VIP Smart Cities Technology
Transportation Users

Team Leader: Robe Zhang
Team Members:
Jian Nan (Andy) Huang
Mikhail Vasilevitskiy
Sihao Chen
Zach Gonzalez-Ruskiewicz
Semester: Fall 2017

Abstract

Throughout the course of the fall semester, the transportation users sub team explored the use of two drones, the Parrot Mambo Mission and the Parrot Bebop2, in the modeling of a routing alogrithm. This algorithm will eventually be implemented to route data collecting drones through NYC. After conducting experimental weight testing to understand the capabilities of each particular drone, we have researched methods to create drone simulations. We have looked into the simulation paltforms, ROS, Parrot SDK, Parrot Sphinx, and MAV Proxy. In addition, we have continued to develop the ride-sharing web-based simulation.

Introduction

Drone Simulation

The overarching goal is to create a fleet of autonomous drones that are capable of performing data collection with the sensors, charging accessories, navigation accessories, and microprocessor boards such as the audino or the raspberry pi they will carry. In the future, we plan to work alongside the other sub teams under VIP Smart Cities Technology in order to complete this task. The cities sub-team is working on the sensors to measure various environmental data. The vehicles sub-team is working on LiDAR to help the drone avoid obstacles in navigation. The infrastructure sub-team is working on inductive charging stations and attachable solar power charging to sustain the drone. Our sub-team is responsible for looking into adaptive algorithms to path particular drones to data collection sites while managing the drone's battery life. In particular, we looked into the various flight patterns and paths that a drone can take. We have looked into two varieties of drones from Parrot[1] to test for flight control. The first drone is the smaller Parrot Mambo Mission[2] meant for recreational use. The other drone is the larger Parrot Bebop2[3] designed to take high quality arial pictures with its embedded camera. We have looked into various software to create drone simulations. Finally, preliminary tests were run on the two Parrot drones to assess their real world capabilities.

Ride-sharing Web-based Simulation

Continuing with our work from the previous semester, Spring 2017, we have continued to upgrade the ride-sharing web-based simulation[4]. The goal for the ride-sharing web-based simulation is to provide a way to test and analyze scheduling algorithms with a fleet of virtual autonomous vehicles ready to pick up and drop off customers requesting rides. A major metropolis ideally would implement a ridesharing service alongside a bike sharing service as the primary means of public transportation. In this system, less stress would be placed on the an expensive and outdated underground rail system which only serves certain areas of the city. Moreover, with the reduction of privately own vehicles, traffic would be less congested and human related accidents such as driving under the influence, and texting and driving will be less likely to occur. As a part of the Transportation Users Sub-Team our goal is improve public transportation starting with urban environments. So far, the ride-sharing web-based simulation is concentrated on New York City. The ride-sharing web-based simulation is able to take multiple request and route each request to the respective vehicles.

Literature Review

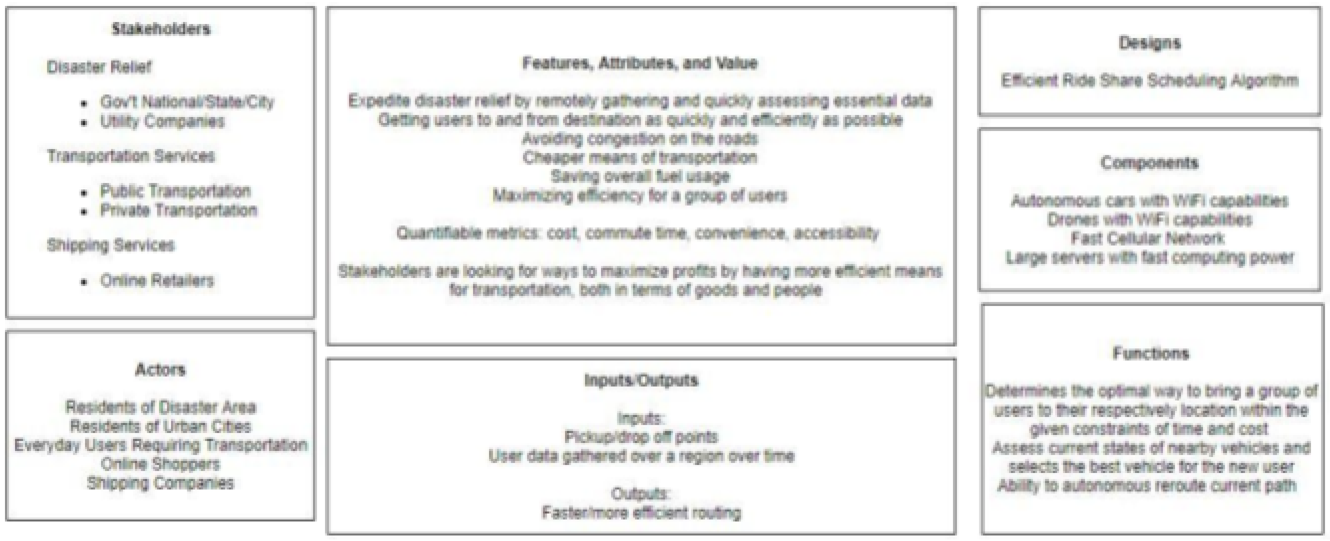
Uber, a ride service company, has been under scrutiny for their shady business practices. In the article, “Uber Says Hackers Stole Personal Data Of 57 Million Users[5]”, written by Aarti Shahani, she reveals an attempted cover up by the company on a data breach that occured in late 2016. Uber kept the data breach that exposed data from over 57 million users a secret for almost a year. The data breach was only recently revealed to the public in November 2017. Aarti reports that Uber has paid the hackers $100,000 to delete the stolen data and to remain silent about it. As a result of the data breach, data, both sensitive and insensitive have been exposed to the blackmarket. The names, driver license numbers of drivers as well as names, email addresses, phone numbers of customers have been lingering in the blackmarket. Such data can lead to identity theft. Uber has recently proposed to offer free credit monitoring to help users deal with identity theft following the example of companies such as Equifax and Yahoo who recently had customer data leaked. The only issue with Uber providing the service now is the large gap between the leak and the monitoring service. It has almost been a year and the damage is already done.

