Okay Loomo

CVG Airport

4/28/2022

Stone Barrett
Karthik Malyala
Lauren Mikula
Saurin Patel
Alex Secor
Donald Wedding

Introduction

With an influx of passengers causing congestion, increase in delays, and rise of COVID-19 transmission, the Cincinnati/Northern Kentucky International Airport (CVG) has requested the development of an autonomous robotic concierge service using the Segway Loomo in order to provide passengers with easier accessibility and efficient travel times. In recent times, passengers face heavy emotional stress and anxiety in addition to terrible travel experiences. In hopes to enhance their passenger experience, CVG specifically requested the following goals to be addressed: (1) Provide research and development services to the sponsor. (2) Deliver a language translator to help with the international passengers. (3) Transport passengers to their destination throughout the airport. The second goal was addressed by a previous capstone team during the Spring 2021 semester. Goals 1 and 3 were addressed and achieved in the scope of this project while Goal 2 was achieved by the Capstone team from last year.

This project demonstrates and aims to act as a prototype towards utilizing the Loomo Personal Robot as an autonomous assistant in a non-industrial setting. Loomo is an advanced personal assistant robot that incorporates Intel RealSense technology in order to provide riding and autonomous track and follow operations. Loomo's navigation capabilities to follow a predetermined path are explored in order to achieve the previously defined goals.

Given that the previous Capstone team had worked on live language translation, our primary objective was to develop self-driving capabilities to allow Loomo to guide airport passengers to a specified gate in the shortest and most efficient path. In order to do so, we utilized the concepts of Grid-Based Navigation and the A* Algorithm Heuristic with Java, Android Studio, and the Segway Loomo API to find the shortest and most effective route to a desired destination (gate).

System Description

The Segway Loomo is an advanced personal robot along with being a smart self-balancing electric transporter which allows the rider to cruise on most type of terrains with speeds up to 11 mph and 22 miles of range in a single charge. Apart from riding, Segway Loomo uses advanced computer vision along with powerful AI that incorporates Intel RealSense technology which allows the Loomo to autonomously follow the rider, avoiding any obstacles in between. It also uses voice and gesture controls to shoot stabilized video, take pictures, and much more. Segway Loomo is very developer friendly with its free Android SDK which allows developers to access all Loomo's features and AI capabilities to apply real world solutions.

Figure 1: Segway Loomo Personal Robot

System Requirements

To begin developing with Loomo, Loomo needs to be in developer mode. Since Loomo runs on the Android Operating System, developers need to use Android Studio as a development tool. Loomo runs specifically on Android 5.1 Lollipop Operating System, therefore developers must develop using Android 5.1 (Lollipop) development kit. Segway Loomo provides developers with SDK for all its features and while developing only the dependencies need to be added to the build.gradle file of the application to develop with specific features of Loomo.

```
dependencies {
    implementation 'com.segway.robot:visionsdk:0.6.607'
    implementation 'com.segway.robot:speech-sdk:0.5.329'
    implementation 'com.segway.robot:headsdk:0.6.814'
    implementation 'com.segway.robot:basesdk:0.6.814'
    implementation 'com.segway.robot:sensorsdk:0.6.814'
    implementation 'com.segway.robot:robot-connectivity-sdk:0.5.122'
    implementation 'com.segway.robot:mobile-connectivity-sdk:0.5.122'
    implementation 'com.segway.robot:emoji:0.1.29'
    implementation 'com.segway.robot:support-lib:1.1.2'
}
```

Figure 2: SDK dependencies for all features of Loomo

System Specifications

Loomo Specifications:

Size	650mm height, 310mm length, 570mm width
Weight (Loomo)	42 lbs.
Payload (Rider)	Maximum of 220 lbs.
LCD Screen	4.3 inch, 480 x 800 pixels
HD Camera	1080p 30Hz streaming with 104 degrees FOV

3D Camera	Intel RealSense ZR300 Camera for depth-sensing & motion tracking
Sensors	Ultrasonic sensors, Infrared distance sensors, touch sensors, encoders, IMUs
Speed Limit	4.3 mph in robot mode and 11 mph in ride mode
Range	About 22 miles per charge
Traversable Terrain	Paved roads and sidewalks, packed dirt, slopes < 15°, obstacles < 1cm, gaps < 3cm
Battery	329 Wh
Waterproof Rating	IPX4

VA50EC Android Tablet Specifications:

Processor	Intel Cherry Trail Type4 Z8550/Z8750, 2.4GHz, 64bit
Memory	4GB, LPDDR3 1600
Storage	64 GB
Wireless	Wi-Fi 802.11 ac, Bluetooth 4.0 dual band
Touchscreen	4.3 inch IPS, 480 x 800 pixels, 5 Point multi touch
Cameras	DS4T Module (2MP Camera), 2 infrared cameras, SoC color camera, Motion camera, 8MP front camera
Battery	2600 mAh
Input/Output	Power connector: 12V/3A, USB Type C / OTG, LD0 Power, SPI, Microphones
Operating System	Android 5.1 Lollipop
Weight	350g

