Technical Report

Traffic Simulation with Dynamic and Machine Learning Routing Prediction

Gennaro De Luca, Yinong Chen, Chengxu Liang, Drew Mudry, Gyllian Gaylor, Hannah Buell, Anik Rahman

School of Computing and Augmented Intelligence
University: Arizona State University

July 2022

Embedded System Design

ABSTRACT

This project develops a traffic simulator that allows students to use a visual programming language to program a car to navigate through the traffic to travel from point A to point B. The visual programming language used is the ASU VIPLE (Visual IoT/Robotics Programming Language Environment). The traffic simulator is developed using Microsoft Unity game engine. The traffic is generated through mapping New York's Manhattan traffic dataset into our traffic simulator to create realistic traffic pattern. Three algorithms are implemented to test the traversing algorithm effectiveness: (1) Simple Static Dijkstra's Algorithm using road section length as graph weights; (2) Dynamic Dijkstra's Algorithm that takes dynamic traffic into consideration and recalculates the shortest path at every intersection; (3) Machine-Learning based Dijkstra's Algorithm that predicts the best route based on the past traffic dataset training. Both VIPLE and Unity traffic simulator are performance demanding. We use Intel GNS-V40 processor board to run both VIPLE and Unity traffic simulator. We also run physical robot as the "avatar" of the simulated robot to show the programmed car moving outside the simulation into physical space.

Keywords: Dijkstra, VIPLE, Machine learning, Unity Traffic Simulator, traffic

Chapter 1 Introduction

1.1 Introduction

In this work, we describe the system used to calculate the shortest path to a destination in Manhattan, including the algorithms used to calculate the path, the environment and data sets used for simulating it, and the simulation experiment. The primary goal of this project is to provide a system for educational purposes, enabling students to gain practical experience with machine learning, shortest path algorithms, and traffic simulations. The project was extended with the novel hardware as an entry to the Intel Cup Competition through cooperation from the team's students and faculty at Arizona State University (ASU). The development of this project includes multiple key subgoals, including adopting the data set and machine learning model, mapping the Manhattan map and its traffic data to Unity simulator based on the current mapping from Maricopa data, rerouting the traffic density data sent to the web heat map to VIPLE, creating a new Dijkstra Algorithm implementation that takes AI predictions of traffic density on each street, and linking VIPLE program and Unity Simulator and their communication through JSON. This report is organized as follows. Section 2 introduces the Dijkstra algorithm and describes the applications and implementation of the algorithm. Section 3 discusses the experiment's environment, including the platforms the experiment is run on and how it works. Section 4 covers data collection and mapping to describe where the data was sourced and organized. Section 5 discusses data processing, training, and model establishment to explain how the data is used. Section 6 discusses the final experimentation and simulation.

1.2 Related Work

This project builds on works developed by teams at ASU, including an undergraduate honor's thesis, a master's thesis, and other capstone team projects. This section will introduce the key contributions from these other projects.

1.2.1 A Graph-Based Machine Learning Approach to Realistic Traffic Volume Generation

This honor's thesis explores the potential for realistic and accurate generation of hourly traffic volume with machine learning (ML), using the ground-truth data of Manhattan Road segments collected by the New York State Department of Transportation (NYSDOT). This work provides a baseline for correctly determining the shortest route to navigate Manhattan roads that are mapped in the traffic simulator [1][2][3].

1.2.2 Dynamic Routing Algorithm for Unity Traffic Simulator based on Real Traffic Data

This master's thesis and its capstone precursors explores the creation of a simulated environment similar to but simpler than Google Maps. It focuses on creating realistic traffic data for the Unity Traffic Simulator and implementing a dynamic routing algorithm in VIPLE. The traffic data is generated from the Arizona Maricopa County traffic data. This work provides the baseline for the simulated traffic components of the project. However, those works were targeted at Arizona traffic data, whereas our project employs Manhattan data and novel ML extensions.

Chapter 2 Dijkstra’s Shortest Path Algorithm

2.1 Introduction

This project uses Dijkstra’s algorithm to calculate the route that the vehicle will take. We provide three different implementations of Dijkstra’s algorithm which are the static, dynamic, and ML-enhanced implementations. Each implementation will take a series of weights listed as times which will be used in calculating the route the vehicle will take. The ML model is used for predicting traffic data. The corresponding Dijkstra’s algorithm code is implemented as a C# Code Activity in VIPLE. A min heap is used in the Dijkstra’s algorithm as the data structure to store the nodes, edges, weights, and the adjacency list information. The output of the C# code activity is stored in two variables, “current” and “path”. The “current” variable stores the intersection the vehicle is at. The second variable is “path” which is the planned path to the destination that Dijkstra’s algorithm will calculate using the current intersection. The implementation takes three inputs, “src”, “dst”, and a JSON file. The JSON file will contain all the map and traffic data that is received from the simulator and store it into a graph that is needed to calculate the shortest route. All 3 inputs are necessary to calculate the route from the destination to the source.

2.2 Static Dijkstra’s Algorithm

The static Dijkstra’s algorithm will calculate the complete path at the beginning of the route and follow that route without any changes until the vehicle reaches the desired destination [4]. If there is heavy traffic later in the route due to change over time, this implementation will not change to reflect that. Rather, it will always follow the predetermined route.

2.3 Dynamic Dijkstra’s Algorithm

The dynamic Dijkstra’s algorithm will calculate the complete path at the beginning of the route as in the static approach. However, the dynamic approach will recalculate the route as the car arrives at each intersection to find the new optimal route. By using the new traffic data, it will recalculate the route and use this new recalculated route if a better route is found [3].

2.4 ML-Enhanced Dijkstra’s Algorithm

The ML-enhanced algorithm will start by using an ML model to predict the traffic data. Using this data and other necessary information such as the edges, nodes, and weights, the Dijkstra’s algorithm code activity calculates the complete path.

Chapter 3 Environment

3.1 VIPLE

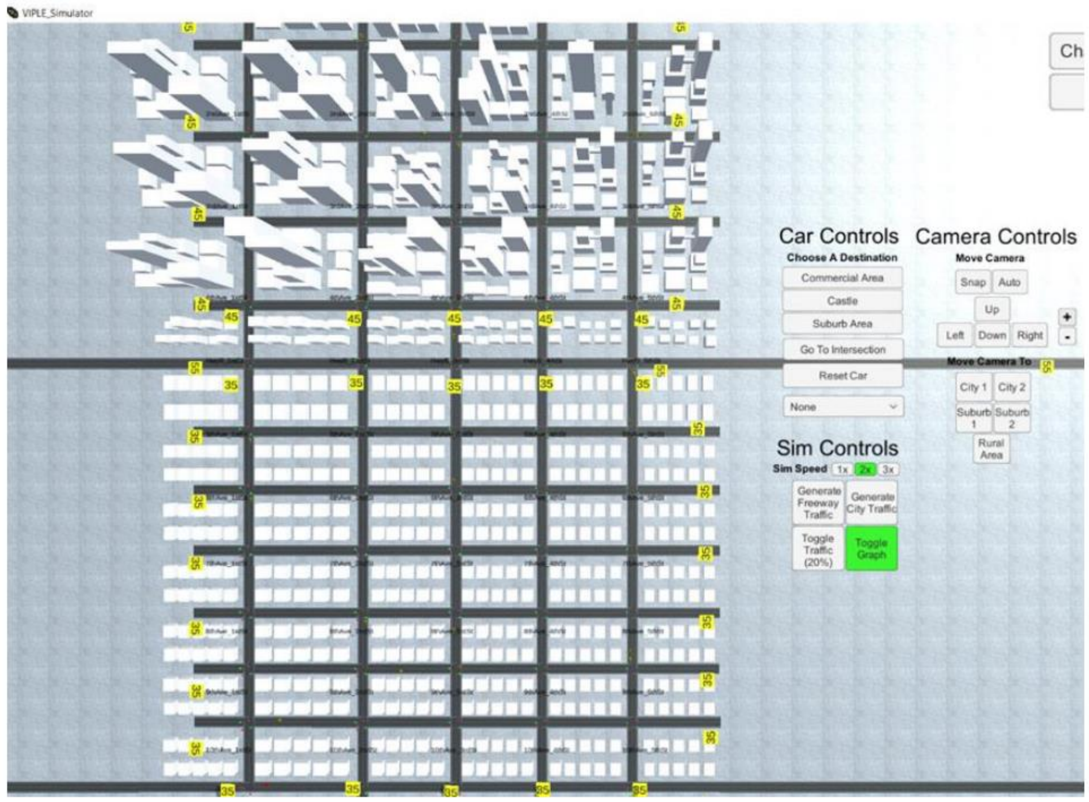
The environment used in this project is ASU VIPLE (Visual IoT/Robotics Programming Language Environment). The initial version of VIPLE is based on Microsoft Visual Programming Language (VPL), which was published in 2006 and adopted by many universities and schools, including Arizona State University (ASU), for computer science and robotics education [7]. Various simulators have been implemented in VIPLE, including both desktop simulators and Web simulators [8]. Traffic simulation is one of the functions that has been recently added to the VIPLE environment.

