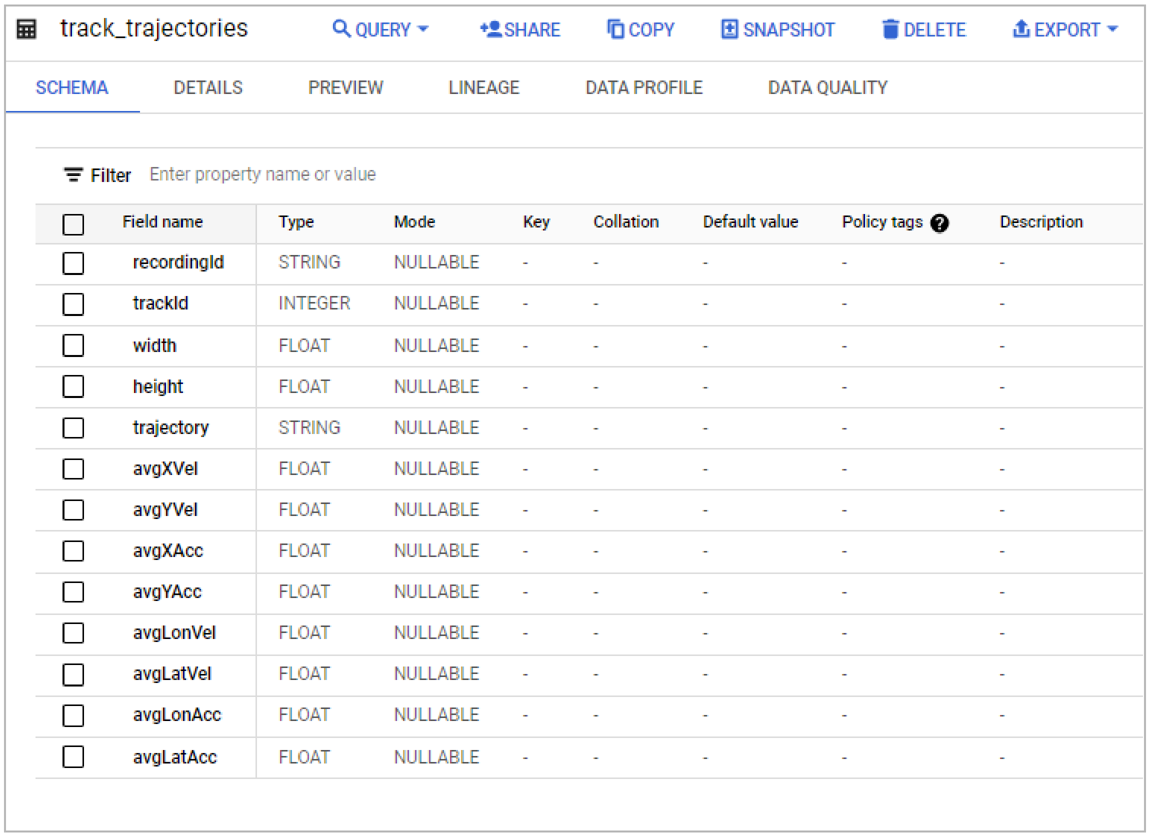
VIEW ALL PRODUCTS

Two buckets are then created. The first one is ind_tracks which is for the csv files. The second one is track_trajectories which is used for the temporary collection of results.


Filter Filter buckets

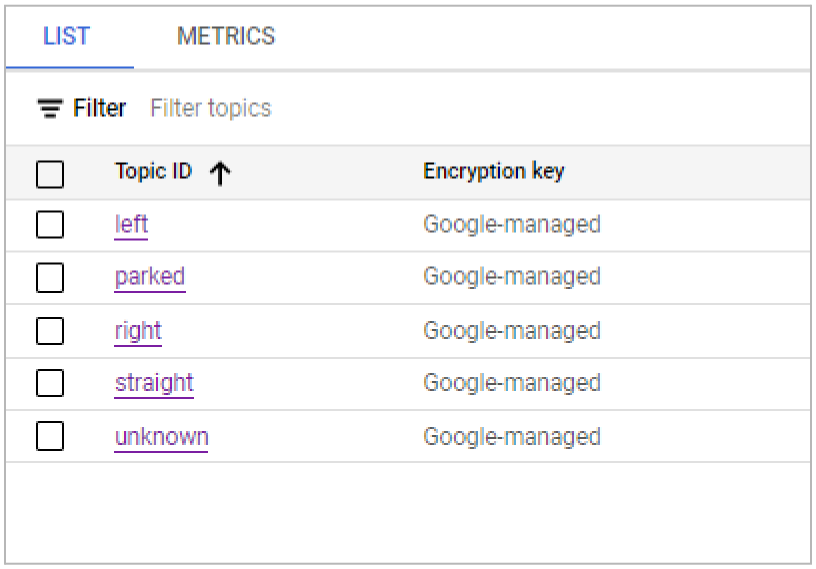
	Name ↑	Created	Location type
☐	ind_tracks	3 Apr 2024, 03:44:48	Multi-region
☐	track_trajectories	3 Apr 2024, 01:28:11	Multi-region

A schema for the track trajectories is created in BigQuery. This is where the results are going to be stored.



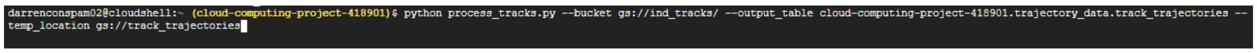
Field name	Type	Mode	Key	Collation	Default value	Policy tags	Description
recordingId	STRING	NULLABLE	-	-	-	-	-
trackId	INTEGER	NULLABLE	-	-	-	-	-
width	FLOAT	NULLABLE	-	-	-	-	-
height	FLOAT	NULLABLE	-	-	-	-	-
trajectory	STRING	NULLABLE	-	-	-	-	-
avgXVel	FLOAT	NULLABLE	-	-	-	-	-
avgYVel	FLOAT	NULLABLE	-	-	-	-	-
avgXAcc	FLOAT	NULLABLE	-	-	-	-	-
avgYAcc	FLOAT	NULLABLE	-	-	-	-	-
avgLonVel	FLOAT	NULLABLE	-	-	-	-	-
avgLatVel	FLOAT	NULLABLE	-	-	-	-	-
avgLonAcc	FLOAT	NULLABLE	-	-	-	-	-
avgLatAcc	FLOAT	NULLABLE	-	-	-	-	-

Topics are created. They are the points where data is fed. The publisher can feed data to a topic and the subscribers are able to receive the messages based on the topics they subscribed to.