System Diagrams

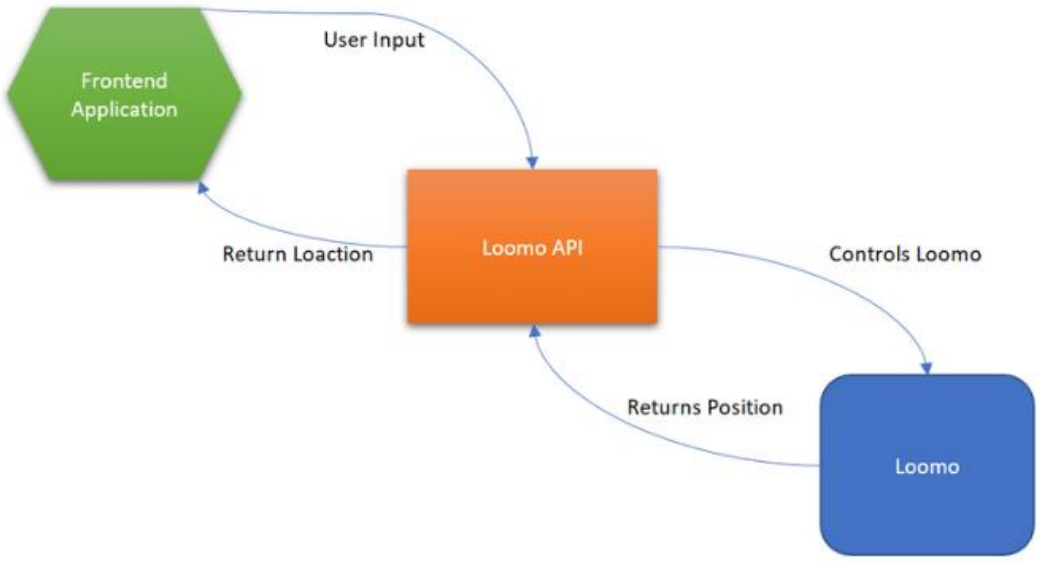


Figure 3: System diagram of Loomo interacting with the application

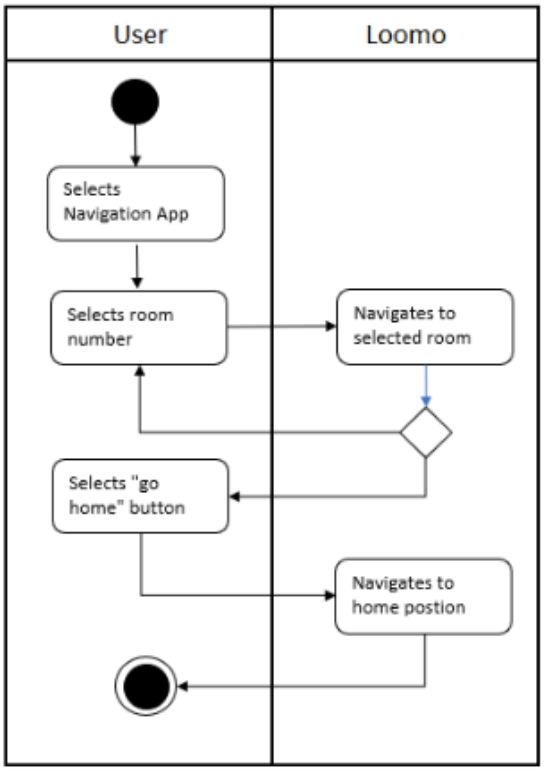


Figure 4: Activity and Use case diagram of the Navigation app

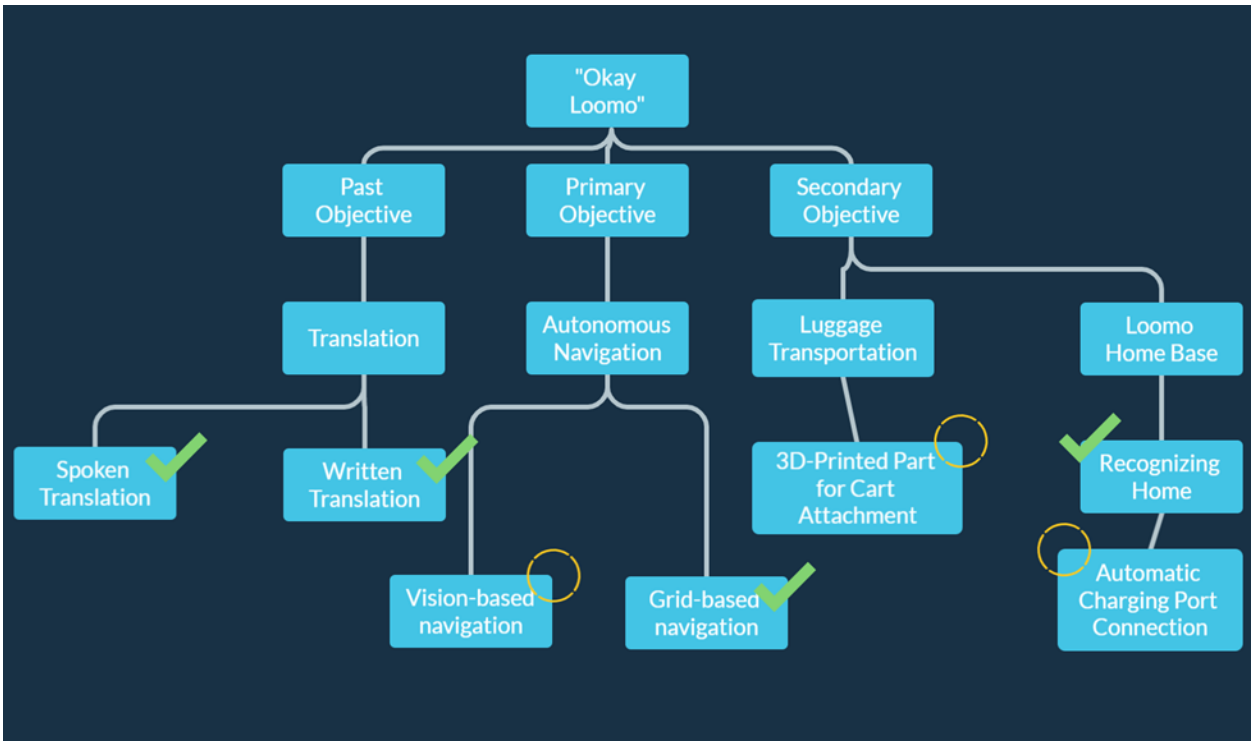


Figure 5: Objective tree by the end of the project

Hardware Overview Diagram

Segway Loomo front side

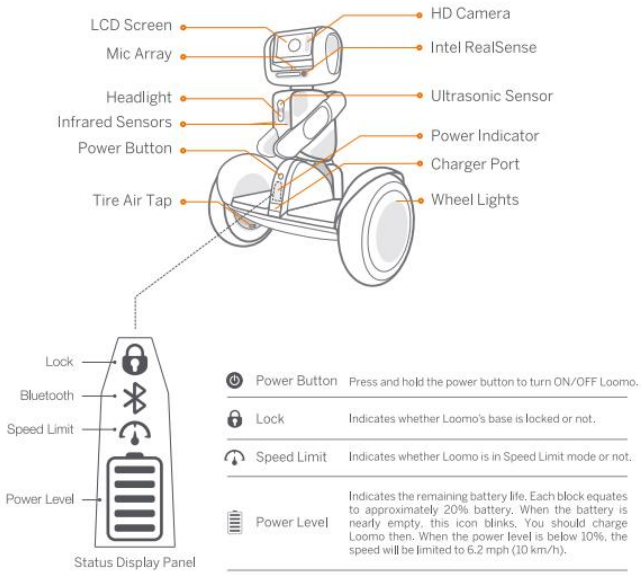


Figure 6: Segway Loomo front side diagram

Segway Loomo back side

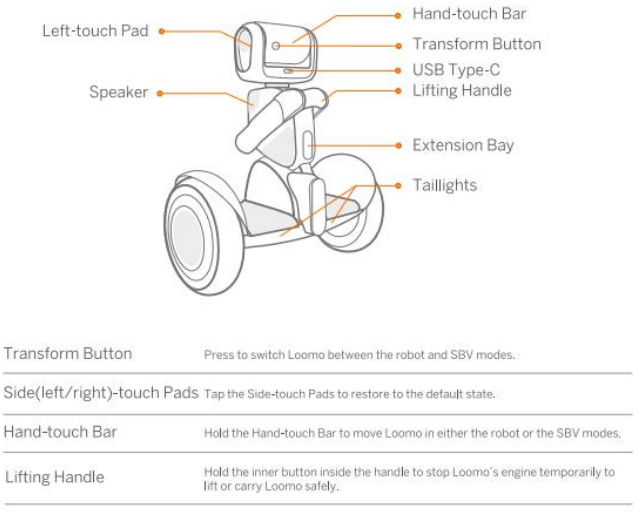


*Figure **7:** Segway Loomo back side diagram*

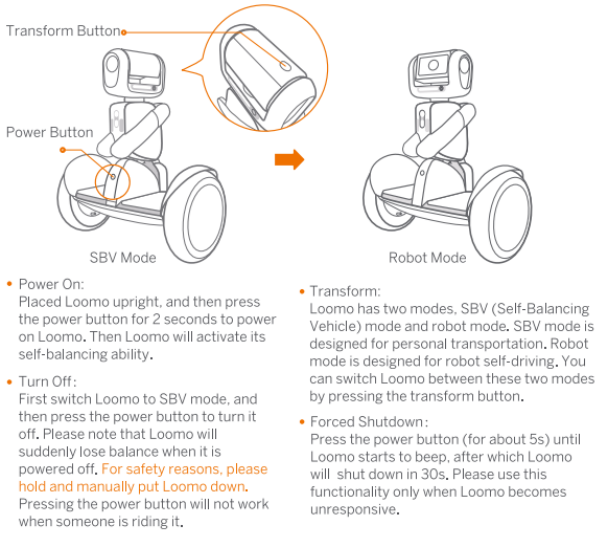


Figure 8: Basic guide to switch between two modes of Loomo: SBV and Robot

Segway Loomo has an integrated Android tablet VA50EC. The LCD screen on the front side of Loomo is the LCD screen of the VA50EC tablet. The VA50EC tablet also had multiple cameras that Loomo uses.

No.	Name	No.	Name
1	LCD Touch Panel	2	8M-pixel Camera

Figure 9: VA50EC Android tablet front screen

No.	Name	No.	Name
1	Right Infrared Camera	4	Left Infrared Camera
2	SoC Color Camera	5	Motion Camera
3	LED Flash Light		