Beside this case Uber is trying to bend the laws for its profit motives. In the United Kingdom, Uber claims that drivers are contractors rather than employees to attempt to counteract the legislative removal of Uber’s private hire operator licensing rights in London[6]. The Traffic for London (TFL) regulates London's taxi and private hire trades is designed to ensure passenger safety. There has been multiple cases of Uber failing to comply with proper regulations in hiring its drivers. Uber neglects proper reporting of criminal offenses, and background checks on medical certificates.

Another major issue with Uber is its wide range of data collection. Uber has raised many privacy violations where it intentionally collects and conceals data without allowing the public know. Uber secretly uses sensors to track driver’s driving performance without notifying drivers about the data collection or releasing data to the driver. Laurel Wamsley reports an article[7] about Uber tracking its users immediately after using its service for a duration up to 5 minutes starting which began in November 2016. As of August 2017, Uber has given the option to the user to share the data. Uber claimed that the secret data collection was part of it service “to improve pickups, drop-offs, consumer service and to enhance safety.”

Methods

We created the design canvas for our rideshare algorithms, mainly to keep us on track and remind us what we want to accomplish for the algorithms. Our objective has not changed throughout the semester since efficiency of the algorithm remained our main concern. As of right now, efficiency is determined by the distance and time between each pickup and drop-off, getting the generated values from the algorithms to meet the the customer's constrained values. In the upcoming semesters, hopefully, we can extend our parameters and incorporate the cost into our algorithm as well. Distance traveled, time, and cost are the main considerations in urban transportations. For example, MTA transportation has thrived on low cost while taxi and Uber are less time consuming and offer a more direct route of travel. However, these means of transportation still often contain serious issues of congestion and delays which our algorithm aims to minimize. This minimzation would certainly give us an edge over the competitors we



Stakeholders

Disaster Relief

- Gov't National/State/City
- Utility Companies

Transportation Services

- Public Transportation
- Private Transportation

Shipping Services

- Online Retailers

Actors

Residents of Disaster Area
Residents of Urban Cities
Everyday Users Requiring Transportation
Online Shoppers
Shipping Companies

Features, Attributes, and Value

Expedite disaster relief by remotely gathering and quickly assessing essential data
Getting users to and from destination as quickly and efficiently as possible
Avoiding congestion on the roads
Cheaper means of transportation
Saving overall fuel usage
Maximizing efficiency for a group of users

Quantifiable metrics: cost, commute time, convenience, accessibility

Stakeholders are looking for ways to maximize profits by having more efficient means for transportation, both in terms of goods and people

Inputs/Outputs

Inputs:
Pickup/drop off points
User data gathered over a region over time

Outputs:
Faster/more efficient routing

Designs

Efficient Ride Share Scheduling Algorithm

Components

Autonomous cars with WiFi capabilities
Drones with WiFi capabilities
Fast Cellular Network
Large servers with fast computing power

Functions

Determines the optimal way to bring a group of users to their respectively location within the given constraints of time and cost
Assess current states of nearby vehicles and selects the best vehicle for the new user
Ability to autonomous reroute current path

Design Canvas

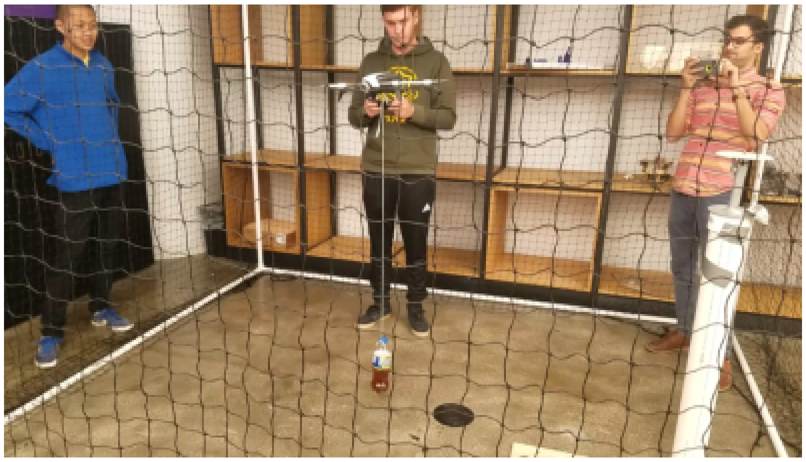
have previously mentioned.