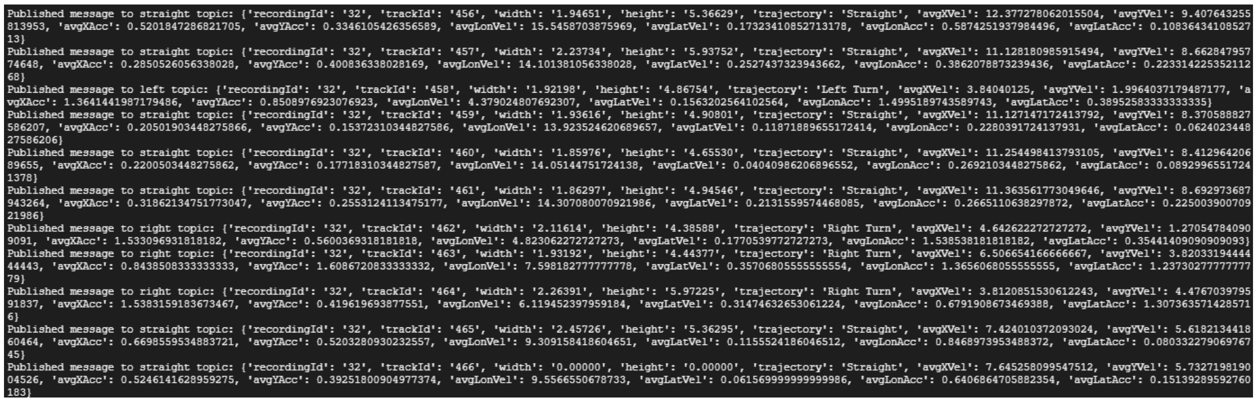
Topic ID	Encryption key
left	Google-managed
parked	Google-managed
right	Google-managed
straight	Google-managed
unknown	Google-managed

The following script is run to initialize the whole process. Take note of the arguments required to run the code. These include: the input bucket from which to read the data from, the output table to save the processed records to, and a second GCS bucket for temporary record collection.



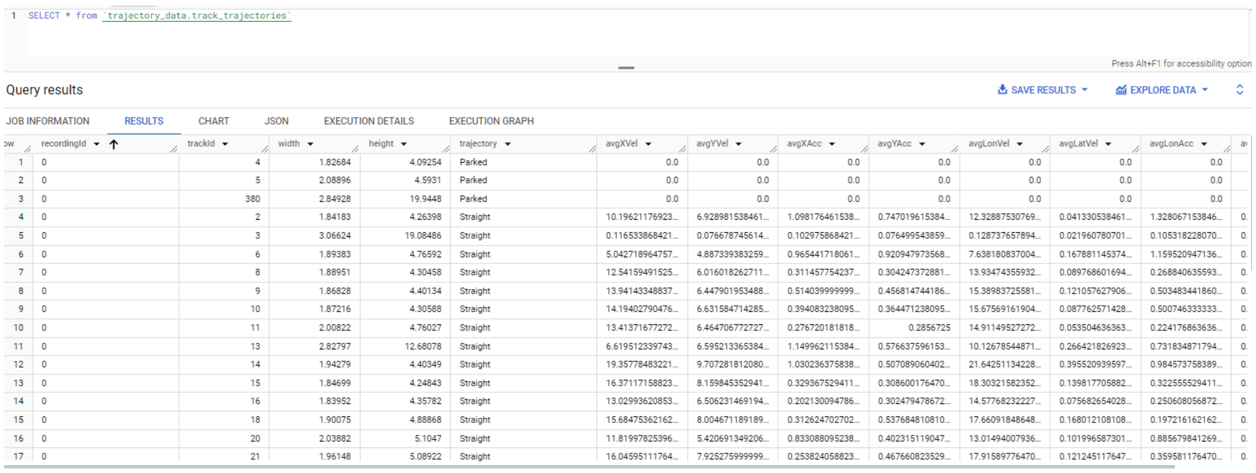
```
darrenconspam02@cloudshell:~ (cloud-computing-project-418901)$ python process_tracks.py --bucket gs://ind_tracks/ --output_table cloud-computing-project-418901.trajectory_data.track_trajectories --
temp_location gs://track_trajectories
```

The following figure provides a sample output of running the python script. The messages are now being published to the topics.



```
Published message to straight topic: {'recordingId': '32', 'trackId': '456', 'width': '1.94651', 'height': '5.36629', 'trajectory': 'Straight', 'avgXVel': 12.377278062015504, 'avgYVel': 9.407643255
813953, 'avgXAcc': 0.5201847286821705, 'avgYAcc': 0.3346105426356589, 'avgLonVel': 15.5458703875969, 'avgLatVel': 0.17323410852713178, 'avgLonAcc': 0.5874251937984496, 'avgLatAcc': 0.10836434108527
13}
Published message to straight topic: {'recordingId': '32', 'trackId': '457', 'width': '2.23734', 'height': '5.93752', 'trajectory': 'Straight', 'avgXVel': 11.128180985915494, 'avgYVel': 8.662847957
74648, 'avgXAcc': 0.2850526056338028, 'avgYAcc': 0.400836338028169, 'avgLonVel': 14.101381056338028, 'avgLatVel': 0.2527437323943662, 'avgLonAcc': 0.3862078873239436, 'avgLatAcc': 0.223314225352112
68}
Published message to left topic: {'recordingId': '32', 'trackId': '458', 'width': '1.92198', 'height': '4.86754', 'trajectory': 'Left Turn', 'avgXVel': 3.84040125, 'avgYVel': 1.9964037179487177, 'a
vgXAcc': 1.3641441987179486, 'avgYAcc': 0.8508976923076923, 'avgLonVel': 4.379024807692307, 'avgLatVel': 0.1563202564102564, 'avgLonAcc': 1.4995189743589743, 'avgLatAcc': 0.3895258333333335}
Published message to straight topic: {'recordingId': '32', 'trackId': '459', 'width': '1.93616', 'height': '4.90801', 'trajectory': 'Straight', 'avgXVel': 11.127147172413792, 'avgYVel': 8.370588827
586207, 'avgXAcc': 0.20501903448275866, 'avgYAcc': 0.15372310344827586, 'avgLonVel': 13.923524620689657, 'avgLatVel': 0.11871889655172414, 'avgLonAcc': 0.2280391724137931, 'avgLatAcc': 0.0624023448
27586206}
Published message to straight topic: {'recordingId': '32', 'trackId': '460', 'width': '1.85976', 'height': '4.65530', 'trajectory': 'Straight', 'avgXVel': 11.254498413793105, 'avgYVel': 8.412964206
89655, 'avgXAcc': 0.2200503448275862, 'avgYAcc': 0.17718310344827587, 'avgLonVel': 14.05144751724138, 'avgLatVel': 0.04040986206896552, 'avgLonAcc': 0.2692103448275862, 'avgLatAcc': 0.0892996551724
1378}
Published message to straight topic: {'recordingId': '32', 'trackId': '461', 'width': '1.86297', 'height': '4.94546', 'trajectory': 'Straight', 'avgXVel': 11.363561773049646, 'avgYVel': 8.692973687
943264, 'avgXAcc': 0.31862134751773047, 'avgYAcc': 0.2553124113475177, 'avgLonVel': 14.307080070921986, 'avgLatVel': 0.2131559574468085, 'avgLonAcc': 0.2665110638297872, 'avgLatAcc': 0.225003900709
21986}
Published message to right topic: {'recordingId': '32', 'trackId': '462', 'width': '2.11614', 'height': '4.38588', 'trajectory': 'Right Turn', 'avgXVel': 4.642622272727272, 'avgYVel': 1.27054784090
9091, 'avgXAcc': 1.533096931818182, 'avgYAcc': 0.5600369318181818, 'avgLonVel': 4.823062272727273, 'avgLatVel': 0.1770539772727273, 'avgLonAcc': 1.538538181818182, 'avgLatAcc': 0.35441409090909093}
Published message to right topic: {'recordingId': '32', 'trackId': '463', 'width': '1.93192', 'height': '4.44377', 'trajectory': 'Right Turn', 'avgXVel': 6.506654166666667, 'avgYVel': 3.82033194444
44443, 'avgXAcc': 0.8438508333333333, 'avgYAcc': 1.6086720833333332, 'avgLonVel': 7.598182777777778, 'avgLatVel': 0.3570680555555554, 'avgLonAcc': 1.3656068055555555, 'avgLatAcc': 1.23730277777777
79}
Published message to right topic: {'recordingId': '32', 'trackId': '464', 'width': '2.26391', 'height': '5.97225', 'trajectory': 'Right Turn', 'avgXVel': 3.8120851530612243, 'avgYVel': 4.4767039795
91837, 'avgXAcc': 1.5383159183673467, 'avgYAcc': 0.419619693877551, 'avgLonVel': 6.119452397959184, 'avgLatVel': 0.31474632653061224, 'avgLonAcc': 0.6791908673469388, 'avgLatAcc': 1.307363571428571
6}
Published message to straight topic: {'recordingId': '32', 'trackId': '465', 'width': '2.45726', 'height': '5.36295', 'trajectory': 'Straight', 'avgXVel': 7.424010372093024, 'avgYVel': 5.6182134418
60464, 'avgXAcc': 0.6698559534883721, 'avgYAcc': 0.5203280930232557, 'avgLonVel': 9.309158418604651, 'avgLatVel': 0.1155524186046512, 'avgLonAcc': 0.8468973953488372, 'avgLatAcc': 0.080332279069767
45}
Published message to straight topic: {'recordingId': '32', 'trackId': '466', 'width': '0.00000', 'height': '0.00000', 'trajectory': 'Straight', 'avgXVel': 7.645258099547512, 'avgYVel': 5.7327198190
04526, 'avgXAcc': 0.5246141628959275, 'avgYAcc': 0.39251800904977374, 'avgLonVel': 9.5566550678733, 'avgLatVel': 0.061569999999999986, 'avgLonAcc': 0.6406864705882354, 'avgLatAcc': 0.15139289592760
183}
```