Figure 10: VA50EC Android tablet integrated camera system

Software Overview Diagram

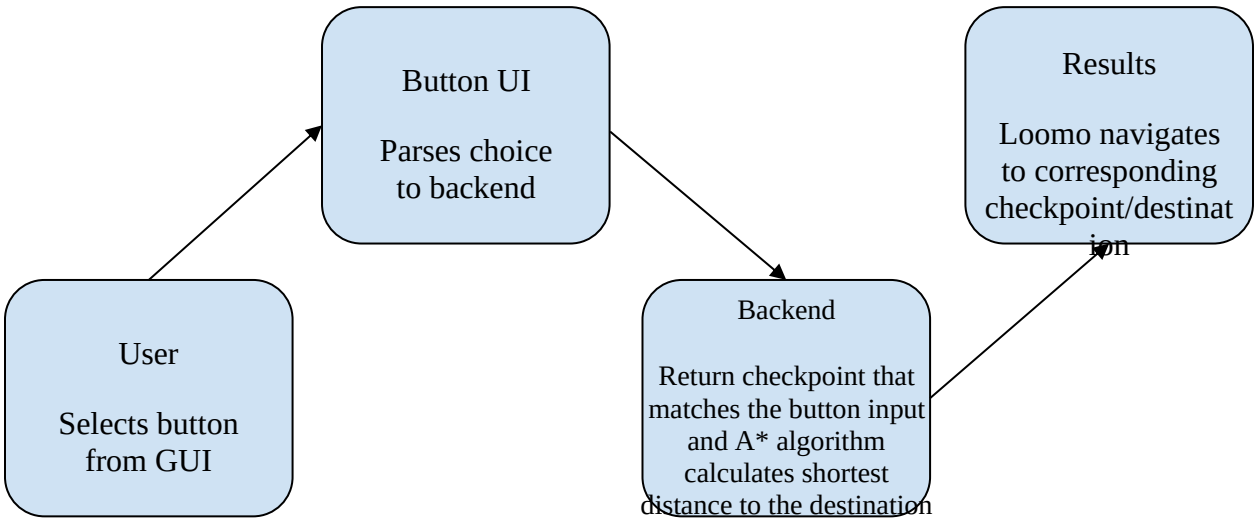


Figure 11: Software overview diagram

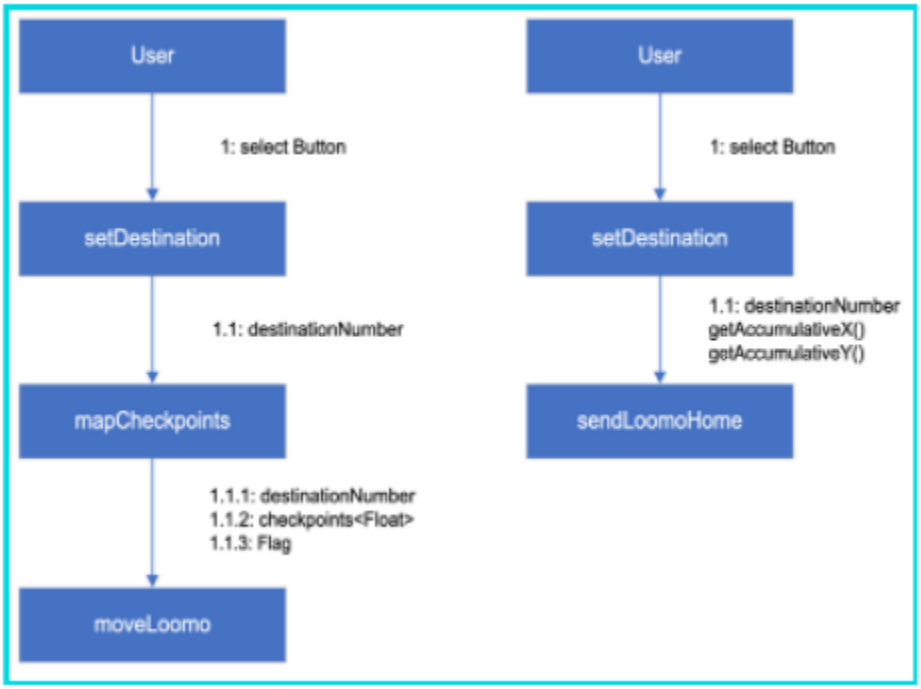


Figure 12: Communication diagram show communication between user and Loomo

Economical, Technical and Time Constraints

Since this project was in partnership between JB Speed School of Engineering and CVG airport, two Segway Loomos were provided to work on. One was provided by Speed School and the other was provided by the CVG airport. Therefore, there weren't any economical constraints to our project.

For the entirety of the project, we only worked on the Loomo provided by the Speed School. While testing our application on the Loomo provided by the Speed School, we discovered a problem of it steering away towards the right side over long distances. We concluded, after many tests, that there was some problem in Loomo's hardware. That was one of the technical constraints we faced. Another technical constraint we faced was not being able to test the application on the Loomo provided by the CVG airport. We wanted to test the application onto CVG's Loomo to test if there is the same problem of Loomo steering towards the right as it was on Speed School's Loomo.

From the beginning, we set weekly milestones for our project to keep the project on track and finish the project on time. This allowed us to stay focused on the project and divide up tasks if we lagged behind.

Detailed Implementation

Hardware Detailed Implementation

Loomo provided most of the hardware capabilities required to achieve the team's goal. On the back side of Loomo, there is a USB Type C port that gives access to developers to import applications onto Loomo from Android Studio. Everytime an update was made to the application, we had to import the updated application onto Loomo using a USB Type C cable. The USB Type C port was also used to help debug the code when stepping through the code line by line.

Figure 13: USB Type C port on the back of Loomo used for importing apps

Software Detailed Implementation

Loomo API Navigation Program

After becoming familiar with Loomo's software development kit, or SDK, manual, we began working on an application that allowed us to experiment with Loomo's locomotion. This app was developed in Android Studio, where the front-end is created using XML and the code-behind is Java. To begin, we started with designing an app that had a single button. This button's OnClick event would call Loomo's locomotion classes and move a specified number of meters forward.

To move Loomo forward, there are preexisting classes that make it possible. Loomo's SDK provides a *Base* class, which pertains to everything that Loomo's base needs to function. After creating an instance of its *Base* class, we can specify the control mode to be in *CONTROL_MODE_NAVIGATION*. In this mode, we can command Loomo to move to a certain point through checkpoints. This is a crucial part of how Loomo's locomotion is made possible. Before we can set the checkpoints, Loomo also stores its original location, so we clear that before Loomo is told to start a new route. Next, we capture the pose of Loomo relative to its current orientation. This pose is recorded in x, y, and z coordinates; positive x is always pointing forwards from the base, positive y is to the left of the base, and z is perpendicular to the ground. Lastly, the captured pose is set as Loomo's original point. With the *Base* class initialized, Loomo is ready to begin its route.

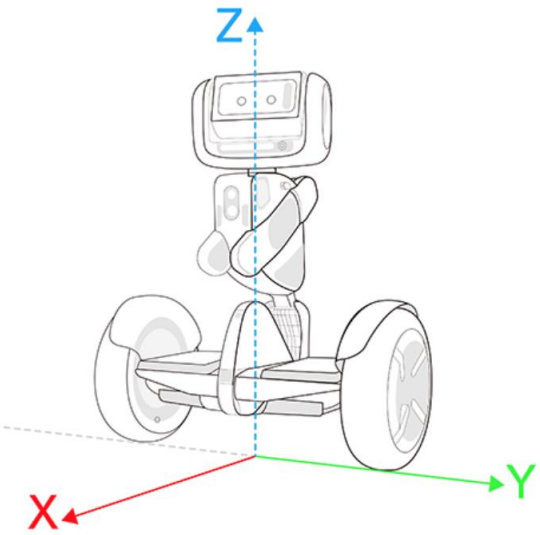


Figure 14: Loomo's base pose

```
mBase.setControlMode(Base.CONTROL_MODE_NAVIGATION);
mBase.cleanOriginalPoint();
Pose2D pose2D = mBase.getOdometryPose( time: -1);
mBase.setOriginalPoint(pose2D);
```

Figure 15: Instantiating Loomo's base

As previously mentioned, with Loomo's control mode, we can set checkpoints that Loomo can follow. This generated many ideas, as the goal of the project is to guide airport travelers to their respective gates. The initial idea was, if we could manage to create checkpoints at each gate, and by making each checkpoint unique, Loomo could differentiate between them and successfully travel to each one. So, the work began, and the initial application was created. This app was a bare-bones approach to the situation, but it allowed us to become familiar with Loomo's unique movements. Each of the checkpoints were added using the addCheckpoint() method, and we supplied the x and y coordinate where Loomo needed to travel. Once we

gathered enough information, we decided to use our workplace to further our processes and use the door numbers as destination points.

```
// move Loomo
mBase.addCheckPoint((float)moveX, (float)moveY);
```

Figure 16: addCheckPoint() method with x and y float parameters

These destination points are rooms 204 to 209, where 206 and 207 are aligned adjacent to one another on a corner; so, it creates the test case of how Loomo would react after turning its body. The new app had a reconfigured user interface, or UI. Each door has a corresponding button, and 204 is labeled as the “Go Home” button because it was decided to be the starting point. For the code-behind, Loomo was programmed with a list of distances. These distances were the measurements between door to door. By giving each button a unique key, the code-behind was able to parse the clicked button’s key and determine which distances Loomo needed to travel. For example, if Loomo was commanded to travel to room 206, Loomo will move 1.9 meters in the positive x direction, followed by another 3.45 meters where it successfully reaches room 206. This is all made possible by the addCheckpoints() method and the list of distances between the doors.