Drone Simulation

We first assess our two options for drones, the Parrot Mambo Mission and the Parrot Bebop2. Utilizing the Parrot Free Flight[8] as the drone controller, and the drone net setup in the MakerSpace, we tested for the capabilities and limitations of each drone. Afterwards we looked at the hardware and software specification for each drone. We have selected to continue with the Parrot Bebop2 drone.

Weight Testing

For the weight testing, we have attached several objects to the drones to test its capabilities. The most prominent object would be a 16 fl oz bottle which is shown above in the picture. We tied the bottle to drone using a string so the drone

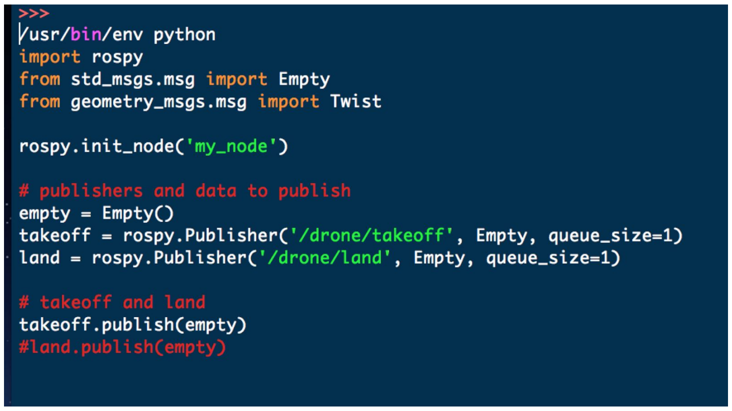


Weight Testing

can lift it up. In the begin, the drone was a little bit shaky due to the swinging of the bottle but the drone did a good job hovering in the air to stabilize the movements. We feel the bottle dragged down the drone from its actual elevation that it is appointed to fly at because of the weight of the bottle. Also, the bottle was directly under gyro sensor, affecting the movement of drone in one way or another. We did several trials to confirm that the drone can carry at most between 200-225g which is not much comparing the sensors that the drone potentially has to carry for collecting data.

ROS

The first simulation that we have considered is ROS Development Studio (ROS)[9]. ROS Development Studio allows for low setup in terms of drone simulation. The ROS Development Studio runs on the web browser and it is not operating system dependent unlike the Parrot Sphinx simulator. ROS also allows for programming across various brands of robot, not limited to Parrot products unlike the Parrot SDK. Step up for the ROS Development Studio is simple. It requires creating an account on theconstructsim.com which we have created a free student account. The student account comes with 2 vCPU or 2 virtual central processing units and no GPU power or graphics processing unit. After creating an account and selecting a drone, the simulation runs automatically. We have tested basic commands to make the drive take off, hover, fly in a direction and land. The code for ROS Development Studio is in Python. The code to launch the drone and land the drone is shown below:



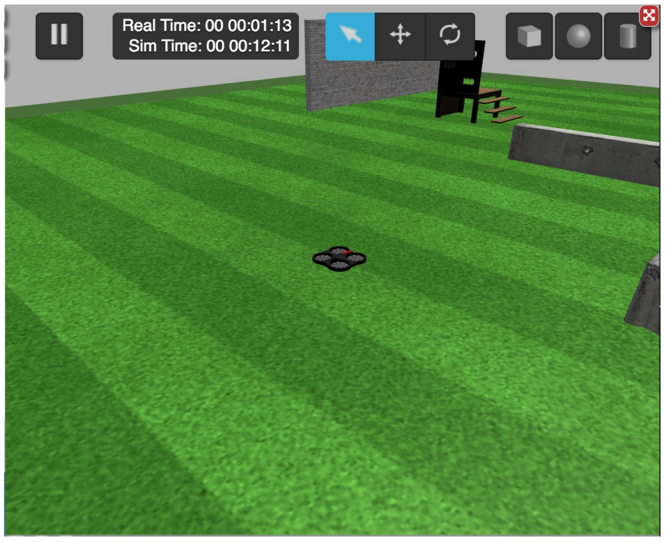
```
>>>
/usr/bin/env python
import rospy
from std_msgs.msg import Empty
from geometry_msgs.msg import Twist

rospy.init_node('my_node')

# publishers and data to publish
empty = Empty()
takeoff = rospy.Publisher('/drone/takeoff', Empty, queue_size=1)
land = rospy.Publisher('/drone/land', Empty, queue_size=1)

# takeoff and land
takeoff.publish(empty)
#land.publish(empty)
```