The results are then saved in bigQuery. The following is a sample snapshot of the results.



```
1 SELECT * from `trajectory_data.track_trajectories`
```

Press Alt+F1 for accessibility option

Query results

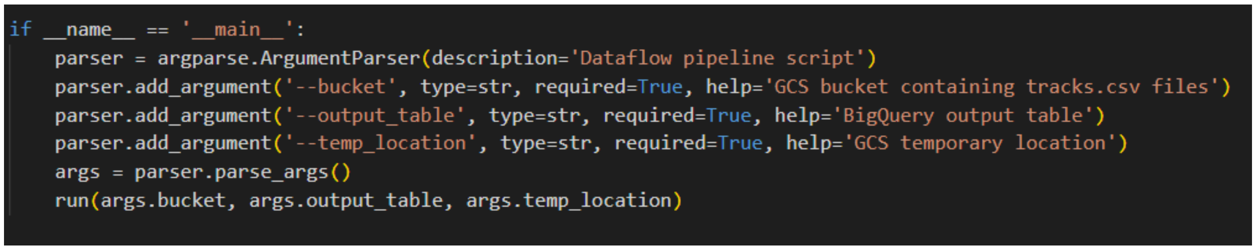
ow	recordingId	trackId	width	height	trajectory	avgXVel	avgYVel	avgXAcc	avgYAcc	avgLonVel	avgLatVel	avgLonAcc	a
1	0	4	1.82684	4.09254	Parked	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0	5	2.08896	4.5931	Parked	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0	380	2.84928	19.9448	Parked	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0	2	1.84183	4.26398	Straight	10.19621176923...	6.928981538461...	1.098176461538...	0.747019615384...	12.32887530769...	0.041330538461...	1.328067153846...	0.
5	0	3	3.06624	19.08486	Straight	0.116533868421...	0.076678745614...	0.102975868421...	0.076499543859...	0.128737657894...	0.021960780701...	0.105318228070...	0.
6	0	6	1.89383	4.76592	Straight	5.042718964757...	4.887339383259...	0.965441718061...	0.920947973568...	7.638180837004...	0.167881145374...	1.159520947136...	0.
7	0	8	1.88951	4.30458	Straight	12.54159491525...	6.016018262711...	0.311457754237...	0.304247372881...	13.93474355932...	0.089768601694...	0.268840635593...	0.
8	0	9	1.86828	4.40134	Straight	13.94143348837...	6.447901953488...	0.514039999999...	0.456814744186...	15.38983725581...	0.121057627906...	0.503483441860...	0.
9	0	10	1.87216	4.30588	Straight	14.19402790476...	6.631584714285...	0.394083238095...	0.364471238095...	15.67569161904...	0.087762571428...	0.500746333333...	0.
10	0	11	2.00822	4.76027	Straight	13.41371677272...	6.464706772727...	0.276720181818...	0.2856725	14.91149527272...	0.053504636363...	0.224176863636...	0.
11	0	13	2.82797	12.68078	Straight	6.619512339743...	6.595213365384...	1.149962115384...	0.576637596153...	10.12678544871...	0.266421826923...	0.731834871794...	0.
12	0	14	1.94279	4.40349	Straight	19.35778483221...	9.707281812080...	1.030236375838...	0.507089060402...	21.64251134228...	0.395520939597...	0.984573758389...	0.
13	0	15	1.84699	4.24843	Straight	16.37117158823...	8.159845352941...	0.329367529411...	0.308600176470...	18.30321582352...	0.139817705882...	0.322555529411...	0.
14	0	16	1.83952	4.35782	Straight	13.02993620853...	6.506231469194...	0.202130094786...	0.302479478672...	14.57768232227...	0.075682654028...	0.250608056872...	0.
15	0	18	1.90075	4.88868	Straight	15.68475362162...	8.004671189189...	0.312624702702...	0.537684810810...	17.66091848648...	0.168012108108...	0.197216162162...	0.
16	0	20	2.03882	5.1047	Straight	11.81997825396...	5.420691349206...	0.833088095238...	0.402315119047...	13.01494007936...	0.101996587301...	0.885679841269...	0.
17	0	21	1.96148	5.08922	Straight	16.04595111764...	7.925275999999...	0.253824058823...	0.467660823529...	17.91589776470...	0.121245117647...	0.359581176470...	0.

From the stored results, we can conduct some validation tests to determine if computed trajectories, along with average speed and acceleration metrics, were correctly computed.

8. Code Implementation

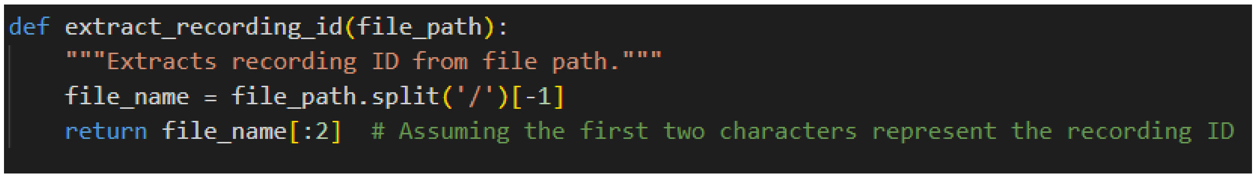
The following provides an overview of the code design to construct this pipeline. A description along with code snippets is provided below.