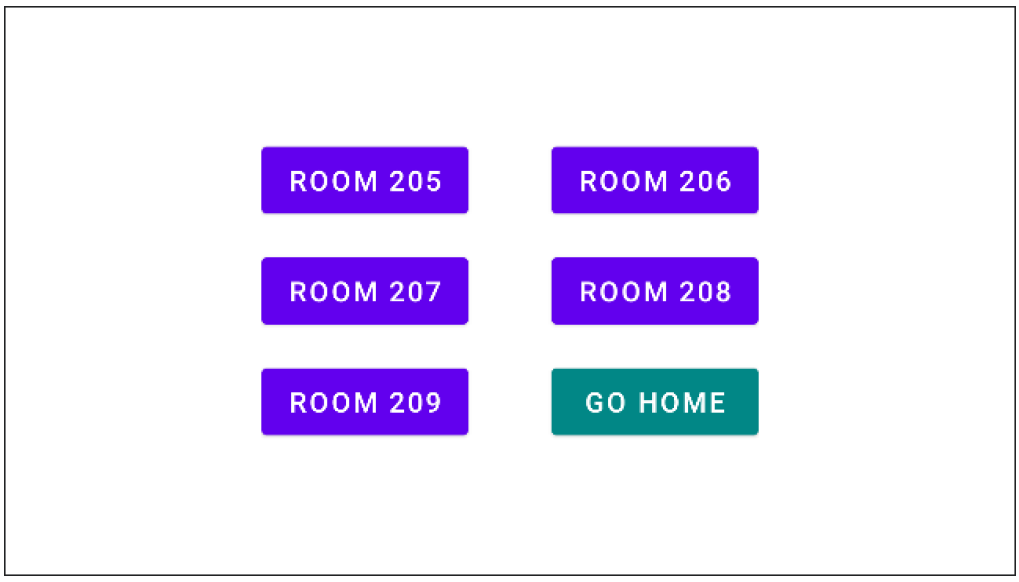


Figure 17: Loomo's locomotion application's UI

Predetermining the destinations between each door was a good idea for a small-scale usage of our concierge task. We knew when CVG wanted to implement this into their airport, that predetermination was not the best way for Loomo to create its routes. If there were obstructions, Loomo would not avoid them. Furthermore, Loomo should not have to avoid the same obstructions if they are stationary (for example, moving sidewalks). So, we decided that implementing a path finding algorithm would best suit our needs moving forwards. In our case, the doors would be used as destinations again, blocks were implemented around tables, and Loomo would determine the most optimal path to travel to successfully traverse to the doors.

To implement the a* algorithm to work correctly with Loomo, we wanted the algorithm to generate and return a path of (x,y) coordinates. To accomplish this, we created a *Point* class that stores x and y values in a coordinate layout. (More information about the A* algorithm can be found in the following section: A* Algorithm). The class was paired with methods that can return the single x and y value of a single point, which is later used to feed into Loomo's addCheckpoint() method. Next, we measured our entire workspace area and decided that one

point in the grid would be equivalent to .1 meters in the real world. Meaning every progression between coordinates on the grid will correspond to Loomo moving .1 meters in the same direction. Using this paired with the returned list of (x,y) coordinates,

By using this, we were able to have Loomo send its starting position and destination to the a* algorithm to compute the most efficient path. This path is returned back to the code-behind of Loomo, and is then parsed and passed to another method findDifferenceOfCoordinates*()*, where two coordinates are sent to find the difference between them. Due to the way we measured the area, this function will always return 1, and if it returns negative 1 then Loomo is traveling backwards. These differences are fed into Loomo's addCheckPoint() method and Loomo successfully moves to each destination without any hardcoded values besides the starting and destination positions.

```
// finds the difference between coordinates
public int findDifferenceOfCoordinates(int coord1, int coord2)
{
    // returns the distance needed to travel.
    // we mapped out the astar grid to give us a movement of 1.
    return coord2 - coord1;
}
```