3.2 Unity

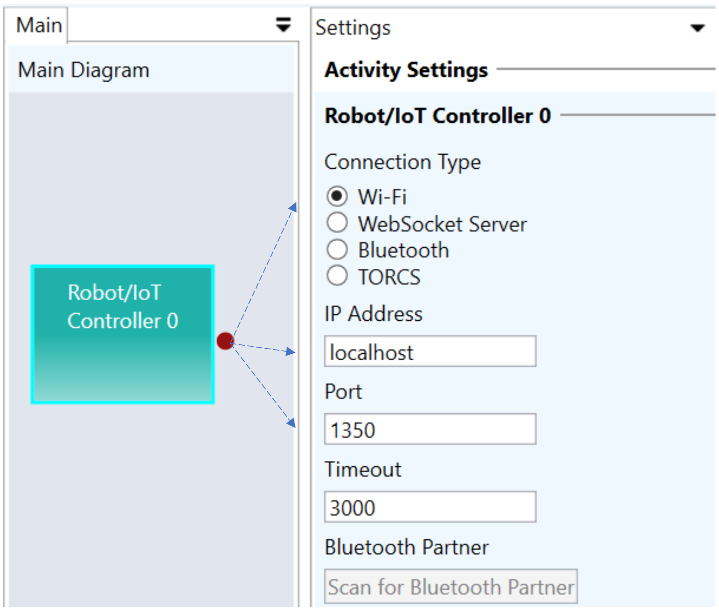
This project focuses on the traffic simulator, which is a desktop simulator developed using the Unity game engine. The simulator's appearance is grid-looking and contains different areas on the map. Yellow cars on the roads are the cars that are spawned by the simulator at any position on the roads with random driving directions, which can be used to reflect the traffic. The yellow cars can also be made to reflect an input traffic dataset. The red car is VIPLE programmable, which allows users to develop programs to drive the red car manually or autonomously.

3.3 Environment Applications

Upon selection of a portion of Manhattan, we required traffic data. The selected roads are mapped to the traffic simulator according to the geographical position in Manhattan. The real traffic data was incorporated into the traffic simulator through a JSON data format and the Unity platform where the traffic simulator is developed.



The traffic simulator is arranged in a grid. The whole traffic simulator is stored as a graph data structure. Each road is stored as an edge, and intersections are represented as vertices/nodes. The adjacency list is formed by the neighboring intersections of each edge. The weight for each segment of road is represented as the Manhattan distance and number of cars on the road. The traffic simulator generates a JSON format.



Activity Settings

Robot/IoT Controller 0

Connection Type

[✓ Wi-Fi] WebSocket Server Bluetooth TORCS

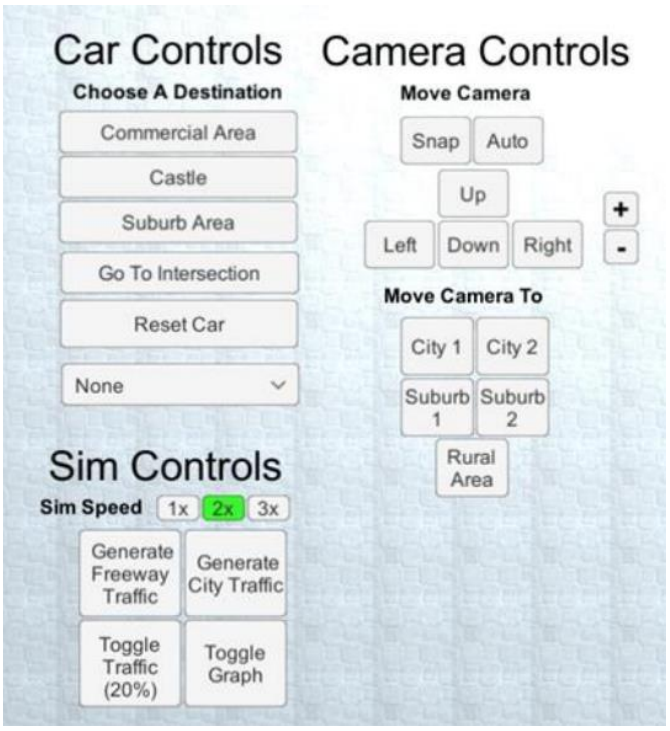
IP Address: localhost

Port: 1350

Timeout: 3000

Bluetooth Partner: Scan for Bluetooth Partner

To communicate between VIPLE and the traffic simulator, firstly Robot/IoT Controller service needs to be configured properly to connect the traffic simulator with VIPLE. The image above presents the proper configuration applied for this project. Port number 1350 is used to connect the main traffic simulator with VIPLE, which is used for this project. Robot/IoT Message In is the service that can receive JSON files from the traffic simulator by partnering with the Robot/IoT Controller configured earlier. Robot/IoT Message Out will be used to send JSON files to the traffic simulator also by partnering with the Robot/IoT controller that has been configured. The traffic simulator will receive the JSON messages and respond accordingly.



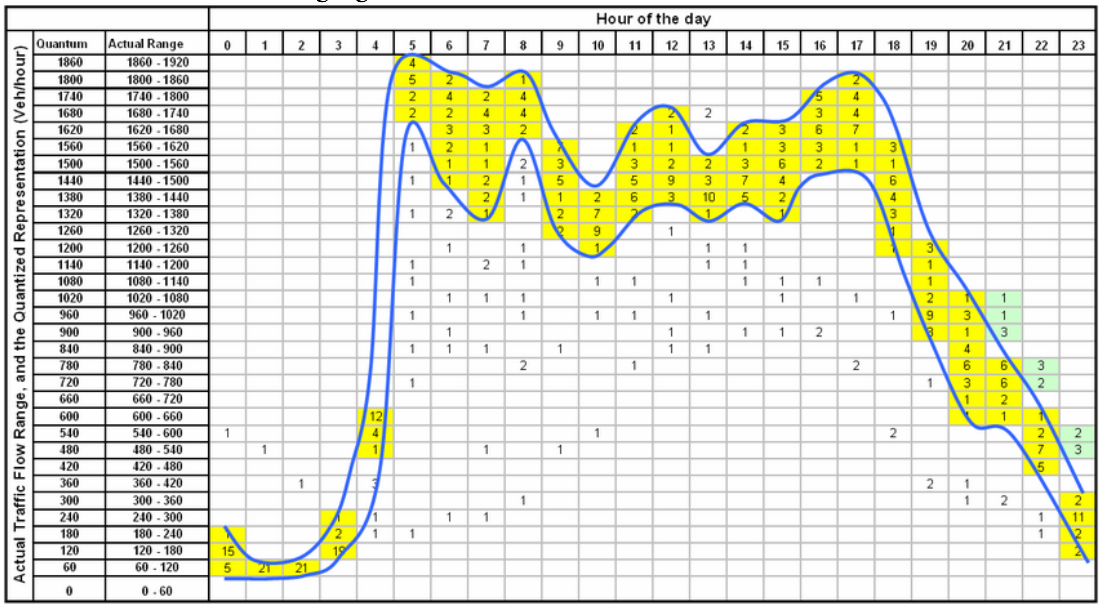
Most of the control buttons in the menu as shown above are straightforward to understand. The Car Controls section's buttons will take the red car to those areas of the map accordingly. The dropdown list includes different traffic patterns that control the number of yellow cars on the map, which is also the traffic density for the simulator. Camera Controls help to navigate the traffic simulator to different areas. "W", "A", "S", "D" from the keyboard may also use Up, Left, Down, Right, "Z" can be used for zoom in, and "X" is used for zoom out. Toggle Traffic from the Sim Controls section will toggle yellow cars at 0%, 20%, and 50% different spawn rates. Toggle Graph will show up each intersection's name on the graph.

The computation of the hourly edge weights requires (1) hourly traffic volume data, (2) hourly traffic density data, and (3) road segment lengths. Since (3) is static, this can be easily obtained from the road network structure. Although access to (1) is relatively scarce, we found that the New York State Department of Transportation (NYSDOT) reports hourly traffic volume counts for a fraction of road segments in Manhattan, New York City ("HDSB Raw Traffic Data"). Lastly, (2) is not easily recorded, so we did not find any public data for traffic density; therefore, we set traffic density to be constant, and focus on strictly working with traffic volume counts.