Three arguments are expected in this code snippet, the bucket which contains the stored data to be used, output table, where the processed records will be written to and temp location, where the data will be stored temporarily. The '**run**' function uses the arguments passed. It is used to orchestrate the dataflow pipeline.



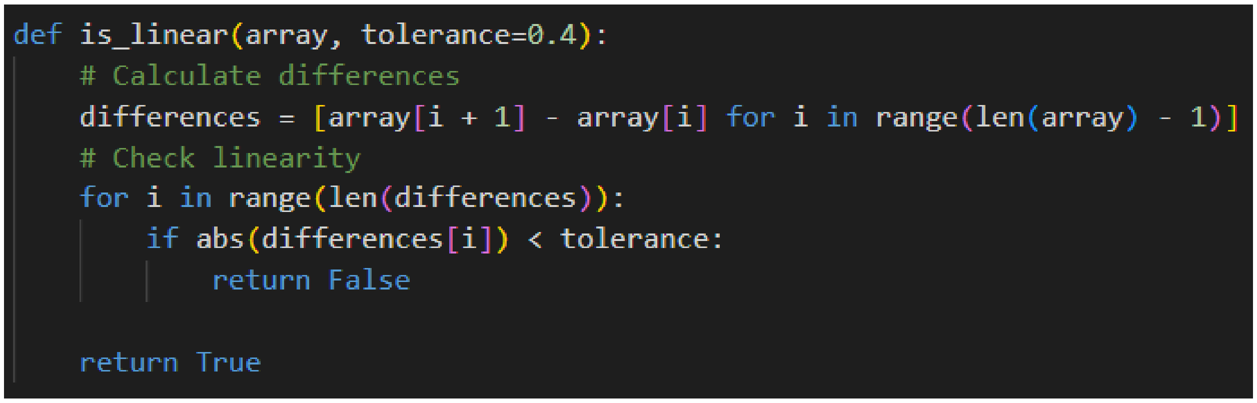
```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Dataflow pipeline script')
    parser.add_argument('--bucket', type=str, required=True, help='GCS bucket containing tracks.csv files')
    parser.add_argument('--output_table', type=str, required=True, help='BigQuery output table')
    parser.add_argument('--temp_location', type=str, required=True, help='GCS temporary location')
    args = parser.parse_args()
    run(args.bucket, args.output_table, args.temp_location)
```

Recording ID extraction algorithm
This function is used to extract the recording ID from the file path of each CSV file. This assumes that the file name structure follows the pattern: '**XX_tracks.csv'**. An alternative approach to get this information could involve taking the first record of the CSV file and reading the first column to retrieve the recording ID.



```
def extract_recording_id(file_path):
    """Extracts recording ID from file path."""
    file_name = file_path.split('/')[-1]
    return file_name[:2]  # Assuming the first two characters represent the recording ID
```

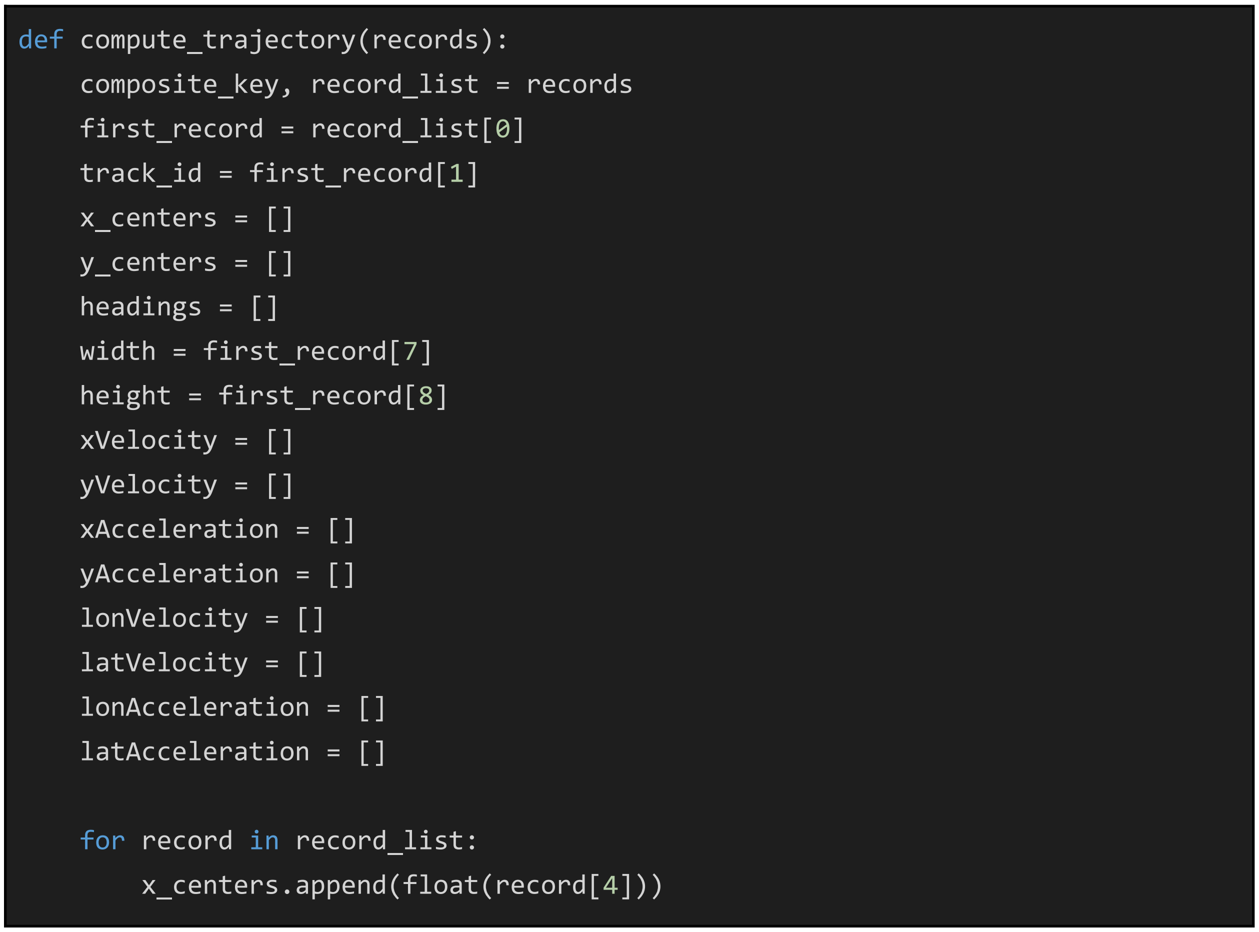
Linear model classification algorithm
The algorithm is used to check if a given array of numbers follows a linear pattern. The motive behind it is to determine if a x center coordinates of the object resembles uniform motion. If the object slows down of stops, this is observed as a promising candidate for left/right turn classification.



```python
def is_linear(array, tolerance=0.4):
    # Calculate differences
    differences = [array[i + 1] - array[i] for i in range(len(array) - 1)]
    # Check linearity
    for i in range(len(differences)):
        if abs(differences[i]) < tolerance:
            return False

    return True
```

Trajectory classification algorithm

In this algorithm, the rotations which are the heading changes of the vehicle and the x centroids, which are the position information of the objects, are taken in as variables and used to classify the behavior of a vehicle. There are 4 categories of trajectory behavior that a vehicle can be classified into: “straight” , “left turn” , “right turn”, or “parked”. The classification is done based on the examination of centroid locations and heading differences.