Figure 18: findDifferenceOfCoordinates() method

```
SO 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 X 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 X 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 X 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 X 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 X 0 BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 X BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 BL BL BL BL BL BL BL BL BL BL BL BL BL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X X X X X X X X DE
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 19: The grid produced by the A algorithm*

Due to the initial progression of Loomo being created in a maintainable way, we were able to reuse a lot of the functions and properties. The adaptation of implementing the a* algorithm with Loomo's locomotion logic was successful, and now Loomo can decide on its own what the best route will be from its starting to end location. As a group, we believe that this is the most crucial step in transforming Loomo into a concierge service for the CVG airport. Once a grid is created for the airport, blockades and destinations can be set, and Loomo should flawlessly transition into a service robot.

A* Search Algorithm Program

As previously mentioned, we decided to use a path finding method instead of hard-coding the predetermined distances and paths between different positions. Programming the predetermined distances worked well enough for our initial small-scale testing, but we knew there would be a better, more efficient solution. Due to our team member's experience and previous knowledge of the A* Search Algorithm, we chose it as our path finding method for Loomo.

The A* Search Algorithm generates a graph of nodes and uses a combination of an exact and estimated cost between two nodes in order to determine an optimal path. For this project, an optimal path is considered to be the shortest valid path between two points. To do this, A* determines the cost to reach a neighboring node and the cost to get from that node to the goal node. This function can be described as

$$f(n) = g(n) + h(n)$$

where n is the current node, g(n) is the exact cost from the start node to node n, and h(n) is the estimated heuristic cost from n to the goal node. h(n) is a heuristic function, meaning it is a problem specific function that must be chosen carefully. We decided that h(n) would be calculated as the Manhattan Distance between the current and goal node. Manhattan Distance can be described by the following function:

$$h(n) = |x_1 - x_2| + |y_1 - y_2|$$

where (x_1, y_1) are the coordinates of the current node n and (x_2, y_2) are the coordinates of the goal node. Manhattan Distance was chosen as the heuristic for this implementation because of the grid-based system being used. Manhattan Distance is effective when movement in the

algorithm is restricted to non-diagonal movements. Loomo's locomotion service does not accept diagonal directions, hence Manhattan Distance is the appropriate choice as the heuristic.

The exact cost, or g(n), was determined by the Pythagorean Theorem. For the implementation with Loomo, we needed to ensure that up-down movements had the same weight as left-right movements in the algorithm. Therefore, the cost between horizontal and vertical movements needed to be equivalent, and the cost of diagonal movements had to accurately be reflected. Since $\sqrt{10^2 + 10^2} = 14.142$, we chose the cost of horizontal/vertical movements to be 10, and diagonal movements to be 14. The diagonal distance was rounded down to 14 for simplicity in the A* calculations.

Once we determined how to set up the cost function, we were ready to implement the algorithm. The algorithm accepts a user-defined grid as well as source and destination position as input. The grid is essential to the algorithm, so providing it with an appropriate representative grid of the testing area was incredibly important. We measured our entire testing area and decided that one point in the grid would be equivalent to .1 meters in the real world. In other words, every progression between coordinates on the grid will correspond to Loomo moving .1 meters in the same direction. Additionally, the A* Algorithm can be supplied with grid positions that are blocked from the final path. In other words, these blocked grid positions can force Loomo to avoid certain areas and find a path around them. This additional aspect was essential to the success and future usability of the Loomo, as this allows developers to restrict Loomo's movements.

We roughly measured the testing area to be a 6.7 x 9.3-meter rectangle. This was translated into a 67 x 93 block grid for the A* algorithm. Based on these measurements, we also

obtained the corresponding grid positions of each room in the testing area. These positions can be seen in the following table.

Room	Corresponding Grid Position
Home	(0,8)
205	(22,8)
206	(55,8)
207	(55,8)
208	(55,46)
209	(55,83)

The testing area also included sections that we wanted to block from the algorithm. Those sections included a lab room and a sitting/break room area with tables and chairs. We did not want Loomo to weave in and out of all the tables/chairs, so the entire area was blocked off. A visual representation of this grid and the placements of all room positions and blocked areas can be seen in the next figure. Green areas represent available cells and red areas represent blocked cells.

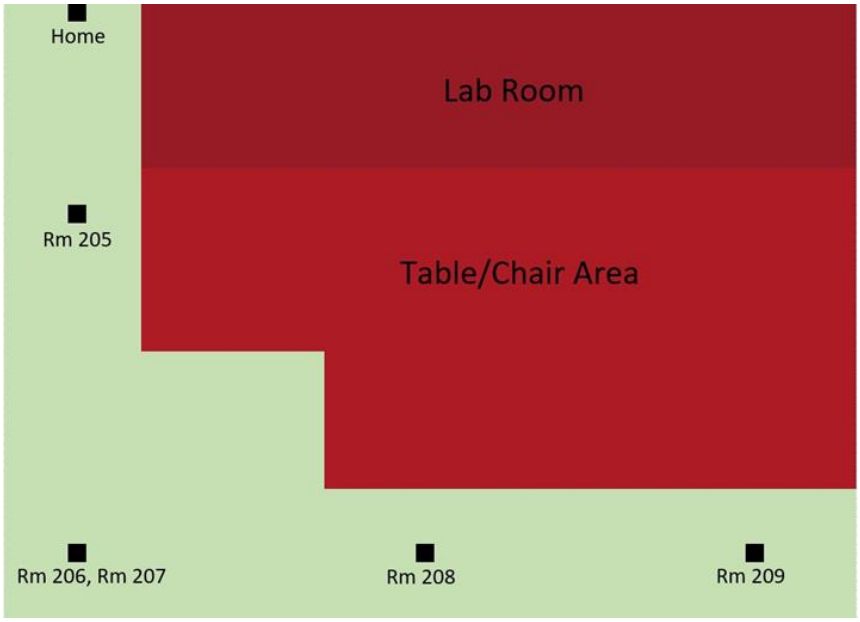


Figure 20: Representation of the grid created for A Algorithm*

After defining the grid, the starting positions, the ending positions, and the blocked positions, the algorithm can finally calculate the optimal path. A* essentially maintains a tree of paths, originating at the starting node and expanding the paths one edge at a time until the goal node is reached. In order to keep track of theses paths, A* uses a priority queue. At each step, A* calculates f(n). The node resulting in the smallest f(n) value is removed from the priority queue, and the f(n) and g(n) values of its neighboring nodes are updated. At any point, if the algorithm finds a shorter f(n) value for a node already removed from the priority queue, the algorithm backtracks and updates the path to reflect the newer, smaller cost. This process continues until the goal node is reached. A diagram of the algorithm can be seen in the following figure.. Red nodes are blocked nodes, blue nodes are available nodes, and green nodes have been added to the path. Nodes with a dotted outline are the current nodes being expanded.

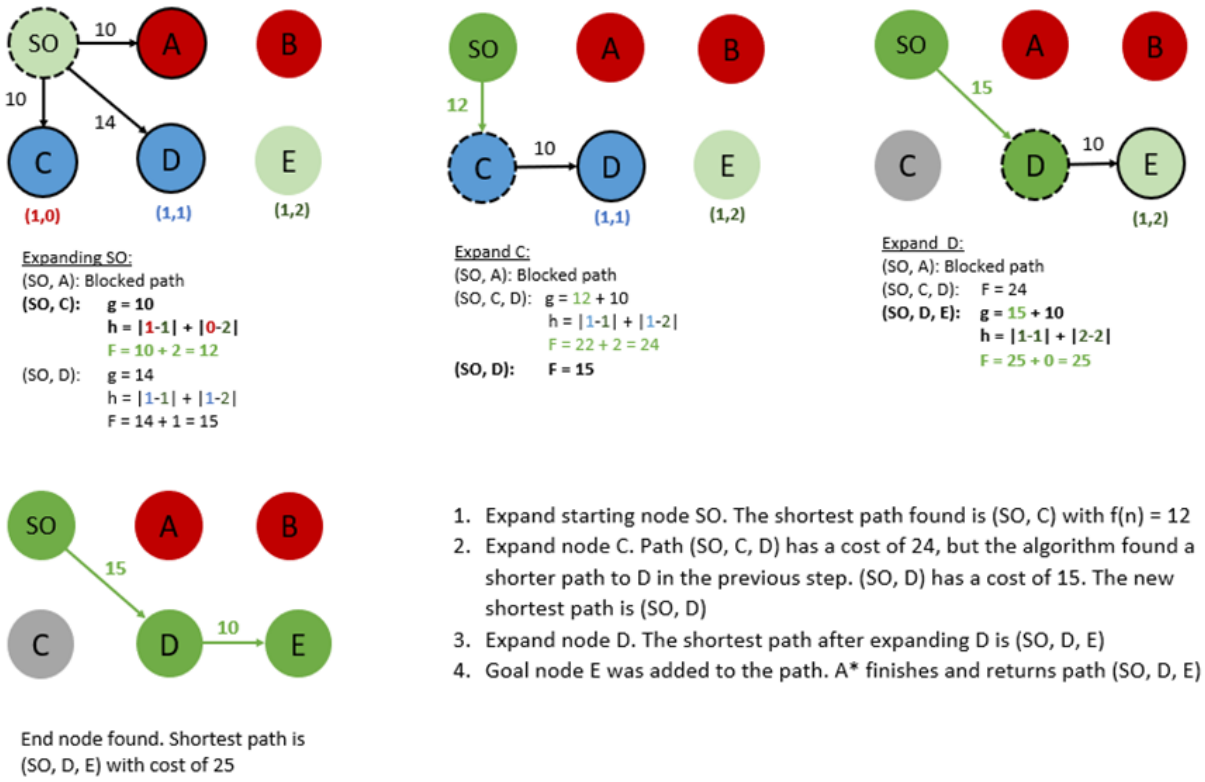


Figure 21: Detailed diagram of A Algorithm*

In order to implement the A* algorithm with Loomo, the algorithm was written in Java. 3 classes were created: public class Cell, public class AStar, and public class Blocks. Class Cell initializes each cell, or block, in the grid. Class Blocks initializes which nodes are to be blocked. Class AStar is where the algorithm is actually implemented.