Chapter 4 Data Collection and Processing

This section details the preprocessing of the dataset, including data extraction, generation, and the analysis of possible approaches. While there is limited data reported online for hourly traffic, the New York State Department of Transportation database does have information for a small subsection of Manhattan streets. There is also a geographical database of the world available, called OpenStreetMap, that is being used for the detailed geographical elements of this project.

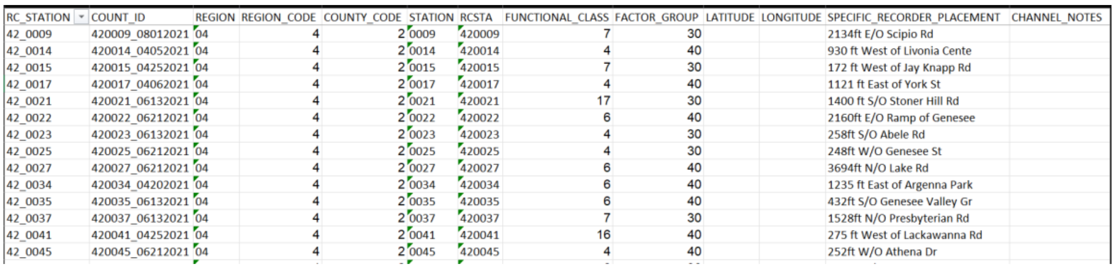
Initially the team encountered the problem of limited reported online data. We discussed three options for data generation in order to fill this gap. The team's first approach was based on the hourly high and low traffic count averages, calculated for any point of interest by a research team at the University of Virginia, as shown in the figure below. Our team would generate sets of data for each day's entry by randomizing a point between the calculated high and low points for every hour. This data would then be sent to the machine learning algorithm.



Another approach the team tried was generating Manhattan data based on the patterns of other similar US cities. Arizona's Maricopa County traffic data is similar to Manhattan in that both urban regions are laid out in a grid. For similar traffic counts, we can look to cities such as San Francisco or Washington D.C.

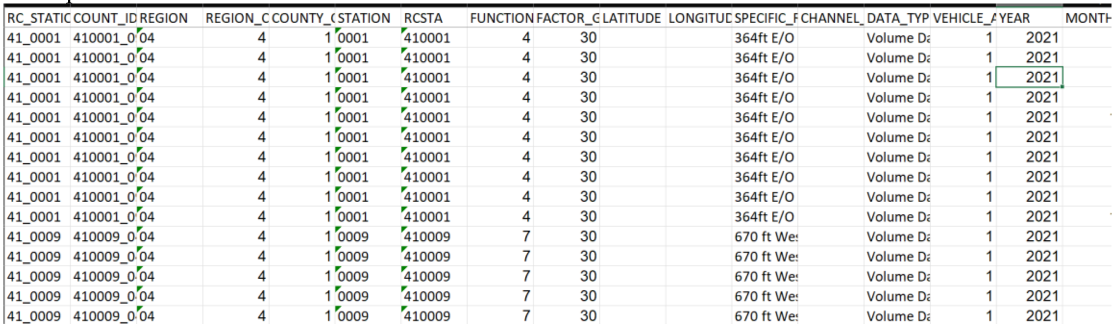
A third approach we considered was extrapolating the existing Manhattan traffic data to cover all hours of the day. Fortunately, the team was eventually able to find sufficiently reported data for Manhattan in the NYSDOT database. Reported data consists of volume traffic counts over fifteen minute intervals for approximately 300 traffic stations around the city, and specifically the borough of Manhattan. Reported data also includes the number of lanes, the street direction, the speed limit, and the street names. The team's data comes from the NYSDOT "HDSB Raw Traffic Data" and combines the reported "SC Speed Data 2021" with the "SC Volume Data 2021", and the "Historic Traffic Data 1977 - 2020." The data does have its limitations, as noted in the NYSDOT Traffic Volume Report. Each traffic count was required to be taken on a weekday, which therefore excludes weekends in these calculations. Traffic counts were also not permitted to be taken before or after a national holiday, thus making our data insufficient for holiday traffic and other variance.

Excerpt from "SC Speed Data 2021":



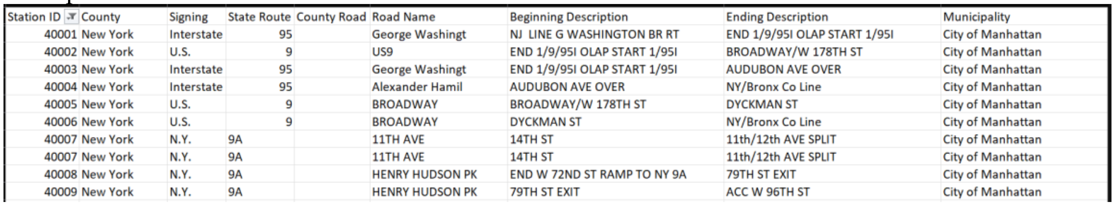
RC_STATION	COUNT_ID	REGION	REGION_CODE	COUNTY_CODE	STATION	RCSTA	FUNCTIONAL_CLASS	FACTOR_GROUP	LATITUDE	LONGITUDE	SPECIFIC_RECORDER_PLACEMENT	CHANNEL_NOTES
42_0009	420009_08012021	04	4	2	0009	420009	7	30			2134ft E/O Scipio Rd	
42_0014	420014_04052021	04	4	2	0014	420014	4	40			930 ft West of Livonia Cente	
42_0015	420015_04252021	04	4	2	0015	420015	7	30			172 ft West of Jay Knapp Rd	
42_0017	420017_04062021	04	4	2	0017	420017	4	40			1121 ft East of York St	
42_0021	420021_06132021	04	4	2	0021	420021	17	30			1400 ft S/O Stoner Hill Rd	
42_0022	420022_06212021	04	4	2	0022	420022	6	40			2160ft E/O Ramp of Genesee	
42_0023	420023_06132021	04	4	2	0023	420023	4	30			258ft S/O Abele Rd	
42_0025	420025_06212021	04	4	2	0025	420025	4	30			248ft W/O Genesee St	
42_0027	420027_06212021	04	4	2	0027	420027	6	40			3694ft N/O Lake Rd	
42_0034	420034_04202021	04	4	2	0034	420034	6	40			1235 ft East of Argenna Park	
42_0035	420035_06132021	04	4	2	0035	420035	6	40			432ft S/O Genesee Valley Gr	
42_0037	420037_06132021	04	4	2	0037	420037	7	30			1528ft N/O Presbyterian Rd	
42_0041	420041_04252021	04	4	2	0041	420041	16	40			275 ft West of Lackawanna Rd	
42_0045	420045_06212021	04	4	2	0045	420045	4	40			252ft W/O Athena Dr	

Excerpt from "SC Volume Data 2021":



RC_STATIC	COUNT_ID	REGION	REGION_C	COUNTY_(	STATION	RCSTA	FUNCTION	FACTOR_G	LATITUDE	LONGITUD	SPECIFIC_F	CHANNEL_	DATA_TYP	VEHICLE_A	YEAR	MONTH
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0001	410001_0	04	4	1	0001	410001	4	30			364ft E/O		Volume Da	1	2021	
41_0009	410009_0	04	4	1	0009	410009	7	30			670 ft Wes		Volume Da	1	2021	
41_0009	410009_0	04	4	1	0009	410009	7	30			670 ft Wes		Volume Da	1	2021	
41_0009	410009_0	04	4	1	0009	410009	7	30			670 ft Wes		Volume Da	1	2021	
41_0009	410009_0	04	4	1	0009	410009	7	30			670 ft Wes		Volume Da	1	2021	
41_0009	410009_0	04	4	1	0009	410009	7	30			670 ft Wes		Volume Da	1	2021	

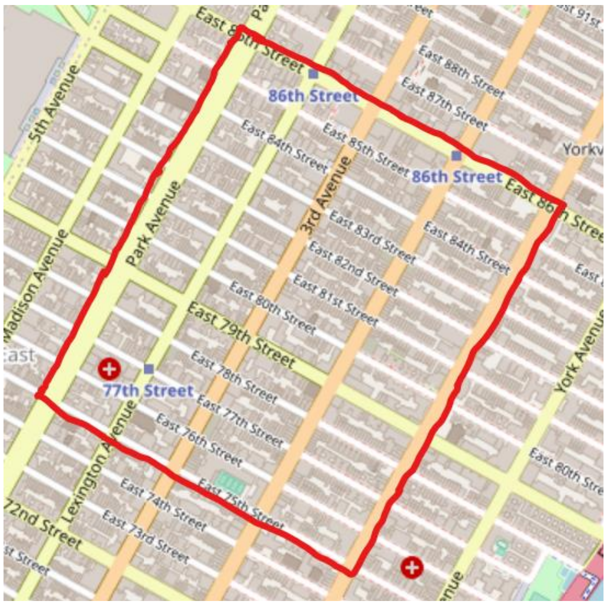
Excerpt from "Historic Traffic Data 1977-1920":