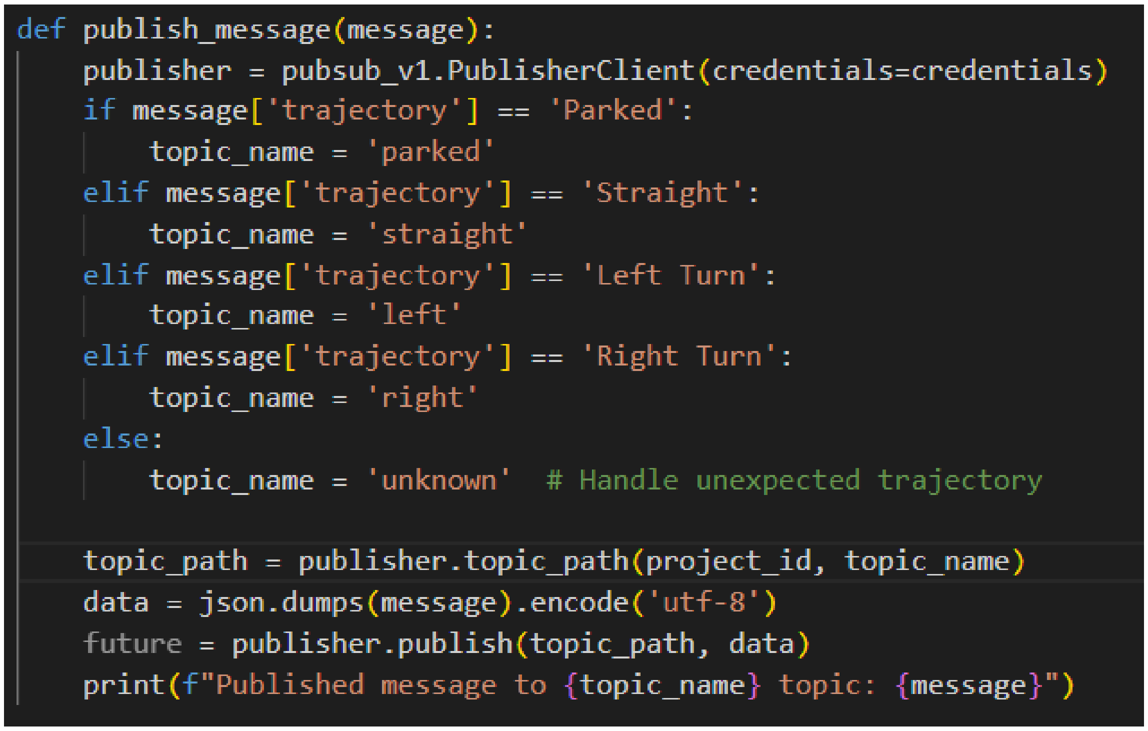


```python
def compute_trajectory(records):
    composite_key, record_list = records
    first_record = record_list[0]
    track_id = first_record[1]
    x_centers = []
    y_centers = []
    headings = []
    width = first_record[7]
    height = first_record[8]
    xVelocity = []
    yVelocity = []
    xAcceleration = []
    yAcceleration = []
    lonVelocity = []
    latVelocity = []
    lonAcceleration = []
    latAcceleration = []

    for record in record_list:
        x_centers.append(float(record[4]))
        y_centers.append(float(record[5]))
        headings.append(float(record[6]))
        # Take absolute values to measure speed and acceleration
        xVelocity.append(abs(float(record[9])))
        yVelocity.append(abs(float(record[10])))
        xAcceleration.append(abs(float(record[11])))
        yAcceleration.append(abs(float(record[12])))
        lonVelocity.append(abs(float(record[13])))
        latVelocity.append(abs(float(record[14])))
        lonAcceleration.append(abs(float(record[15])))
        latAcceleration.append(abs(float(record[16])))

    # Compute the differences in heading between consecutive frames
    heading_diff = np.diff(headings)



    # Calculate the average heading difference
    avg_heading_diff = np.mean(heading_diff)

    # Calculate the average of other attributes
    avg_xVelocity = np.mean(xVelocity)
    avg_yVelocity = np.mean(yVelocity)
    avg_xAcceleration = np.mean(xAcceleration)
    avg_yAcceleration = np.mean(yAcceleration)
    avg_lonVelocity = np.mean(lonVelocity)
    avg_latVelocity = np.mean(latVelocity)
    avg_lonAcceleration = np.mean(lonAcceleration)
    avg_latAcceleration = np.mean(latAcceleration)

    # Check if the average heading difference is 0 -> T = Stationary object
    if avg_heading_diff == 0:
        trajectory = "Parked"
    else:
        # Check if x_center metric resembles a linear plot -> T = Straight
trajectory
        if is_linear(x_centers):
            trajectory = "Straight"
        else:
            if avg_heading_diff < 0:
                # Check the direction of change in x_centers -> if x_center of
last frame is greater than intial frame, classify as left
                dx = x_centers[-1] - x_centers[0]
                if dx > 0:
                    trajectory = "Left Turn"
                else:
                    trajectory = "Right Turn"
            else:
                trajectory = "Straight"
```

Also note, the prepared message should follow the schema defined earlier in BigQuery to ensure that all information from the record is saved correctly.

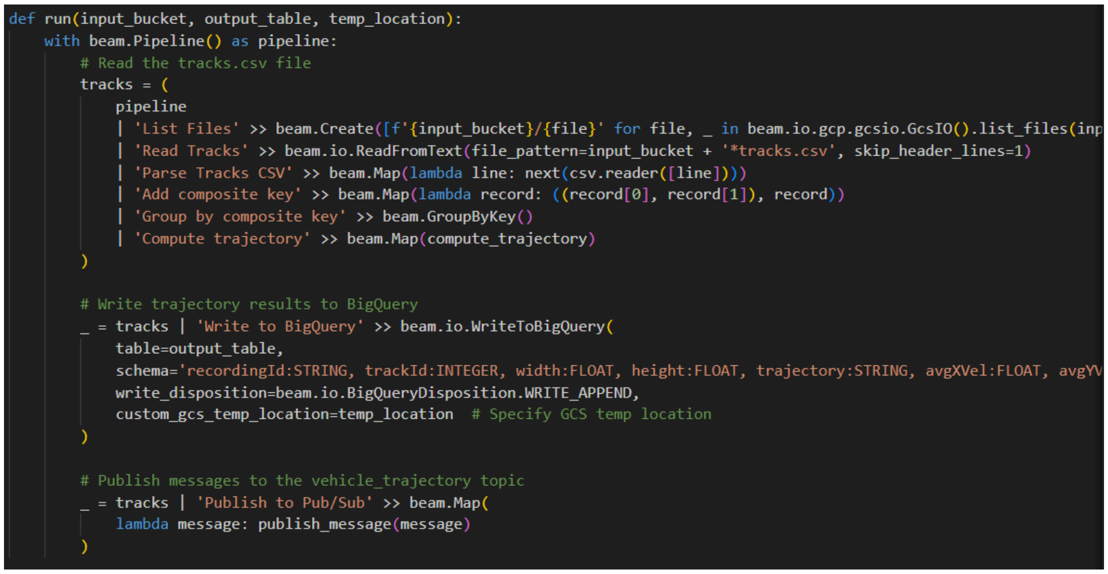
```
    # Prepare the message
    message = {
        'recordingId': first_record[0],
        'trackId': track_id,
        'width': width,
        'height': height,
        'trajectory': trajectory,
        'avgXVel': avg_xVelocity,
        'avgYVel': avg_yVelocity,
        'avgXAcc': avg_xAcceleration,
        'avgYAcc': avg_yAcceleration,
        'avgLonVel': avg_lonVelocity,
        'avgLatVel': avg_latVelocity,
        'avgLonAcc': avg_lonAcceleration,
        'avgLatAcc': avg_latAcceleration
    }

    return message
```