The class AStar accepts a grid with n columns and m rows, the i and j coordinates of a starting node, the i and j coordinates of the destination node, and the blocked cells. When an instance of the AStar class is created, AStar uses Cell to define and keep track of the grid and each individual cell being evaluated. A cell is defined by its i and j coordinates. For each cell, the algorithm determines f(n) as described previously until the goal node is reached.; g(n) is determined to be either 14 (set with variable DIAGONAL_COST) or 10 (set with variable

V_H_COST); h(n) is calculated by obtaining the manhattan distance of the (i,j) coordinates and kept in variable heuristicCost; and f(n) is calculated as a sum of either DIAGONAL_COST or V_H_COST and heuristicCost.

While f(n) is being calculated and cells are being discovered, A* keeps track of this information in a priority queue. The priority queue is defined as PriorityQueue<Cell>((Cell c1, Cell c2)) where c1 represents the current node being evaluated, and c2 represents the same cell if it has already been evaluated and added to the list. This is used to determine if the current path's cost has been determined as larger than a previous path with the same node(s). The priority queue, final cost, and optimal path are all calculated in the method process(). Eventually, the destination node is reached and the path is returned.

The separate, original A* program has additional methods that provide useful information for the developer. The methods include display(), displayScores() and displaySolution(). display() returns a visual representation of the graph to the terminal. The developer can clearly see the size of the grid, where the source and destination positions are located, and which cells are blocked. Regular cells are displayed with a '0', blocked cells are displayed with 'BL', the starting cell is displayed with 'SO', and the destination cell is displayed with 'DE'. displayScores() returns a similar view to display(), but instead of different symbols representing cells, the cost of moving to each cell is displayed in that position. This method is useful to confirm the accuracy of the algorithm and see why the algorithm chose the final path. displaySolution() returns the original grid from display(), but with the symbol 'X' in the corresponding final path positions. It also prints the coordinates of the path to the terminal. These methods were not included in the implementation with Loomo, but they are mentioned here due to their usefulness for future developments.

The methods display() and displayScores() were not included when implemented with Loomo. There is no need for the general users to see that information. Instead, only methods process() and displaySolution() are kept. displaySolution(), however, is altered to only return the final path. The Loomo API program is able to take this path as an input, and apply transformations/calculations as described in the previous section.

Testing and Evaluation

To test Loomo with the Android application, we used the second floor area of the Duthie Center of Engineering. Upstairs you will find professors' offices, with the rooms labeled as 204 – 209. Since we could not deploy Loomo to CVG, we instead used a small-scale test space where Loomo will travel from its home location, room 204, to the other doors.

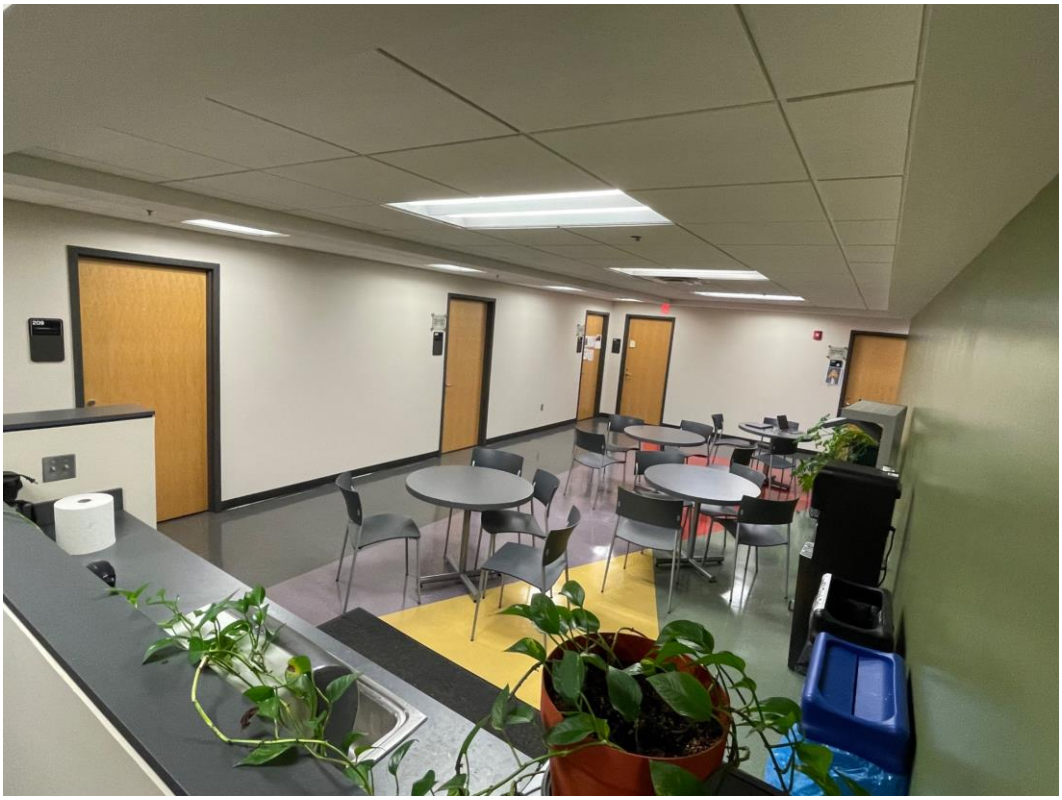


Figure 22: Loomo's testing area (from left to right: room 209, room 208, room 207, room 206, room 205)

Due to the layout of Duthie Center, Loomo could be tested moving forwards, backwards, and on a turn. Handling the turn Loomo would have to make when traveling from rooms 204 through 207, to rooms 208 or 209 was helpful when evaluating each possible scenario that it would face in the CVG airport. This basic layout was exactly what we needed for the midterm

demo. It allowed Loomo to move to each room as if it were a gate at the airport. However, we intended for Loomo to move in the most efficient way possible, so moving in straight lines is not something we wanted to limit Loomo to.

When we introduced the a* algorithm, we wanted to show that Loomo will create the best path possible. If every door is a straight path, Loomo would not be showcasing one of its top features. For this, we measured the entire room when creating the a* grid. Each meter was recorded and translated into the grid's x and y coordinates by breaking them up into .1 meters. So, every point traveled will correlate to .1 meters. We also wanted to showcase Loomo's pathfinding abilities, so we decided to allow Loomo to cut a corner in its grid, and it translated perfectly.

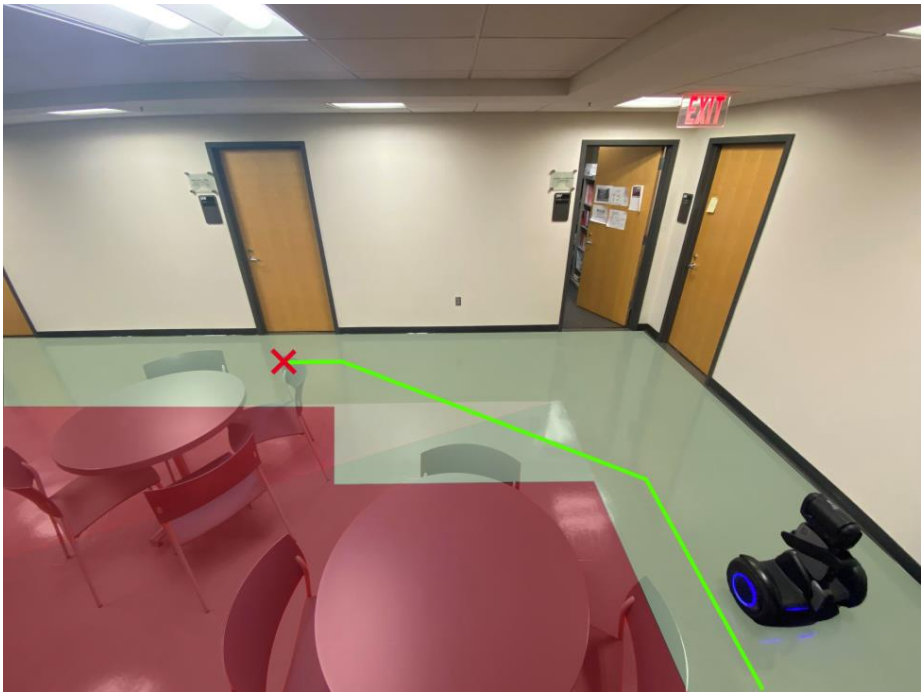


Figure 23: Visualization of Loomo following the a generated path*

The testing done upstairs of the Duthie Center was informative for us when we began working with Loomo's locomotion. A lot of advancements were made through trial and error,

but we found by working out specific problems, related problems would also be resolved. This is not to say that the test area was perfect; in fact, Loomo would occasionally run into walls when testing. It would be ideal to find a more open area to test Loomo in; so, instead of running into walls, Loomo could complete its course and the programmer could understand exactly what Loomo's path code was generating. However, Loomo could not leave the Duthie Center, and the advancements made through the small-scaled testing was far greater than not having testing at all.

Analysis of Results

Through the results of testing, we were able to understand Loomo's movement through its addCheckpoint() method. We also learned about Loomo's faults. Loomo tends to drift to the right while it is traveling between its destinations. To counter this, in the midterm demo we made

Loomo over-correct itself when turning. Meaning that it would turn more to the left or right so that it could drift while moving towards the next door, and in result would be corrected by the time it arrives. For the final demo, we did not notice the drift to be nearly as bad. In fact, we were impressed with how accurate Loomo would travel to each destination.

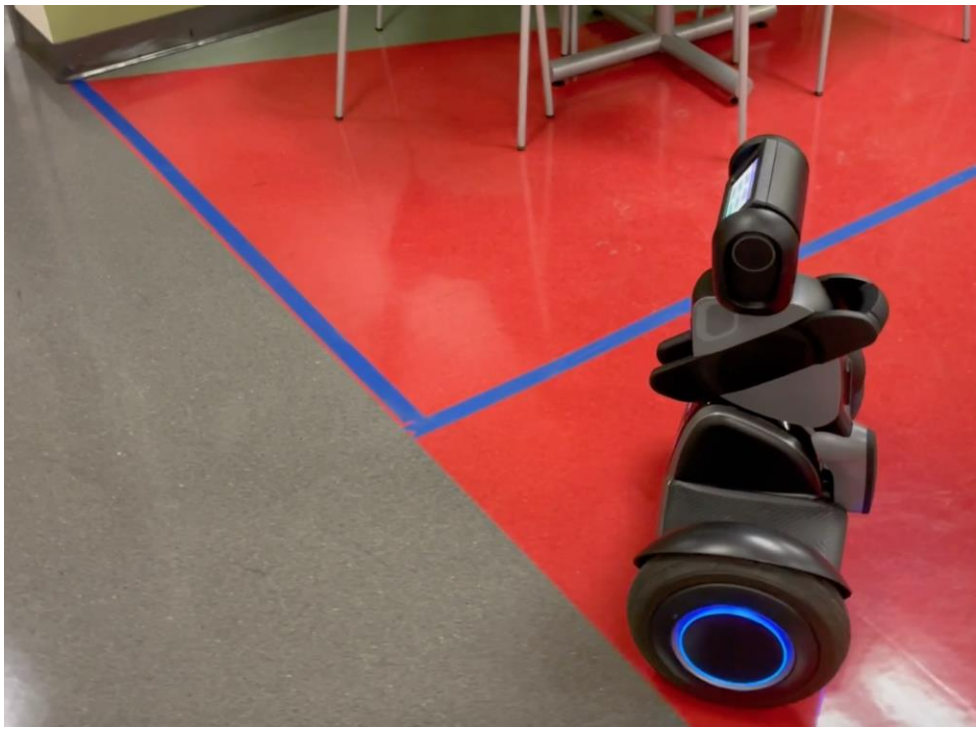


Figure 24: Testing Loomo's path creation with taped blockades

For Loomo to succeed, countless hours of testing must be done. There is no emulator for Loomo, so all the debugging and testing must be done manually. The knowledge accumulated during testing was far greater than reading online documentation. Due to the time spent towards testing, we got Loomo exactly where we wanted it to be; at a solid foundation for future groups to add onto its functionality.

Societal Impact

The “Okay Loomo” project likely will not have an immediate earth-shattering impact on society, but it is the first few steps towards greater progress. As science fiction movies have all but made a robot controlled dystopia some kind of inevitability, a tiny, cute robot acting as a guide would definitely be a good taste of what robots actually can do and how they can be helpful and used for more than hunting down Sarah Connor. The Loomo is also very easy to use, making it usable by both young and old, and using its built- in voice recognition could be made to help nearly anyone, including the blind, who normally would have a problem pressing a