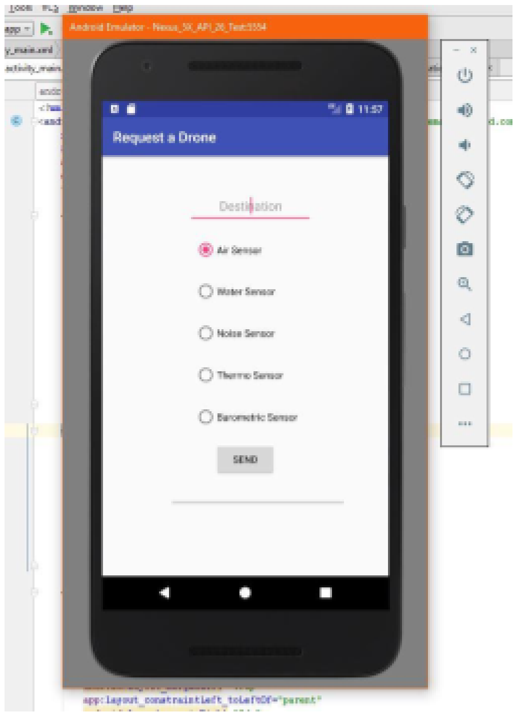
The simulation drone in the ROS Development Studio environment is shown as below:



Parrot Sphinx

We looked into Parrot’s SDK for Android[10] as an alternative way to setup a Parrot Bebop2 drone simulation. The Parrot SDK for Android is written exclusively in Java. As a result, we have to use the Android Studio integrated development environment[11]. Many of the team members did not have experience with Java, so we explored the basics in Java programming. After setting up and exploring Android Studio, we developed some graphical user interface code to get a sense of what type of data we are gathering and how to operate the drones. We followed the Parrot SDK for Android tutorial to get the basic commands to control the drone. Following this step, we looked into Sphinx, a Parrot Drone simulation software[12]. The Sphinx software requires a Linux 64 bits operating system to run. The installation for Sphinx is done by the follow lines of code on terminal:

```
$ echo “deb http://plf.parrot.com/sphinx/binary ‘lsb_release -cs’/”
| sudo tee/etc/apt/sources.list.d/sphinx.list > /dev/null
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
508B1AE5
$ sudo apt-get update
$ sudo apt-get install parrot-sphinx
```



Request Drone GUI

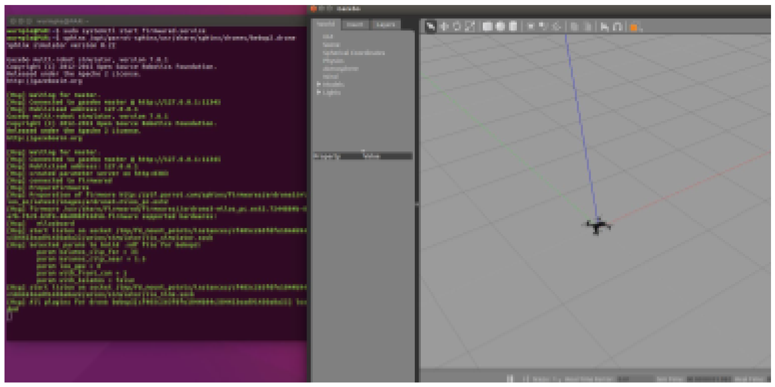
In order to run a Parrot Bebop2 Drone simulation on Sphinx, the following lines of code is run on terminal:

```
$ sudo systemctl start firmwared.service
$ sphinx /opt/parrot-sphinx/usr/share/sphinx/drones/bebop2.drone
```

To broadcast the Parrot Bebop2 Drone simulation to be able to connect to a controller via wifi, the following is run on terminal :

```
$ sphinx --port-forwarding={IPADDRESS}
/opt/parrot-sphinx/usr/share/sphinx/drones/bebop2.drone
```

The Parrot Sphinx simulation for the Bebop2 is shown below:

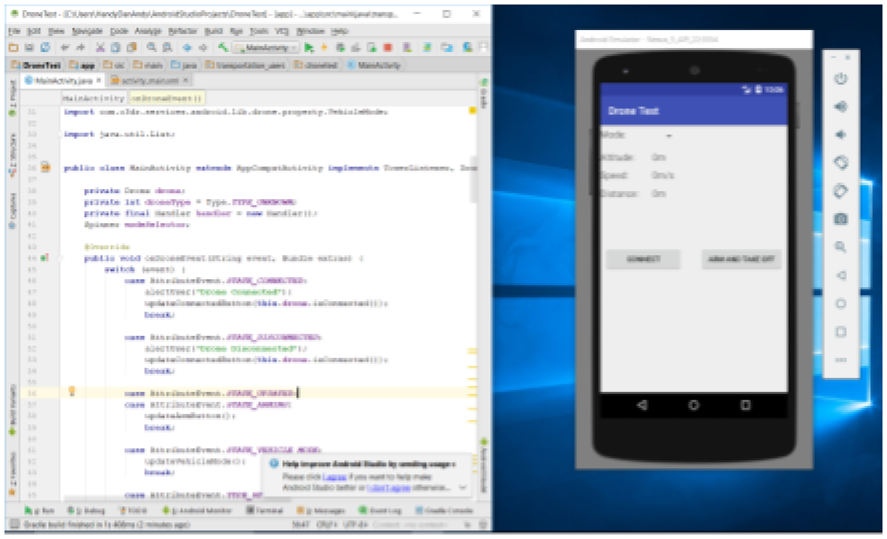


Sphinx Simulation

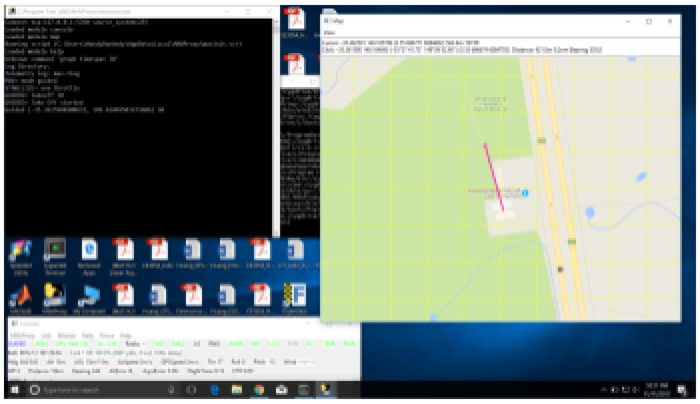
At first, we ran Ubuntu 16.04 off a virtual machine. The issue with running Ubuntu 16.04 in a virtual environment is lack of external wireless connection required to connect to the Android Studio code. We tried the alternative of running Ubuntu 16.04 as the main operating system via dual boot. Running Ubuntu 16.04 as the main operating system allowed for an external wireless connection to be established. We tried to connect the Parrot Free Flight app to the simulation, however the app wasn't able to process the Sphinx simulation. We looked at various forums on Parrot's developer forum without any success.

MAV Proxy

The third method of drone simulation that we considered was using MAV Proxy[13] and Flight Gear[14]. The program MAV Proxy generates a 2D world map, and takes in commands that sets the drone in various modes. The drone will fly in certain speeds and altitudes (based on command inputs) and will show the drone flying toward the desired location. The Flight Gear program will provide a 3D view of the MAV Proxy simulation. In addition to this dual usage of the MAV Proxy and Flight Gear app, an Android app was also written that uses the 3DR Tower app[17]. The 3DR Tower app is used for planning out drone missions; the drone will follow the instructions sent out by the 3DR Tower and reach its destination. In Android Studio[11], we can perform virtual device simulations. This virtual device can install .apk files (apps), which can be downloaded from the web. The virtual device that we used was a Nexus 5, and the 3DR Tower app was installed by downloading the .apk file from the web and dragging it to the virtual device. In the grand scheme of things, the simulation of the drone would follow these steps: the 3DR Tower device takes in user input (start location and end location), sends the user input to the MAV Proxy app, and lastly displays the 3D simulation in Flight Gear.



Android App Simulated on Virtual Device



MAV Proxy Simulation

Ride-sharing Web-based Simulation

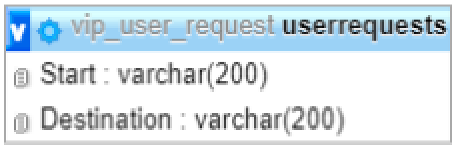
Expanding onto the work from the previous semester, the ride-sharing web simulation code was cleaned up and reorganized. The ride-sharing web-based simulation was originally pushed forward for the Spring Research Expos to be functional, however the organization and the readability of the code was lacking. Within the main file, the similar functions were grouped together and placed within separate script files. In doing so, the readers are easily able to distinguish which parts of the code does what functionality. The entire code base can be viewed on Github[4]. There are also instructions for users to be able to access the simulation and run the simulation for themselves. Besides reorganizing the structure of the code, there were also added features to allow for additional information to be displayed by toggling the space bar on the simulation screen. The user request website has been restricted to allow only one request per section. There is also a plan to upgrade the storage of data from being session based to be connected to the database. Currently, the data is stored locally in the web simulation session. By occasionally updating the database, the simulation can keep its state and load previous states. The current database only temporarily holds a user's request and deletes the data right away. An updated database schema was designed to accommodate for storage of data. Small steps were taken to integrate the code with the new database schema.

Analysis

Due to the code heavy portions of getting a drone simulation running and connected, there wasn't any quantifiable data gathered during this semester pertaining to drones.