Pub/Sub message publishing algorithm
This algorithm is used to publish messages to Google Cloud Pub/Sub topics. The topic to which the messages are published to is based on the computed trajectory of the object. For example, if an object's trajectory is classified as 'Straight,' the message for this object will be published to the 'straight' Pub/Sub topic.



```python
def publish_message(message):
    publisher = pubsub_v1.PublisherClient(credentials=credentials)
    if message['trajectory'] == 'Parked':
        topic_name = 'parked'
    elif message['trajectory'] == 'Straight':
        topic_name = 'straight'
    elif message['trajectory'] == 'Left Turn':
        topic_name = 'left'
    elif message['trajectory'] == 'Right Turn':
        topic_name = 'right'
    else:
        topic_name = 'unknown'  # Handle unexpected trajectory

    topic_path = publisher.topic_path(project_id, topic_name)
    data = json.dumps(message).encode('utf-8')
    future = publisher.publish(topic_path, data)
    print(f"Published message to {topic_name} topic: {message}")
```

Apache beam pipeline
The ‘**run**’ function orchestrates the Apache Beam pipeline. It reads the track record data from the bucket, parses the records, computed trajectories, and write the processed data to a BigQuery table. Additionally, it publishes the data to Pub/Sub topics based on the computed trajectory, as mentioned earlier.



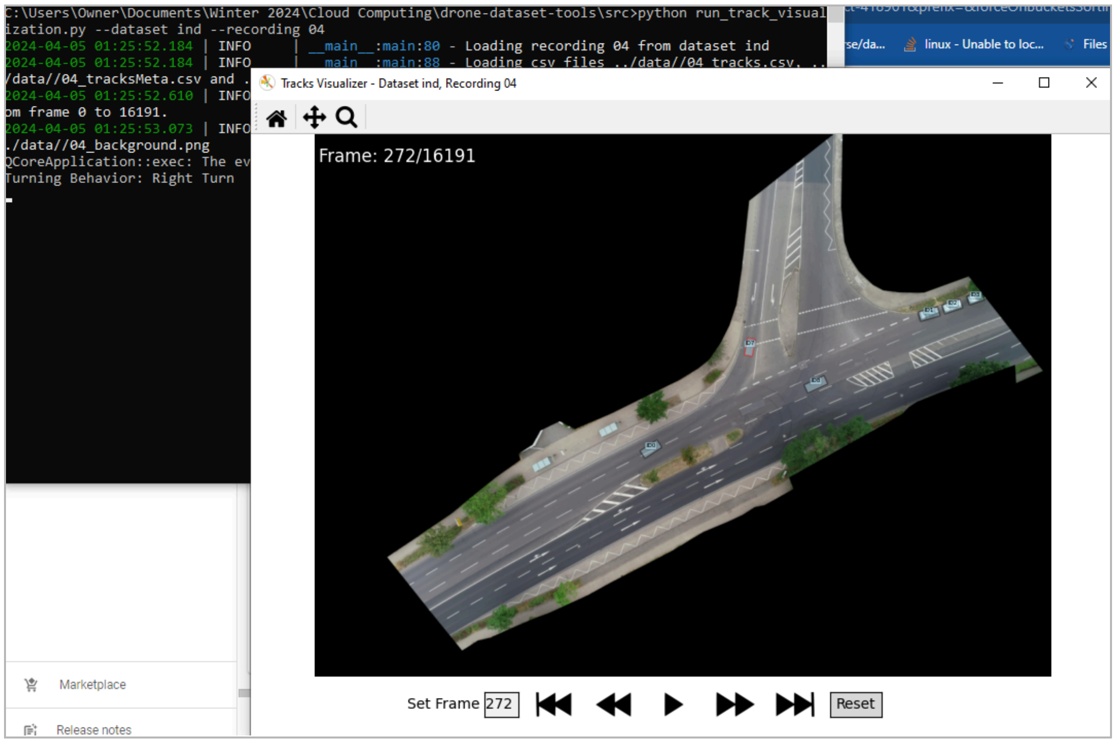
```python
def run(input_bucket, output_table, temp_location):
    with beam.Pipeline() as pipeline:
        # Read the tracks.csv file
        tracks = (
            pipeline
            | 'List Files' >> beam.Create([f'{input_bucket}/{file}' for file, _ in beam.io.gcp.gcsio.GcsIO().list_files(inp
            | 'Read Tracks' >> beam.io.ReadFromText(file_pattern=input_bucket + '*tracks.csv', skip_header_lines=1)
            | 'Parse Tracks CSV' >> beam.Map(lambda line: next(csv.reader([line])))
            | 'Add composite key' >> beam.Map(lambda record: ((record[0], record[1]), record))
            | 'Group by composite key' >> beam.GroupByKey()
            | 'Compute trajectory' >> beam.Map(compute_trajectory)
        )

        # Write trajectory results to BigQuery
        _ = tracks | 'Write to BigQuery' >> beam.io.WriteToBigQuery(
            table=output_table,
            schema='recordingId:STRING, trackId:INTEGER, width:FLOAT, height:FLOAT, trajectory:STRING, avgXVel:FLOAT, avgYV
            write_disposition=beam.io.BigQueryDisposition.WRITE_APPEND,
            custom_gcs_temp_location=temp_location  # Specify GCS temp location
        )

        # Publish messages to the vehicle_trajectory topic
        _ = tracks | 'Publish to Pub/Sub' >> beam.Map(
            lambda message: publish_message(message)
        )
```