button on a screen. Assuming a cart attachment was added to it, Loomo also could help out people carry larger pieces of luggage or maybe even people, keeping children from getting lost or luggage from being forgotten. If the translator functions are fully implemented, Loomo also can act as a translator of sorts between multiple people (kind of like an adorable mix between R2-D2 and C3PO). Loomo also can be made to run 24/7, 365 days of the year, meaning critical airport times such as during the holidays are no issue for it: Loomo needs no vacations, and can guide those who do around without issue without any variation in its happiness or care. This may help airports and airlines with less than stellar reputations for customer service, as Loomo can take over some of the duties. Loomo would also act as another piece in the CVG Airport Robot program, showing that a bunch of independently created and programmed robots could easily be made to work together as a collective, even when made by competing companies. This might lead to other airports and companies being more likely to invest in multiple types of robots without worrying about having to be loyal to a single brand else have everything break down. Additionally, Loomo can act as both a spark for younger children to become invested in and hopefully seek to improve and to calm the people who feel threatened by the robot uprising.

Project Contribution to Society

As the field of mobile robotics expands, there is a rising demand for Autonomous Mobile Robot (AMR) technology that can provide autonomy in daily repetitive tasks that humans partake in. Airports can be a mass hub for confusion, social anxiety, stress, and other impactful factors that can make a fun travel experience really terrible for passengers. With the rise of the COVID-19 pandemic, airports have become dangerous hotspots for disease transmission, even with masks and mandates. Select airports have also reported the struggle to cope with the

demanding costs of additional sanitation and labor which makes it even more difficult to withhold their passenger experience from pre-COVID. To combat this, the COVID-19 pandemic has given birth to a new technological era wherein automation and robotics are progressing through the daily challenges that humans must face through this disease. From automated e-commerce shipping to social robotics that treat patients with social disorders like autism, the field of robotics has proven to be fruitful in various ways.

Although ensuring security is the major challenge in the aviation and commercial flight sector, the congestion issue within the airports should also be considered. According to the Operations Network (OPSNET), the official source of Federal Aviation Administration (FAA) air traffic operations and delay data, 8 of the 31 busiest U.S. airports, which handle nearly one quarter of total passenger enplanements, exceeded the regular threshold of 3% of their entire fleet being delayed for 15 minutes or more in 2000.

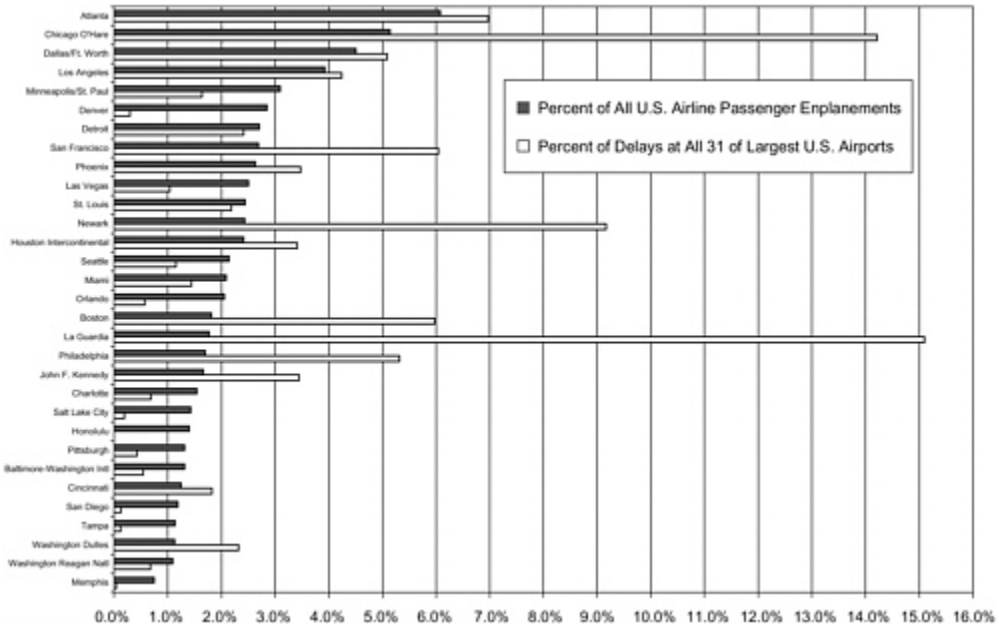


Figure 25

As seen in the above graph, the stark difference between the delays and the percent of enplanements is quite significant, especially in La Guardia's case. FAA, however, measures that

these delays weren't related to air traffic control. Rather, OPSNET records that it is one of the five factors: 1) weather, 2) air traffic control issues, 3) closed runways, 4) high passenger congestion in terminals, or 5) other.

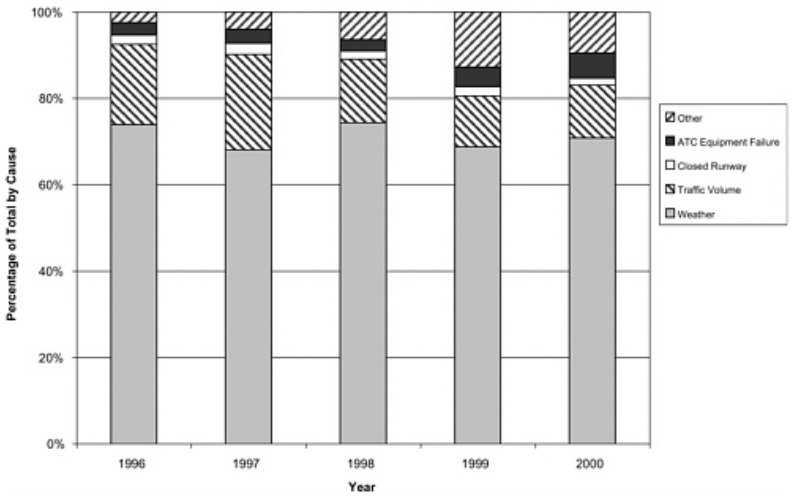


Figure 26

While weather is a primary reason for delays in the above chart, the traffic volume is next in line that can actually be resolved through innovative solutions. Since weather is highly unstable and unpredictable at times, an area where airports can really improve upon is fixing the congestion and traffic volume issues within their airports. The primary solution that most airports around the world are headed towards is the deployment of Autonomous Mobile Robots (AMR) to assist passengers with activities ranging from luggage handling to passenger guidance. As mentioned in an article from Yujin Robot, the "global market for airport robotics reached $426 million in 2020." Additionally, Insight Partners had recorded in their *Airport Robots Market: Forecast to 2028 - Covid-19 Impact and Global Analysis* study that specific regions would be progressing with their implementation significantly. For example, the Asia/Pacific region is projected to grow to $970.8 million/23.3% CAGR, the Europe region to $477.2 million/21.2%

CAGR, and the North American region to $428.4 million/22.6% CAGR. The above statistics only represent 3 out of the 6 major regions across the world.

This “Okay Loomo” project progresses towards resolving the majority of the delays caused in the aviation industry due to a high volume of traffic and congestion in terminals. It has both positive and negative impacts in the sense that it can be beneficial in reducing delays and enhancing passenger experience while disadvantageous in the loss of airport tour guide and assistant jobs. On the positive spectrum, there are several advantages that Loomo provides if it were implemented as an airport concierge: 1) Effective airport navigation services to avoid confusion, stress, and congestion in terminals, 2) Language translation services that allow for cross-cultural communication, and 3) Luggage carriage functionality with added features. Outside of airport concierge activities, employees will have the capability to utilize the Loomo as a mode of quick transportation around the airport in case of emergencies.

On the contrary, the negative aspects revolve around the phenomenon that people often have when it comes to robotics: robots are gonna take over our jobs, lives, and world. Implementing Loomo as an airport concierge will result in a loss of jobs in the airport for tour guides and translators. However, this process only proves more beneficial to the airport due to reduced costs and more opportunity for growth in the technology field to be robot administrators for Loomo. Another negative aspect could be the loss of direct human-to-human interaction due to these robots, but it only makes it safer at the moment given the global COVID-19 pandemic. In the future, airports may want to re-consider the implementation and possibly think of having a hybrid structure with human tour guide and robot ones as well.

Most importantly, this project was built with hopes to add onto the fleet of autonomous robots that CVG has already implemented in their airport. Ottobots are the mobile robots made

by the Ottonomy company that are currently deployed at CVG and allow for autonomous delivery of food, beverage, and lifestyle products to passengers. Joining this fleet would be a bunch of Segway Loomos for additional passenger services. CVG Passengers can have the latest tech to have a Loomo guide them to their desired gate and have food or products delivered to them autonomously by an Ottobot. With hopes to automate the repetitive tasks that are present in major airports around the world, our team would like to further help innovate CVG's technological approach to combat the challenges posed by the COVID-19 pandemic.

Engineering Standards, Constraints, and Security

IEEE Engineering Standards

Software Life Cycle Process (IEEE std 12207)