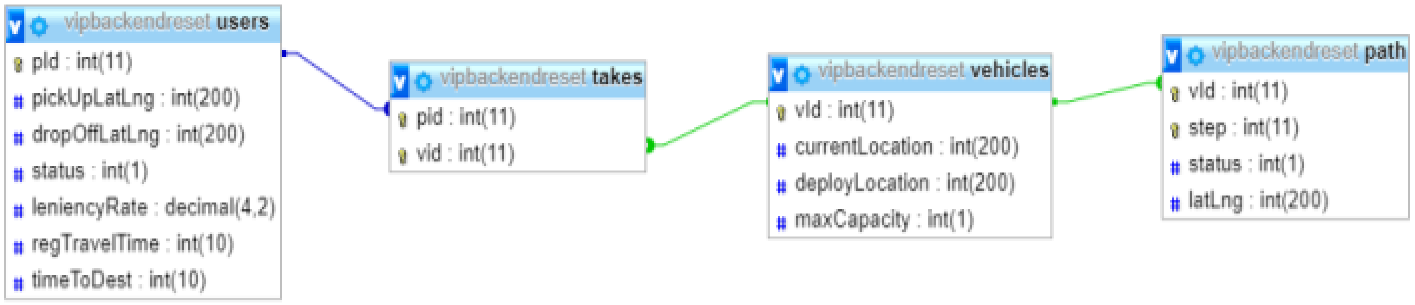
Ride-sharing Web-based Simulation

As for the ride-sharing web-based simulation, the qualitative analysis goes into the creation of the new database schema and anticipating data that might be valuable in the future. The old database only stored information about an individual's pickup and drop off request as displayed below:



vip_user_request **userrequests**
Start : varchar(200)
Destination : varchar(200)

Old Database



New Database Schema

The new database design will allow for states of the simulation to be kept and saved. This feature will allow us to further analyze a predefined set of data multiple times by alternating some of the variables while the rest of the variables remain the same. As the database schema is in third normal form, all of the data and queries for the data will be efficient and effective. The added database server will take stress and memory off of the computer to allow for faster computations. This allows for the creation of a controlled environment to test for the efficiency of algorithms. Perhaps one algorithm works better in a particular case and another in another case. The current algorithm is a variation of the greedy algorithm. The algorithm matches the user to the near available vehicle that is able to pick up the user without affecting the current passengers on the vehicle. The algorithm takes into account the estimated time it regularly takes a user to get from the user's current location to the destination and multiplies that number by 1.5 as a maximum leniency to share a ride with another user. If a user is unable to match with a vehicle, then the user will be placed on a queue until the next available vehicle appears. The assumptions currently taken is that the leniency rate is a constant 1.5. Because all leniency rates are the same, no single user is given higher precedence than another user. The altering of leniency rate was suggested, however that would convert the autonomous vehicle from being a public entity to a private entity. It is also assumed that the vehicles are self sustained in terms of energy and the passenger size of each vehicle is 6 people. It is also noted that the simulation is not running on real time. The animations of the vehicles are sped up.

Discussion

Drone Simulation

The Parrot Mambo Mission has an upper bound elevation detection mechanism, however, it does not have a lower bound elevation detection mechanism built into the drone. It is capable of utilizing two gadgets, a shooter and an arm grabber. The Parrot Bebop2 has both an upper bound elevation detection mechanism and a lower bound elevation detection mechanism and as well is able to hover in place with stabilization. For the Parrot Bebop2, the camera could be used to detect user/face detection with Microsoft Azure and scan barcodes to determine current location/destination[3]. It contains digital stabilization, GPS and Wifi. Its battery life can last up to 25 minutes of flight time. Signal range can extend up to 300 meters around the drone. The drone speed can reach up to 16 m/s horizontally and 6 m/s vertically. For Parrot Mambo, it weighs about 63 grams and it contains a camera with a 60 fps[2]. The battery life of the drone can last up to eight minutes with bumpers and 10 minutes without it. With a 2.1A charger, the charging time is estimated to be 30 minutes. The dimensions are 7.1 x 7.1 in. which is about half the size of the Bebop 2. It also contains a special feature, the cannon, which shoots balls, one shot every 1.5 seconds up to six shots. The shooting range is about 2 meters.

Weight Testing

Given the result of our weight testing it is apparent that the drone cannot adequately handle a load of more than just a few grams. While the Parrot drone did fly, it was very unstable and sounded like it was overworked. The 30 min flight time advertised would certainly be slashed if a load were to be carried for a long period of time. This will pose an issue for any group that is interested in having the drone carry any sort of load whatsoever. Particularly, the drone will have trouble carrying the sensors to be deployed around the city. Any attachment to give the drone inductive charging capabilities might cause problems as well.