10. Test Cases

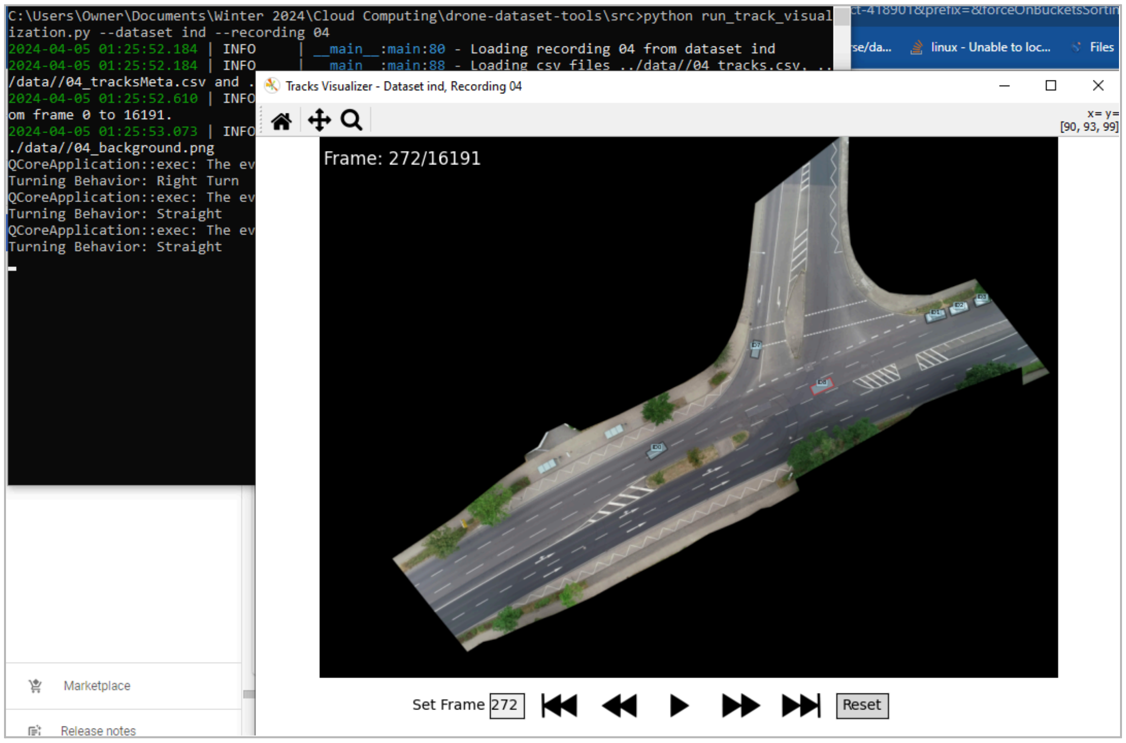
The following table provides a list of test cases to test each component and the system as a whole.

Component/Tool	Test Case
Data Ingestion	Read data from source accurately
	Publish messages to Pub/Sub topic
Data Processing (test cases will be similar across all Dataflow jobs)	Validate Apache Beam pipeline
	Confirm data transformations
	Test publishing processed data to Pub/Sub topic
Trajectory Computation & Classification	Compute vehicle trajectory given the vehicle data
	Ensure correct classification to assure safety validation
Google Pub/Sub	Create Pub/Sub topics successfully
	Validate message publishing and subscription
	Check message ordering and delivery
Looker Studio	Validate data visualization tasks
	Ensure accurate representation
System	End-to-End Data Flow Test
	Performance Test
	Fault Tolerance Test

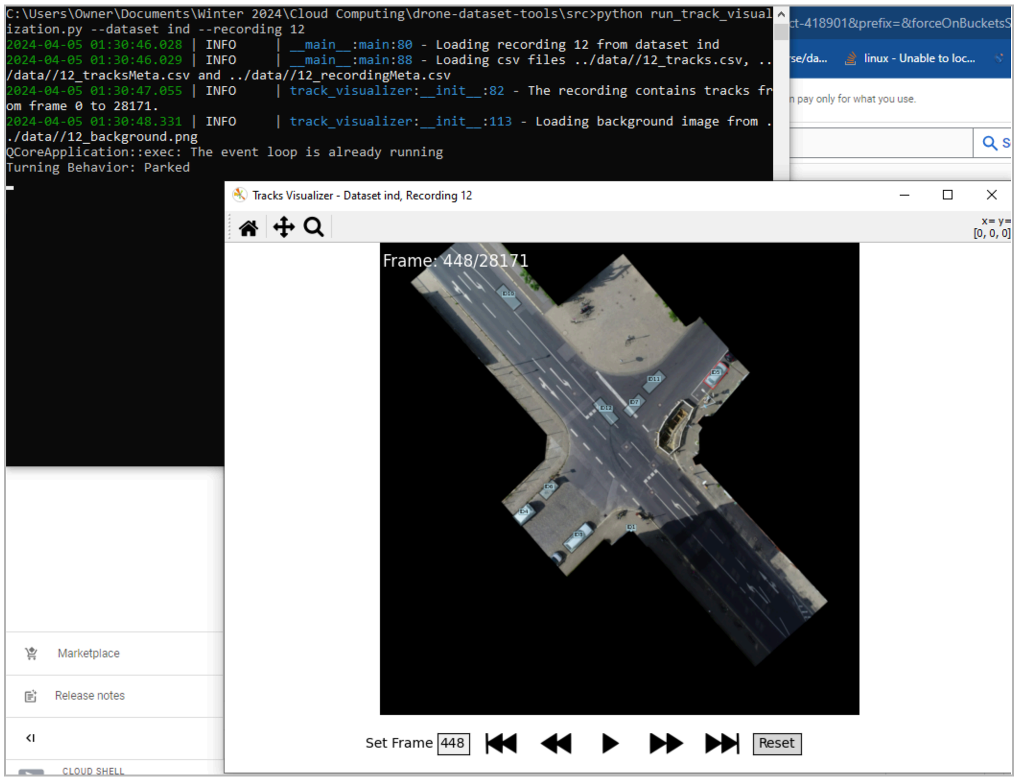
As part of the validation phase, the trajectories of various objects were observed through running simulations of the recordings. By using a drone dataset **tool** and adding the computation logic to the source code, we are able to verify computed trajectories of the objects against their real trajectories. The following provides sample snapshots of running these tests. The object selected is marked in a red outline, and its computed trajectory is displayed in the console window.