Station ID	County	Signing	State Route	County Road	Road Name	Beginning Description	Ending Description	Municipality
40001	New York	Interstate	95		George Washingt	NJ LINE G WASHINGTON BR RT	END 1/9/95I OLAP START 1/95I	City of Manhattan
40002	New York	U.S.	9		US9	END 1/9/95I OLAP START 1/95I	BROADWAY/W 178TH ST	City of Manhattan
40003	New York	Interstate	95		George Washingt	END 1/9/95I OLAP START 1/95I	AUDUBON AVE OVER	City of Manhattan
40004	New York	Interstate	95		Alexander Hamil	AUDUBON AVE OVER	NY/Bronx Co Line	City of Manhattan
40005	New York	U.S.	9		BROADWAY	BROADWAY/W 178TH ST	DYCKMAN ST	City of Manhattan
40006	New York	U.S.	9		BROADWAY	DYCKMAN ST	NY/Bronx Co Line	City of Manhattan
40007	New York	N.Y.	9A		11TH AVE	14TH ST	11th/12th AVE SPLIT	City of Manhattan
40007	New York	N.Y.	9A		11TH AVE	14TH ST	11th/12th AVE SPLIT	City of Manhattan
40008	New York	N.Y.	9A		HENRY HUDSON PK	END W 72ND ST RAMP TO NY 9A	79TH ST EXIT	City of Manhattan
40009	New York	N.Y.	9A		HENRY HUDSON PK	79TH ST EXIT	ACC W 96TH ST	City of Manhattan

Volume Data is exclusively captured in the "SC Volume Data 2021" file but it does not give speed limit information or provide any previous year's comparison. Thus the files must be used together to draw an accurate representation of Manhattan traffic.
Our team decided that these three approaches work best in combination. The code and algorithms of the project were developed on Arizona State University's VIPLE simulator, a student learning platform. Showing each of these separate approaches, along with the combined results, will be beneficial to future students. The implementation of all the approaches will help students learn about variance in algorithms and will help them to discover optimal solutions for future projects.
Arizona State University's VIPLE traffic simulator is laid out in a grid, similar to the geographical street patterns of Manhattan. Thus, a section of the Upper East Side has been taken to fit the model closely. The streets chosen range from 75th Street to 86th street and 1st Avenue to Park Avenue. After data generation was completed for these streets, the data was sent to the machine learning model, to help predict the traffic conditions of Manhattan as a whole.

Manhattan section chosen for data analysis- (East 75th St to East 86th St and 1st Ave to Park Ave):

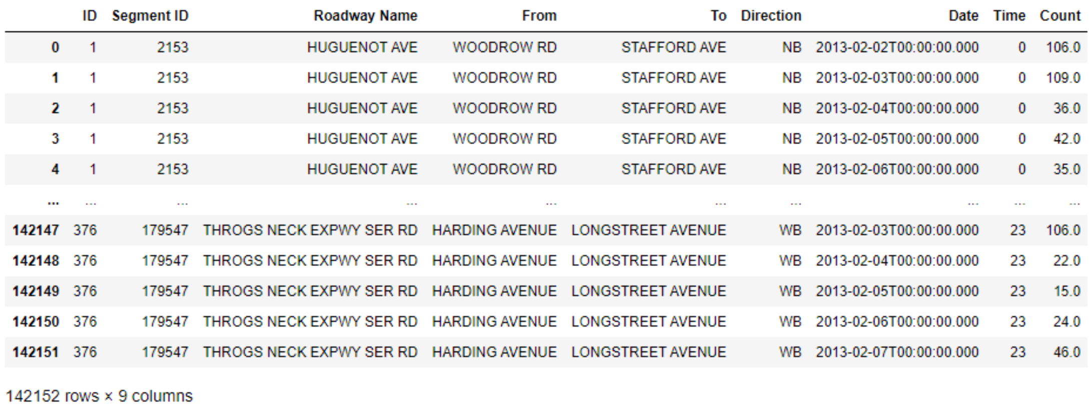


One challenge with this approach is that taking street data from a self-contained subsection does not paint a broad picture of the city as a whole. For example, nightlife hotspots may not be as present in the Upper East Side selected data as they are in other neighborhoods. Another approach that would help to incorporate a more broad view of the city is to pull streets from different areas and combine them into a simulated graph. However, this method has its own set of limitations, including possibly inconsistent data, since each subsection of Manhattan is vastly different. In the current project, the team will be working exclusively with streets from the Upper East Side of Manhattan.

Chapter 5: Machine Learning Algorithms

Machine learning and artificial intelligence are both powerful tools in computational programming as they allow you to process large amounts of data. In the case of traffic prediction, there are many different ML and AI algorithms that can be used to help make predictions based on past data. For our simulation, we will be using past traffic data that consists of road segments and the volume of cars that was counted on each street. From this model, we want to be able to predict the traffic patterns for the streets on our unity map by using the data provided.

Our dataset, found on Kaggle [3] provides us with many different streets and the amount of cars counted on the segment in a given hour. Below is the DataFrame that was generated from the CSV file containing all of the data.



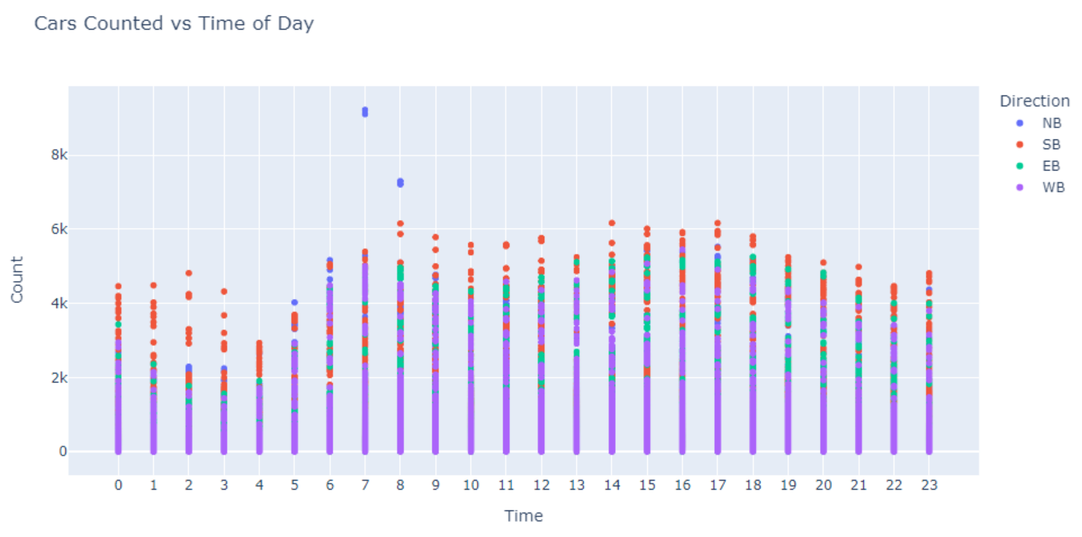
	ID	Segment ID	Roadway Name	From	To	Direction	Date	Time	Count
0	1	2153	HUGUENOT AVE	WOODROW RD	STAFFORD AVE	NB	2013-02-02T00:00:00.000	0	106.0
1	1	2153	HUGUENOT AVE	WOODROW RD	STAFFORD AVE	NB	2013-02-03T00:00:00.000	0	109.0
2	1	2153	HUGUENOT AVE	WOODROW RD	STAFFORD AVE	NB	2013-02-04T00:00:00.000	0	36.0
3	1	2153	HUGUENOT AVE	WOODROW RD	STAFFORD AVE	NB	2013-02-05T00:00:00.000	0	42.0
4	1	2153	HUGUENOT AVE	WOODROW RD	STAFFORD AVE	NB	2013-02-06T00:00:00.000	0	35.0
...	...	...	...	...	...	...	...	...	...
142147	376	179547	THROGS NECK EXPWY SER RD	HARDING AVENUE	LONGSTREET AVENUE	WB	2013-02-03T00:00:00.000	23	106.0
142148	376	179547	THROGS NECK EXPWY SER RD	HARDING AVENUE	LONGSTREET AVENUE	WB	2013-02-04T00:00:00.000	23	22.0
142149	376	179547	THROGS NECK EXPWY SER RD	HARDING AVENUE	LONGSTREET AVENUE	WB	2013-02-05T00:00:00.000	23	15.0
142150	376	179547	THROGS NECK EXPWY SER RD	HARDING AVENUE	LONGSTREET AVENUE	WB	2013-02-06T00:00:00.000	23	24.0
142151	376	179547	THROGS NECK EXPWY SER RD	HARDING AVENUE	LONGSTREET AVENUE	WB	2013-02-07T00:00:00.000	23	46.0