The weight test highlighted just one of the limitations the Mambo and Bebop drones have. However, the weight carrying capacity is not the only limitation of the Parrot drones. First, our team ruled out practical use of the Mambo drone due to limited battery life. Parrot advertised the total flight time on a full charge as 9 minutes. In practice, our team found that the flight time is closer to about 6 minutes. This was the main driver for the decision to continue testing with the Bebop drone. While we must rule out the Mambo as a practical drone, it did serve as an effective practice drone for the team without risking the more expensive Bebop drone.

The Bebop drone has another shortcoming. The sensor which detects the height of the drone at any given time is located directly underneath the drone. As some groups had discussed, including ours, a carrying basket could be placed underneath the drone in order to easily carry objects. However, the placement of that basket would interfere with the altitude sensor possibly causing the drone to rise indefinitely. An alternative placement of the basket may cause balancing issues for the drone.

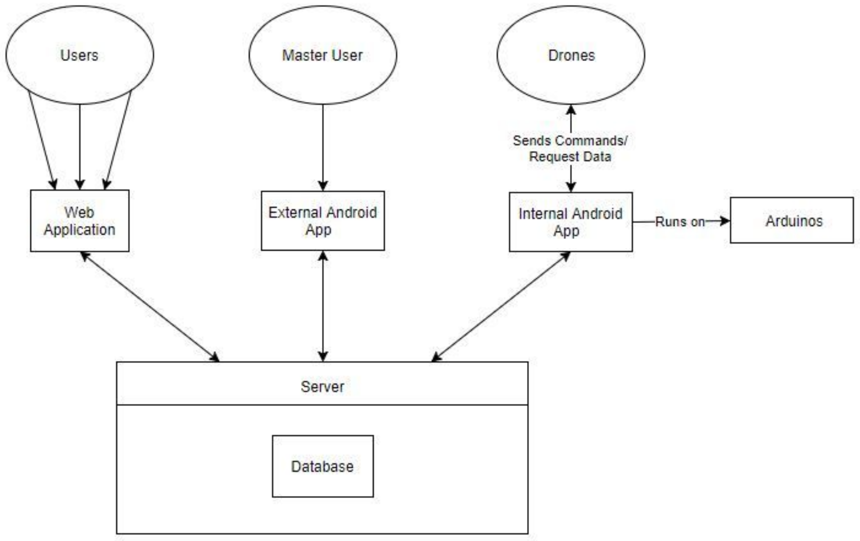
Finally, the biggest limitation of the Bebop drone and one that may significantly affect the progress of the transportation sub-team is the proprietary nature of the drone software. Our team had immense difficulty attempting to interface the drone with any of the applications that were explored. The applications which did interface seamlessly with the drones usually cost money to fully unlock. When fully unlocked, these programs, such as FreeFlight Pro, remained inflexible by only offering a few added features like GPS & visual tracking.

ROS

Although the ROS Development Studio is easy to setup and launch, the environment produced by the free account is very unstable. There is significant delay from where the code is executed to where the drone in the simulation obeys the command. The libraries offered are complicated to utilize and are very inefficient to what we are trying to accomplish. The ROS simulation offers no useful information, neither in terms of potential code or connecting to the drone directly, and was scratched as an idea.

Parrot Sphinx

The Parrot SDK for Android provides a variety of resources to look into. The documentation for working with the Parrot SDK in Java allows for the basic commands to be easily understood such as launch flight and end flight. The Parrot SDK allows for the Android app to access vital information such as GPS location, however the GPS tracking is not perfect with a 3 meter radius of imprecision[15]. The controller of the Bebop2 using the Parrot SDK is an Android application. Using the Android application as the controller requires the Android application to be installed to the Parrot Bebop2. Our vision is to upload the Android application to a microprocessor such as the audino or the raspberry pi and have the Android application connect to a server via internet to receive commands and upload data. The picture below shows the overarching schema of how every entity is connected.



The users will be able to request data from a location to be collected via a web application or application. The request will then be sent to the server which selects a drone to retrieve the data. A master user will be able to monitor the status of every drone. Each drone will be connected to the server via internet on the Android application. We were able to come up the the user Android application. We have made progress on getting the internal Android application to control the drone, however the connection of the Bebop2 drone to the Android application is still under investigation. Without getting a successful connection to the Bebop2 drone or the Bebop2 drone simulator, we are unable to test the code provided by the Parrot SDK.

MAV Proxy

The MAV Proxy simulation provides users the ability to use a virtual phone to control the destination of the drone as well as examining certain physical conditions of the drone (e.g. altitude, flight speed). After installing all software required to run this simulation, we ran into some trouble when running certain commands because all commands direct to the main hard drive (drive C); therefore, we advise the user to install all programs into the main drive. The 3DR app needs to be connected to the virtual drone generated by MAV Proxy. In order to do so, the ip address is needed as input into the MAV Proxy command prompt. This step has not been tested yet as we are under way in obtaining the ip address of a virtual device. Also, a Flight Gear program tutorial is needed as we are unsure on how to use this program. This simulation choice seems promising, but not enough man power and time was put into testing this simulation. If enough time was allotted, perhaps this simulation would produce decent results.