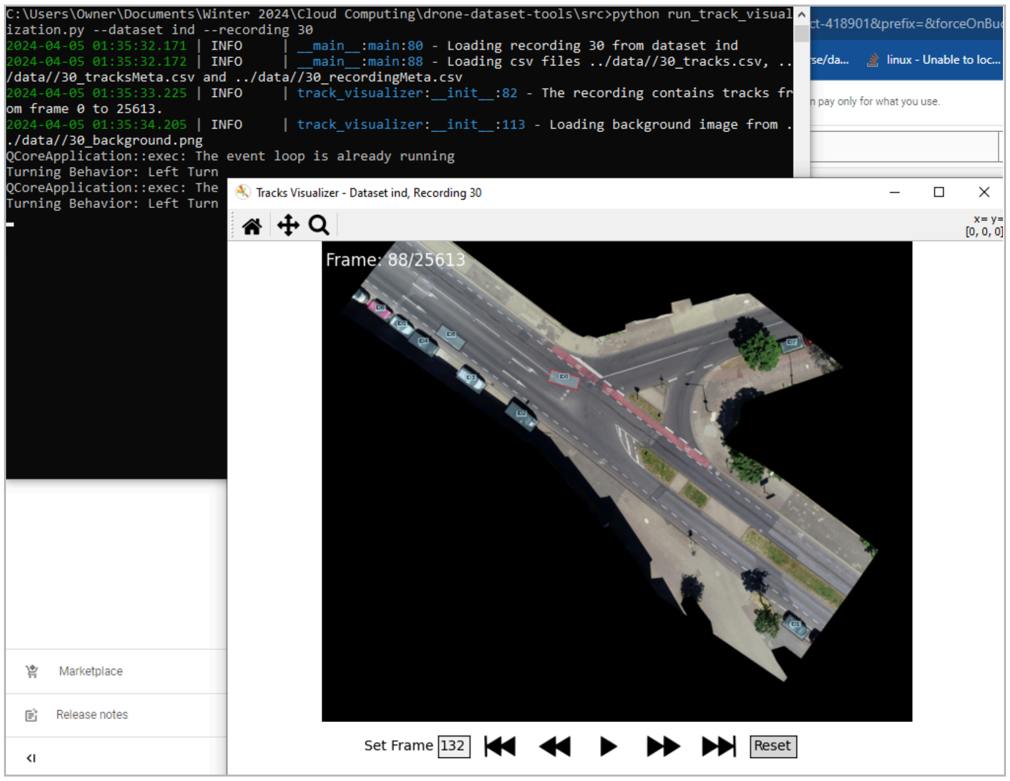
```
C:\Users\Owner\Documents\Winter 2024\Cloud Computing\drone-dataset-tools\src>python run_track_visual
ization.py --dataset ind --recording 04
2024-04-05 01:25:52.184 | INFO     | __main__:main:80 - Loading recording 04 from dataset ind
2024-04-05 01:25:52.184 | INFO     | __main__:main:88 - Loading csv files ../data//04_tracks.csv, ..
/data//04_tracksMeta.csv and .
2024-04-05 01:25:52.610 | INFO
om frame 0 to 16191.
2024-04-05 01:25:53.073 | INFO
./data//04_background.png
QCoreApplication::exec: The ev
Turning Behavior: Right Turn
```



```
C:\Users\Owner\Documents\Winter 2024\Cloud Computing\drone-dataset-tools\src>python run_track_visual
ization.py --dataset ind --recording 04
2024-04-05 01:25:52.184 | INFO     | __main__:main:80 - Loading recording 04 from dataset ind
2024-04-05 01:25:52.184 | INFO     | __main__:main:88 - Loading csv files ../data//04_tracks.csv, ..
/data//04_tracksMeta.csv and .
2024-04-05 01:25:52.610 | INFO
om frame 0 to 16191.
2024-04-05 01:25:53.073 | INFO
./data//04_background.png
QCoreApplication::exec: The ev
Turning Behavior: Right Turn
QCoreApplication::exec: The ev
Turning Behavior: Straight
QCoreApplication::exec: The ev
Turning Behavior: Straight
```



```
C:\Users\Owner\Documents\Winter 2024\Cloud Computing\drone-dataset-tools\src>python run_track_visual
ization.py --dataset ind --recording 12
2024-04-05 01:30:46.028 | INFO     | __main__:main:80 - Loading recording 12 from dataset ind
2024-04-05 01:30:46.029 | INFO     | __main__:main:88 - Loading csv files ../data//12_tracks.csv, ..
/data//12_tracksMeta.csv and ../data//12_recordingMeta.csv
2024-04-05 01:30:47.055 | INFO     | track_visualizer:__init__:82 - The recording contains tracks fr
om frame 0 to 28171.
2024-04-05 01:30:48.331 | INFO     | track_visualizer:__init__:113 - Loading background image from .
./data//12_background.png
QCoreApplication::exec: The event loop is already running
Turning Behavior: Parked
```



```
C:\Users\Owner\Documents\Winter 2024\Cloud Computing\drone-dataset-tools\src>python run_track_visual
ization.py --dataset ind --recording 30
2024-04-05 01:35:32.171 | INFO     | __main__:main:80 - Loading recording 30 from dataset ind
2024-04-05 01:35:32.172 | INFO     | __main__:main:88 - Loading csv files ../data//30_tracks.csv, ..
/data//30_tracksMeta.csv and ../data//30_recordingMeta.csv
2024-04-05 01:35:33.225 | INFO     | track_visualizer:__init__:82 - The recording contains tracks fr
om frame 0 to 25613.
2024-04-05 01:35:34.205 | INFO     | track_visualizer:__init__:113 - Loading background image from .
./data//30_background.png
QCoreApplication::exec: The event loop is already running
Turning Behavior: Left Turn
QCoreApplication::exec: The
Turning Behavior: Left Turn
```

11. Roles and Responsibilities

The following provides a breakdown of the roles and responsibilities of each team member.

Name	Student Number	Role
Laksumi Somaskanthamoorthy	100782723	Develop part of the Apache Beam pipeline to process record data from GCS. Implement data parsing, key grouping, and writing to BigQuery.
Ashley Tyrone	100786450	Optimization of the main function; performing trajectory validation.
Lauryne Zachary	100797920	Identify and set up GCP resources, including buckets for storing track record data and temporary processed data. Configure access permissions and security settings for GCP services.
Ram Baldeo	100701013	Create tests to validate functionality of pipeline and trajectory computations. Report bugs to the team.
Darren Constantine	100782701	Develop trajectory computation algorithm. Construct a trajectory classification analytic report on Looker Studio from the collected results saved on BigQuery.

Collaboration Plan and Communication Strategy
This project will be split into parts that members of the team will approach from the different roles they are assigned. Tasks will be completed in a collaborative manner that will allow members to discuss any issues they may encounter.

12. Conclusion

In conclusion, a criticality metric was developed using various tools, such as Google Cloud Platform (GCP) to implement the Pub/Sub approach, as well as Apache Beam in order to process the large inD dataset that is provided. Google Dataflow was utilized to create a pipeline between the two platforms. The aforementioned objectives have been all met successfully. Through this project, objects that are present in the dataset are extracted and identified. With Apache Beam and Google Dataflow, data from the computed turning behavior, as well as the vehicle trajectory of the objects were all processed, analyzed, and classified. Finally, Google Pub/Sub was utilized to allow users to subscribe to their preferred trajectory notification.

References

[1] S. Noakes, “3 self-driving cars hit Ontario Roads under new pilot | CBC News,” CBCnews, https://www.cbc.ca/news/business/automated-vehicles-1.3870605 (accessed Mar. 30, 2024).

[2] “What is an autonomous car? – how self-driving cars work,” Synopsys, https://www.synopsys.com/automotive/what-is-autonomous-car.html (accessed Mar. 30, 2024).

[3] P. Junietz, F. Bonakdar, B. Klamann, and H. Winner, “Criticality metric for the safety validation of automated driving using model predictive trajectory optimization,” 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 2018. doi:10.1109/itsc.2018.8569326

[4] H. Hu, Y. Wen, T.-S. Chua, and X. Li, “Toward scalable systems for Big Data Analytics: A technology tutorial,” IEEE Access, vol. 2, pp. 652–687, Jun. 2014. doi:10.1109/access.2014.2332453

[5] “InD - levelxdata - real-world scenario data,” leveLXData, https://levelxdata.com/ind-dataset/ (accessed Mar. 31, 2024).

[6] “Programming model for Apache Beam | Cloud Dataflow,” *Google Cloud*. https://cloud.google.com/dataflow/docs/concepts/beam-programming-model#:~:text=Apache%20Beam%20is%20an%20open (accessed Mar. 31, 2024).