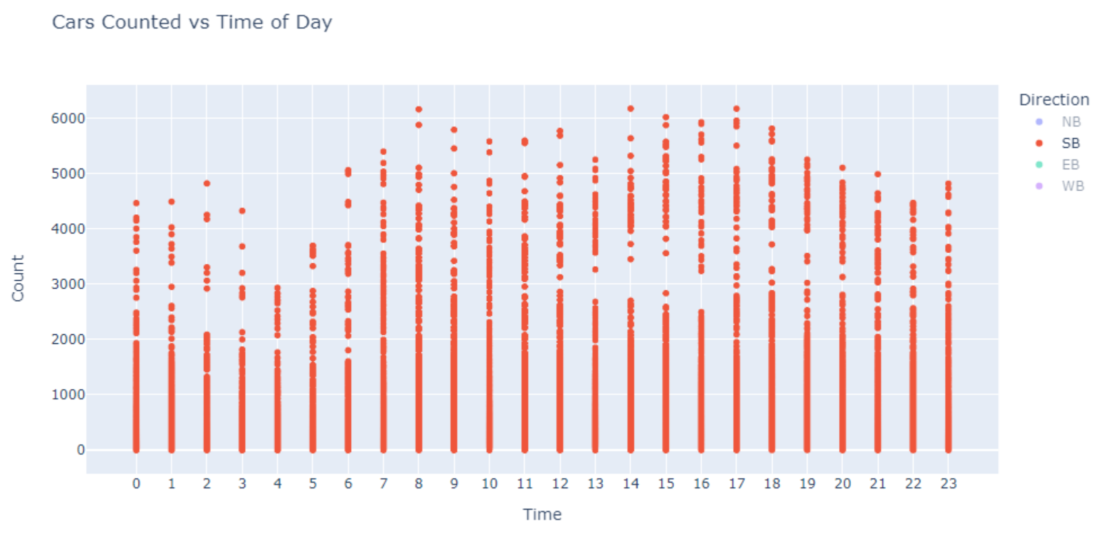
142152 rows × 9 columns

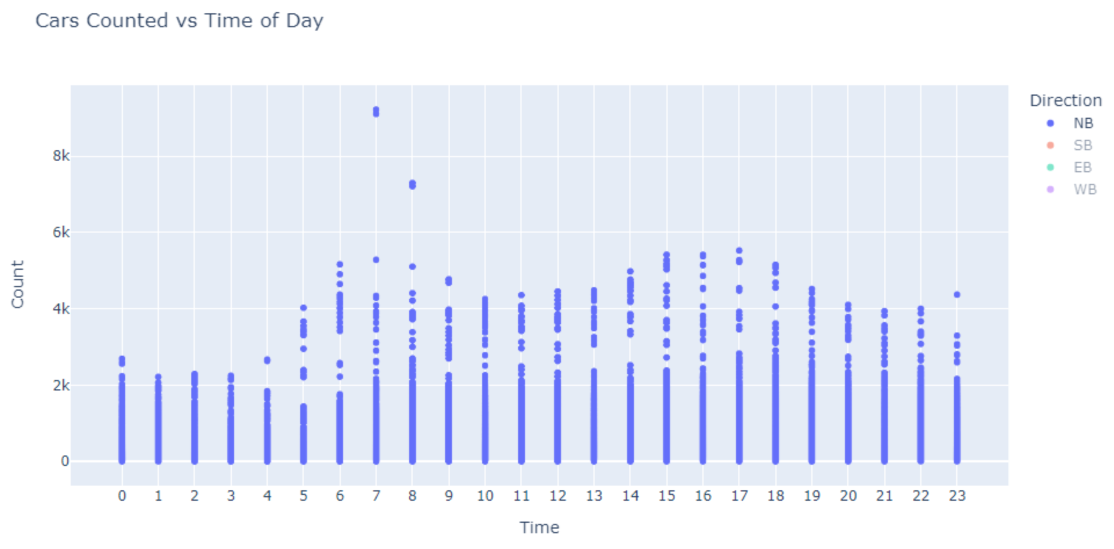
As you can see, there is an extremely large amount of data that is contained in this DataFrame. While this might be overwhelming, it can be simply explained. The *ID* column contains a unique identifier for each road segment. These are similar to *Segment ID* but begin at 1 so we can see how many unique road segments there are. The *Roadway Name* represents the name of the road and the *From* and *To* columns tell us where the segment of the roadway is. While the *Direction* and *Date* columns may be self-explanatory with "NB" meaning "North-Bound", the *Time* and *Count* columns might need some explaining. The Data was collected in hourly increments, such as 2:00-3:00PM. This type of data is categorical and to visualize hourly data, we converted *Time* to represent the number of hours since the day started (ex: 4:00-5:00 AM would be 4). Lastly, *Count* represents the amount of cars that drove by the sensors that were placed along the side of the road to collect the data.

When first working with data, it is important to understand how each feature might contribute to the problem of traffic congestion. In order to better understand our data, it might be useful to plot parts of it to see if there are any patterns that stand out immediately. Immediately, one might think that time and traffic congestion may have a present relationship. Below we can see a plot based on cars on the road and time of day.

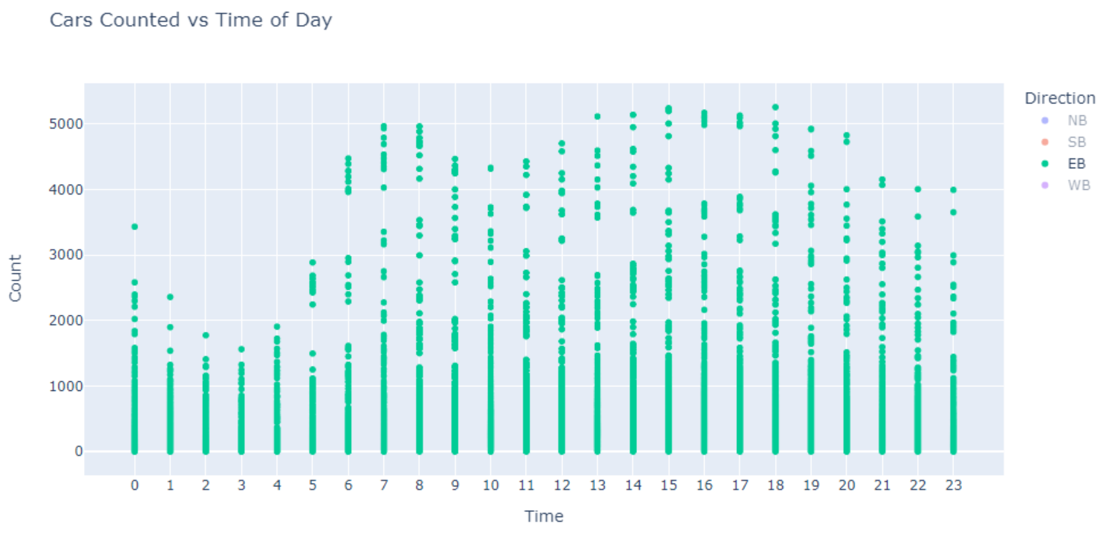
While it may be difficult to see, this plot seems to agree with the sentiment that traffic seems to be lighter during non-business hours as we can see a dip in the amount of cars on the road from 1:00-6:00 AM. This leads us to believe that while time of day may contribute to traffic congestion, it likely is not the whole story that can be concluded from our data. Below, we will see plots of north and south bound traffic to see if they tell a different story.