Ride-sharing Web-based Simulation

The ride-sharing web-based simulation is slowly becoming more user friendly to people without a coding background. Features have been added and minor bugs have been detected and fixed. One prominent feature is the toggle button to display the current status of a person/vehicle. Before this feature, users would have to watch the screen to locate a user and a vehicle without knowing which particular user is him/herself. Most of the progress for the ride-sharing web-based simulation have been put into creating the new database and slowly adapting each section of the code to use the database.

Conclusions

Overall, the ROS Development Studio provides great environment, however the commutating limitations on the free account exist. From exploring the ROS Development Studio, we have understood that a drone simulation requires a fair bit of computing power that web browser is not able to fully handle. Also, the delay between the submission and the execution of the code is a concern that we have for autonomous drones in general. We have to be more cautious and predict in advance the outcome of delayed execution of code.

By looking deeper into the Parrot SDK simulation, the main limitation of having autonomous drones is getting an internet connection to the drone. In order for the drone to successfully communicate with the server, it has to be connected to the internet. That would mean that each drone would require service. Also by having the Android application run on a microprocessor, there would be extra battery consumption that will need to be taken to account.

MAV Proxy provide great a great graphical interface, however similar to the rest of the drone simulations, a connection to the Bebop2 drone was not found.

The ride-sharing web-based simulation is far from being perfected. The algorithm in selecting the best vehicle takes many assumptions and is static. Current ride sharing companies have access to their own private transportation ride service data history that they utilize to select the best algorithm based on many external factors such as time of day and massive social events. Their algorithms uses techniques of machine learning and are dynamic.

Recommendations

Given both the hardware and software limitations that our team has encountered, we need a drone which is easily interfaceable with a laptop or server. Moreover, the drone must be able to adequately transport loads while remaining balanced. Thus, what our team has decided to focus on is to build our own drone through the help of online resources. One of the resources we found gives a very well detailed step by step guide as to how to build the drone, as well as some lessons on programming skills necessary to complete the project on your own[16]. The guide gives some suggestions to what equipment to use and the cost of this project is around $200-$300. The flight controller could also be built on our own from scratch. The final product is an Arduino Quadcopter with stabilization features. With this drone, we will be stepping away from flight simulators and begin working on writing an algorithm and testing it on the drone.

The next steps for the ride-sharing web-based is to have a more interactive/customizable settings to allow for users to choose what they want to test. There will also be an improvement in the storage of data by incorporating occasional updates of the data to the database. There will be a full incorporation of the new database into the code. A future step would be configure locks into the code to allow for the smooth transition of multiple vehicles with causing any collisions in the code.

References

[1] Parrot https://www.parrot.com/us/
[2] Parrot Mambo Misson https://www.parrot.com/us/minidrones/parrot-mambo-mission
[3] Parrot Bebop2 https://www.parrot.com/us/drones/parrot-bebop-2-fpv
[4] Ride-share Web-based Simulation https://github.com/VIPTransportationUsers/VIPWebSimulation/tree/master
[5] Shahani, Aarti. "Uber Says Hackers Stole Personal Data Of 57 Million Users" NPR, NPR, 21 Nov. 2017 https://www.npr.org/2017/11/21/565838123/uber-says-hackers-stole-personal-data-of-57-million-users?sc=tw
[6] Transport For London. "Licensing decision on Uber London Limited" Transport For London, Transport for London, 22 Sept. 2017. https://tfl.gov.uk/info-for/media/press-releases/2017/september/licensing-decision-on-uber-london-limited
[7] Wamsley, Laurel. "Uber Ends Its Controversial Post-Ride Tracking Of Users' Location." NPR. 29 Aug. 2017. http://www.npr.org/sections/thetwo-way/2017/08/29/547113818/uber-ends-its-controversial-post-ride-tracking-of-users-location
[8] Parrot FreeFlight App https://itunes.apple.com/us/app/freeflight-pro/id889985763?mt=8
[9] ROS http://www.theconstructsim.com/rds-ros-development-studio/?next=https%3A//rds.theconstructsim.com/simulations/
[10] Parrot SDK Java http://developer.parrot.com/docs/SDK3/?java#android
[11] Android Studio https://developer.android.com/studio/index.html
[12] Sphinx https://developer.parrot.com/docs/sphinx/index.html
[13] MAV Proxy http://qgroundcontrol.org/mavlink/mavproxy_startpage
[14] Flight Gear http://www.flightgear.org/

[15] Bebop2 GPS Precision https://community.parrot.com/t5/Bebop-2-Knowledge-Base/How-accurate-is-the-GPS-position-of-my-Bebop-during-flight/ta-p/129352
[16] Building Drone http://www.droneybee.com/arduino-quadcopter-guide/
[17] 3DR Tower App https://play.google.com/store/apps/details?id=org.droidplanner.android&hl=en