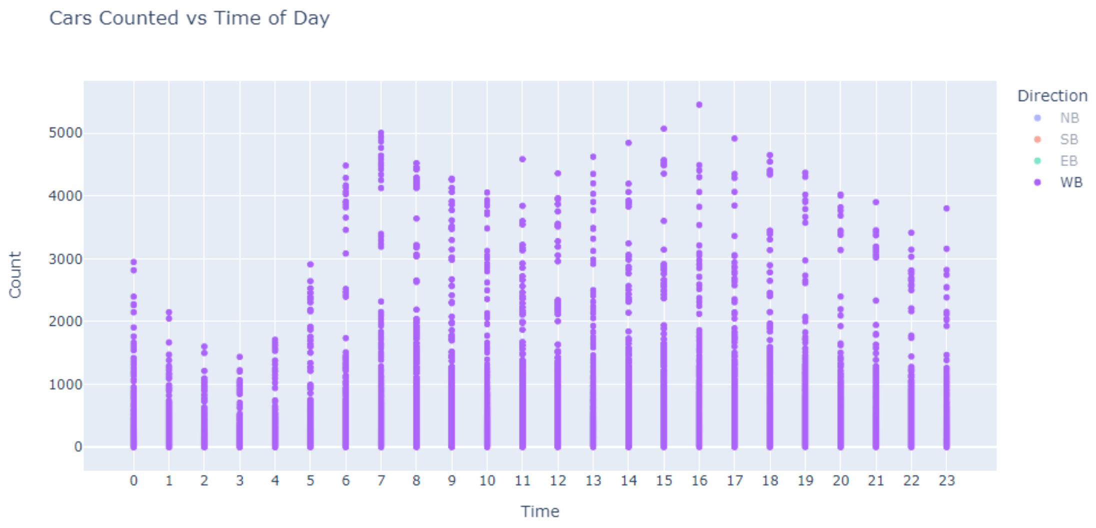






The beginning of an interesting pattern begins to emerge after looking at the north and south bound traffic. During business hours, we can see a gap in the road traffic form at around 3K cars with most road segments being either well above or well below 3K cars in an hour. Let's see if the pattern continues for east and west traffic.





For east and west traffic, it appears that there are far fewer road segments that are above the 3K threshold that appeared for north and south traffic. This pattern is something that can contribute to a traffic prediction based on direction and time.

There are many different algorithms, such as Neural Networks, Naive Bayes and K-Nearest-Neighbors, that can be used to make accurate predictions based on previously measured data. While powerful, most of these algorithms do not quite fit our situation for a traffic simulator for multiple reasons. Neural Networks are commonly used to process large amounts of data, which we have, but often take lots of time to be trained as well as computational to do so. Neural networks are also often used with datasets that have a large amount of features that can contribute to a prediction, which is something we do not have. Naive Bayesian networks are another option that can be used for prediction but are often used with predicting events based on other events rather than predicting an amount of traffic that might appear on a road. K-Nearest-Neighbors (KNN) is another prediction algorithm that is used for pattern recognition. While KNN is often used for classification problems, it can be also used for regression analysis which can be useful for traffic prediction. Simply put, KNN is a distance-based regression model that makes predictions for unknown elements based on their surroundings. A simple example can be found in [11].

When predicting the amount of traffic that may lie ahead at a light, it is important to use both of the features that we have provided to us by the dataset. These, being time and direction, can be used with KNN to make a prediction regarding traffic. With KNN, it is important to determine how distance from each data point is calculated. Distance algorithms such as *Manhattan* and *Euclidean* distance are commonly used, but *Hamming* distance could be also used if you want to emphasize data points in the same category (Hamming, Euclidean and Manhattan are explained in the link above).

In order to construct an algorithm that fits our dataset, we wanted to build onto a preexisting algorithm such as KNN by adding the patterns that we discussed above. To add an emphasis of direction to the algorithm, current direction as well as potential traffic must be taken into consideration by including potential traffic that is traveling perpendicular to your route as it has a probability of turning onto your path. To incorporate time of day, we can modify KNN algorithm to work with both hamming distance and manhattan distance.

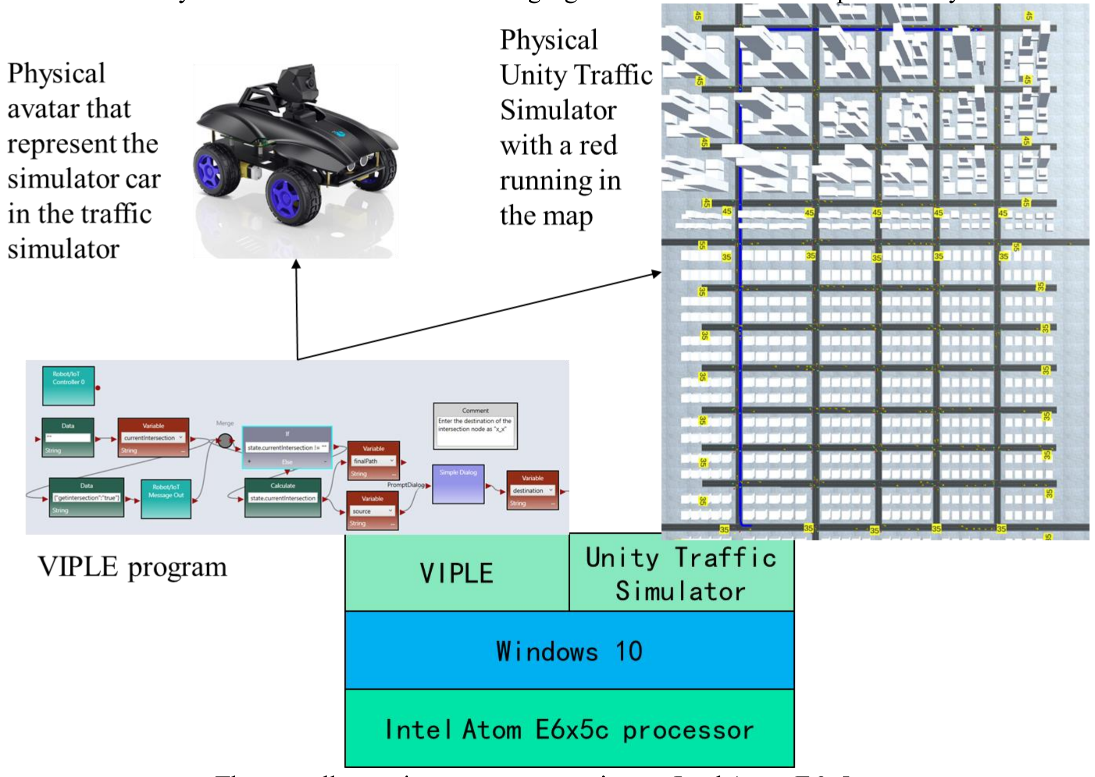
The algorithm works by taking the 3 hour average of the traffic at time, t, for any of the 4 given directions. Then, the algorithm will combine this with any oncoming traffic that might be from the other two perpendicular directions that may turn (left or right) onto your direction. This is similar to a running average but uses the classification concept from KNN to take from like directions.

Chapter 6 Experimentation and Simulation

This chapter will present the experiments conducted under this project. It includes the Intel Atom E6x5c processor platform running VIPLE traffic simulation and a robot that runs as a physical avatar of the simulated robot.

6.1 Overall Experiment System

The Intel GNS-V40 processor board is installed with a Windows 10 operating system that runs VIPLE software and Unity Traffic simulator. The following figure shows the overall experiment system.



Physical avatar that represent the simulator car in the traffic simulator
Physical Unity Traffic Simulator with a red running in the map
VIPLE program
VIPLE
Unity Traffic Simulator
Windows 10
Intel Atom E6x5c processor

The overall experiment system running on Intel Atom E6x5c

6.2 VIPLE Program Controlling the Traffic Simulation

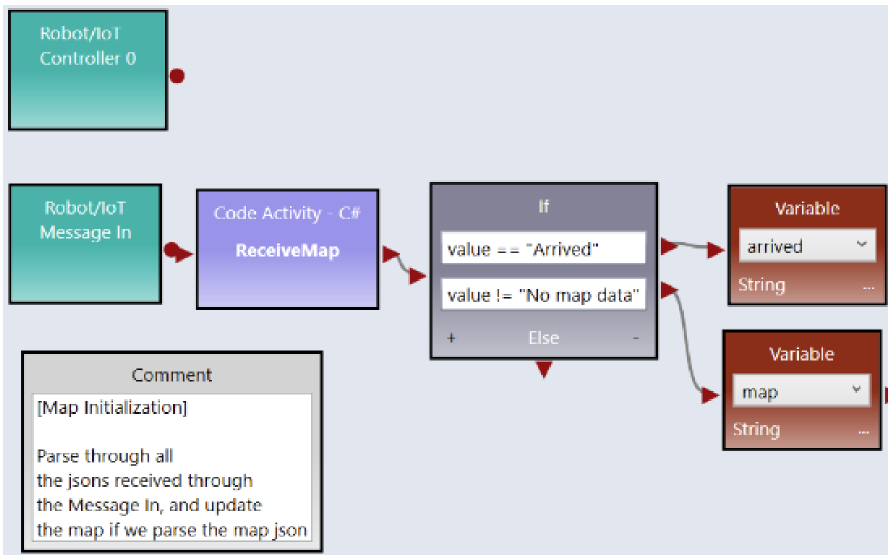
This dynamic Dijkstra algorithm is the primary way to navigate the simulation. It will be triggered every time the car approaches the new intersection, and this is the reason it is named "Dynamic". Based on the current traffic situation obtained from the model at different intersections, the algorithm filters the best cost path so it could keep the schedule updated.

6.2.1 Simulator Initialization

The simulation program starts with the time-frame identification which is an important index to obtain the proper traffic pattern.

6.2.2 Receive Map Information from the Simulator

The VIPLE receives the real-time map information from the simulator via the setup, and the map data will be encoded as a JSON object for the communication between the VIPLE and the simulator.



[Map Initialization]

Parse through all
the jsons received through
the Message In, and update
the map if we parse the map json

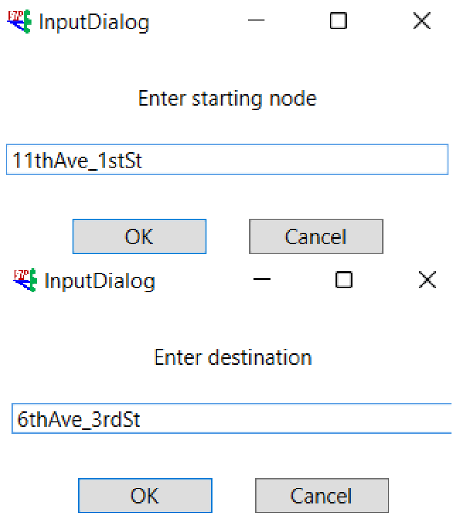
```
// ----------------------- JSON Structure ----------------------- //
 public class MapGraph
 {
     public Dictionary<string, string[]> AdjacencyList { get; set; }
     public Dictionary<string, Position> Nodes { get; set; }
     public Dictionary<string, GraphEdge> Edges { get; set; }
 }

 public class Position
 {
     public double WorldPosX { get; set; }
     public double WorldPosY { get; set; }
     public double WorldPosZ { get; set; }
 }

 public class GraphEdge
 {
     public string From { get; set; }
     public string To { get; set; }
     public float Weight { get; set; }
 }
```

6.2.3 User Interaction

The user is able to assign the departure and arrival location for the program via the pop-up dialog. Respectively, in the VIPLE, the departure location is sent to the variable "*currentIntersection*" and is stored in the variable "*source*", and the arrival location is stored in the variable "*destination*".


InputDialog

Enter starting node

11thAve_1stSt

OK Cancel

InputDialog

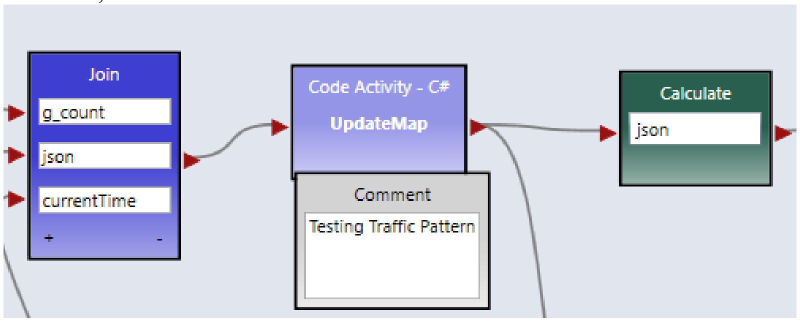
Enter destination

6thAve_3rdSt

OK Cancel

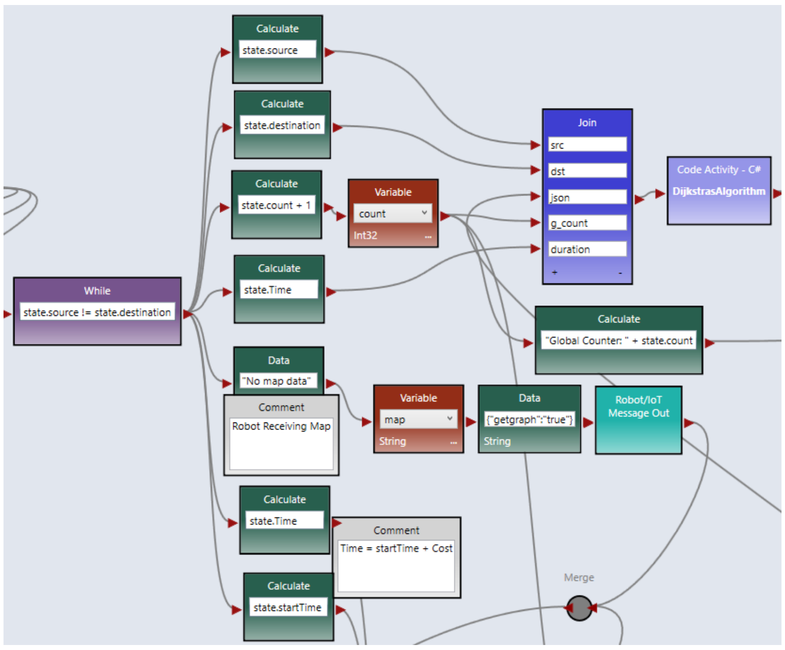
6.2.4 ML Model Prediction

The C# code "UpdateMap" worked as the prediction model in which there is a traffic model stored inside of the block. It took three variables as input and will output the new schedule for the robot- Including the variable "g_count" which records the workflow globally, the variable "json" which stores the current schedule, and the variable "currentTime" which stores the current cost of time.

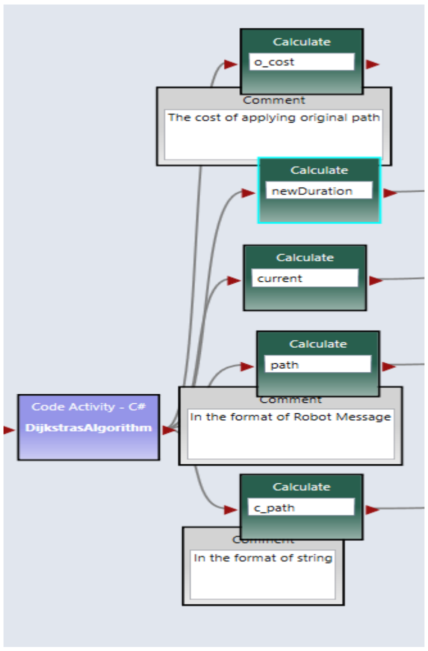


6.2.5 While-Loop & Dijkstra Algorithm

The program applied the Dijkstra algorithm to generate the shortest path and a while-loop integrated with it to achieve the dynamic workflow. In the case we are running, the algorithm takes five variables as the input - the variable "*src*" stored the starting location entered by the user, and the variable "*dst*" worked a similar way but its stored value indicated the destination. Then, the map in terms of JSON goes through by the variable "*json*". In the end, the last two variables record data on the global scope, respectively, the value in the variable "*g_count*" record the aggregated amount of Dijkstra algorithm calculation, and the aggregated cost of time stored in variable "*duration*".



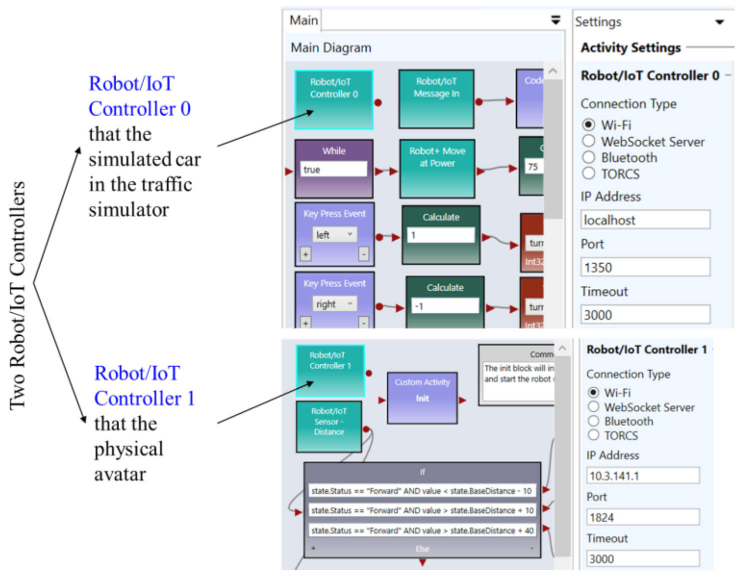
On the output side, the variable "*Time*" and the variable "*source*" saved the new cost of the travel and the current location, respectively. And the shortest path will be interpreted in two different formats - the shortest-path-included schedule in JSON is transported via the variable "*path*", and the variable "*c_path*" keeps the same content but in the format of a string.



6.3 Middleware Implementing the Controlled Physical Avatar

In our current experiment, we will run the physical avatar robot without drawing the map. In other words, the physical robot will mimic the driving behaviors of the simulated robot in terms of driving forward, turning left, and turning right. The robot will be running on a surface with grids that match the traffic simulation map.

In the VIPLE program, we need to create two robots, with one controlling the simulated robot car and the other controlling the physical avatar, as shown in Figure 6.2.



state.Status == "Forward" AND value < state.BaseDistance - 10
state.Status == "Forward" AND value > state.BaseDistance + 10
state.Status == "Forward" AND value > state.BaseDistance + 40

Settings

Activity Settings

Robot/IoT Controller 0

Connection Type
- [✓ Wi-Fi]
- WebSocket Server
- Bluetooth
- TORCS

IP Address: localhost

Port: 1350

Timeout: 3000

Robot/IoT Controller 1

Connection Type
- [✓ Wi-Fi]
- WebSocket Server
- Bluetooth
- TORCS

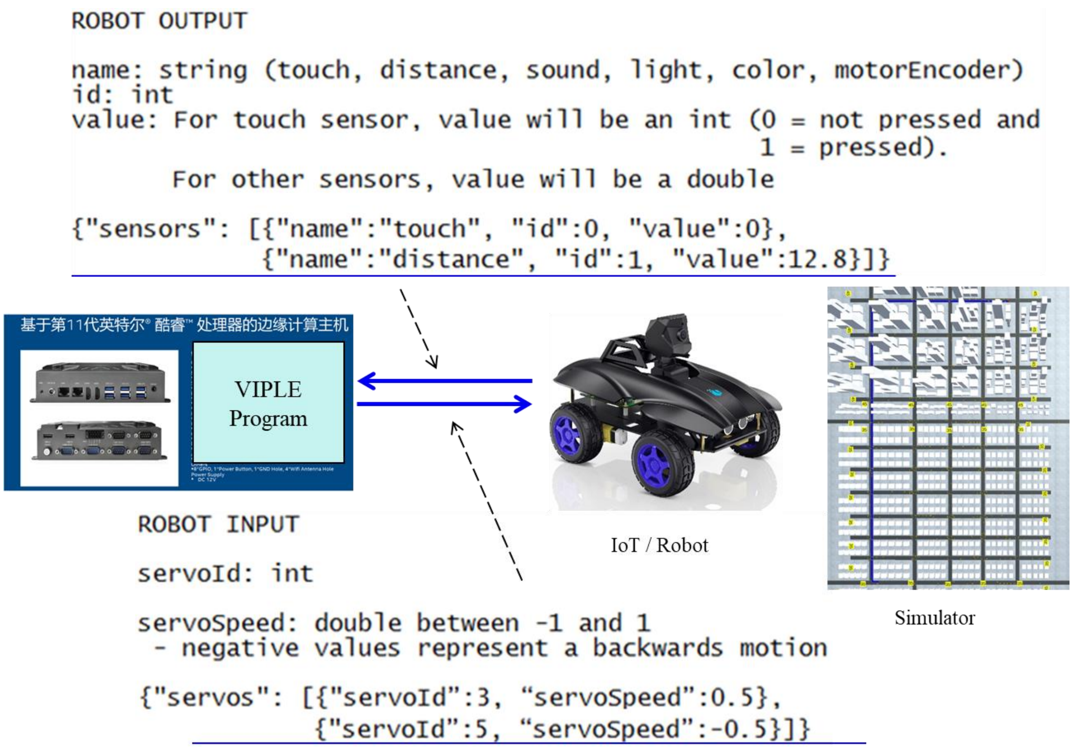
IP Address: 10.3.141.1

Port: 1824

Timeout: 3000

Figure 6.2 VIPLE program with two Robot/IoT controllers and logics for the simulated robot and the physical avatar, respectively.

The two robots can be programmed independently. The simulated robot is the main robot that interacts with the VIPLE program. For the physical avatar robot, no sensors and no data will be sent from the robot to VIPLE. VIPLE sends the same driving signals to both simulated and physical robots.
The communication between the VIPLE and the simulated robot and physical robot uses the same JSON object, as shown in Figure 6.3. Any robot with an open architecture can install a middleware on the robot processor board to communicate with VIPLE. We have implemented the physical robots using any Intel-based processor, such as Galileo, Edison, and GNS-V40.



```
ROBOT OUTPUT

name: string (touch, distance, sound, light, color, motorEncoder)
id: int
value: For touch sensor, value will be an int (0 = not pressed and
                                               1 = pressed).
       For other sensors, value will be a double

{"sensors": [{"name":"touch", "id":0, "value":0},
             {"name":"distance", "id":1, "value":12.8}]}
```

```
ROBOT INPUT

servoId: int

servoSpeed: double between -1 and 1
 - negative values represent a backwards motion

{"servos": [{"servoId":3, “servoSpeed":0.5},
            {"servoId":5, “servoSpeed":-0.5}]}
```

Figure 6.3 The standard communication data format between VIPLE and robots

Chapter 7 Conclusion

The goal of this project is to create an embedded system design that would enable a simulator and a test car to be able to use the map and traffic data from Maricopa and Manhattan. The purpose of the simulator is to find the shortest path to navigate from point A to point B using a Machine learning algorithm that would optimize the path calculated using code created in the VIPLE programming environment. The machine learning algorithm will make predictions based on the traffic data and instead of relying on the Dijkstra algorithm to recalculate the route at every intersection, will reduce the amount of calculations needed to ensure the shortest path. Currently we have plans to improve the runtime and efficiency of our code. We also intend to improve the accuracy of the predictions done by the machine learning algorithm against the Manhattan traffic data and that we plan to design more educational examples of this that would be used to explain things such as graph flow or social media centrality.

References

[1] *2014 Traffic Data Report for New York State*. New York State Department of Transportation. (2014). Retrieved July 18, 2022, from https://www.dot.ny.gov/divisions/engineering/technical-services/hds-respository/NYSDOT_Traffic_Data_Report_2014.pdf

[2] New York State Department of Transportation. (n.d.). *Department of Transportation*. HDSB Raw Traffic Data. Retrieved July 17, 2022, from https://www.dot.ny.gov/divisions/engineering/technical-services/highway-data-services/hdsb

[3] New York traffic dataset: https://www.kaggle.com/datasets/new-york-city/ny-traffic-volume-counts-2012-2013

[4] Zhang, Z., De Luca, G., Archambault, B., Chavez, J., & Rice, B. (2022). Traffic Dataset for Dynamic Routing Algorithm in Traffic Simulation. Journal of Artificial Intelligence and Technology, Vol. 2, Issue 3, July 2022. https://doi.org/10.37965/jait.2022.0106

[5] OpenStreetMap. (n.d.). Retrieved July 17, 2022, from https://www.openstreetmap.org/

[6] Venkatanarayana, Ramkumar & Smith, Brian & Demetsky, Michael. (2008). Quantum-Frequency Algorithm for Automated Identification of Traffic Patterns. Transportation Research Record: Journal of the Transportation Research Board. 2024. 10.3141/2024-02

[7] Yinong Chen, Gennaro De Luca: "VIPLE: Visual IoT/Robotics Programming Language Environment for Computer Science Education" , IPDPS Workshops 2016: 963-971.

[8] Y. Chen, G. De Luca Service-Oriented Computing and System Integration: Software, IoT, Big Data, and AI as Services, 8th edition, Kendall Hunt Publishing, 2022.

[9] Qiang Chen, Yinong Chen, Jinhui Zhu, Gennaro De Luca, Mei Zhang, Ying Guo, "Traffic light and moving object detection for a guide-dog robot", The Journal of Engineering 2020 (13), 675-678.

[10] Yingxu Lai, Yinong Chen, Zenghui Liu, Zhen Yang, Xiulong Li, "On monitoring and predicting mobile network traffic abnormality", Simulation Modelling Practice and Theory Vol. 50, pp. 176-188 (2015).

[11] Aishwarya Singh, A Practical Introduction to K-Nearest Neighbors Algorithm for Regression (with Python code), August 2018, https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/