The Software Life Cycle Process is a very important aspect of IEEE standards that our team followed. First, we met with our project's sponsor for an overview of the project and determined what the requirements were. The main requirement we wanted to meet was implementing Loomo navigatio using a grid-based solution. Once the team had a centralized idea of the project, we researched new possibilities and implementations that could complete the grid-based navigation. While finding solutions, we designed new navigation capabilities to reflect

improvements and changes. We continued this cycle of implementing and improving solutions until we were satisfied. We then began repetitive testing in a controlled environment in order to ensure that Loomo reacted as expected. We maintained ideas of additional functionalities to implement if the main requirements were met much earlier than planned.

Quality Assurance Plan (IEEE std 730)

Quality of the overall project and the code implemented with Loomo was extremely important. There is potential for Loomo to be used in a commercial setting with CVG, so standards of a quality, well-working device for real-world users are essential. To uphold the quality of the project, we strived to write readable, practical, and editable code. We communicated often with the CVG Person of Contact in order to ensure the project was meeting their standards. Additionally, we made sure to properly document and explain the current project and code in order to further help future teams if this capstone project continues in future semesters.

Verification and Validation (IEEE std 1012)

During the development of Loomo's autonomous navigation software, the team noticed issues with Loomo's ability to drive in a perfectly straight line. This is not such a big deal when scaled across a small testing area, such as the upstairs of Duthie, but can be seriously detrimental to accuracy when considered across an area as large as an airport. Once we noticed this issue, we were sure to inflate Loomo's tires to exactly the same PSI (pounds per square inch) and check them multiple times. We also measured over and over the testing area and the distances between obstacles, destinations, and the starting point. Once these values were set in stone, the tests we designed were run many times over the course of a few weeks to ensure the code executed

consistently without issue and Loomo traversed the area without deviation from safe generated pathways. Unfortunately, due to the retirement of Mr. Manny Adams, we were not able to get our hands on a detailed map of CVG's interior. However, we were able to verify based on personal experience inside CVG that the terrain and area were traversable by Loomo with our software guiding it.

System Requirements (IEEE std 1233)

There were a few system requirements for the Loomo project. Loomo can be compatible with Android and IOS operating systems, but in order to develop an application for Loomo, the program must be compatible with Android Studio. Additionally, in order to upload an application to Loomo or debug/test it, Loomo must be in Developer mode and there must be a hardwired connection to the device with the original application.

Testing (IEEE std 829)

The Loomo Airport Concierge project required little testing to get right: once the internal grid based system was discovered, we found it worked easily. However, it was quickly realized that the Loomo frequently veered off course, and had issues turning the correct amount. Loomo was run through a series of tests involving moving straight, moving a set distance then turning, going a long distance through many short bursts, going a long distance through one long burst, and even moving far away and then returning to its original location. As stated in the SDK, it deviated by about .25 meters on average, though it has been known to fly wildly off course for no reason, even when all the code did was make it move forward. It once, in a large hallway, made an almost 15 degree turn and ran into a wall after moving forward for a while. It was determined to be a hardware issue as the same code would be run from the same starting location

only to be off in a completely different manner. This 0.25 meter error is not a major issue in the large empty spaces of the airport, but within the small tight corners of the Duthie Center half a meter is in the wall. It seems Loomo can self correct larger errors, but it has trouble detecting the smaller ones. We also tested Loomo by putting it in a taped out pseudo-airport, with the offices of the Duthie Center acting as the airport gates. It was able to hit the marked "X" spots on the floor consistently, and it would always be within the .25 meters of the spot (though in which cardinal direction it was off by fluctuated). We always made sure to start it in the same space by lining its back wheels up with a door frame: we physically placed the wheels against the door, with the left wheel touching the door frame. It was as straight and consistent as we could get it. However, even with consistent starting locations it would still always be a little off course in random directions.

Constraints

Loomo has multiple built-in constraints, all of which come from Segway's own Software Development Kit. Including a lack of complete control over some of Loomo's exact movements and functions, Loomo's top speed has a cap imposed not by its hardware not being able to take it, but by the software preventing it from reaching above about 4 mph in robot mode and 11mph in riding mode. This is mostly for safety reasons as to prevent Loomo from running down someone at 60mph, but it does make its use as a fast guide limited. Another constraint that the group had to deal with was how Loomo accepted code and how to receive error messages: if a programmer wanted to see what error messages were sent, they had to plug a cord into Loomo and follow it around as it ran. Not the worst thing in the world, but not being able to program Loomo wirelessly makes updating it hard to do frequently. Another constraint Loomo puts upon those

who use it is the requirement of the SDK: there is no easy way of getting around it, which would be useful in cases such as having it go faster than its built in speed limit(s), turning off certain features and modes, and disabling buttons that could be used to take it out of guide mode or turn it to ride mode.

Security

As Loomo is built on the Android Studio framework, it has all of its security features and vulnerabilities built in. However, as Loomo will never be directly communicating with other computers, barring obvious attempts on the robot it should stay safe from any malicious code. However, Loomo currently has a very large security vulnerability: its physical form. Loomo can be picked up and moved with various ease, or knocked off course with a strong blow. This can lead to Loomo becoming lost and unable to orient itself, meaning it could end up running into walls, or getting someone lost as it is unsure of where it is. Furthermore, it can easily be turned off of the program being run on it through buttons intended to help users but now doing little more than preventing it from staying in app mode. Additionally, nothing prevents someone from just turning it off, or switching it over to ride mode and just riding it away. Loomo requires no passwords or verification to code it, just a wire and the correct language. Anyone with a laptop could hi-jack the robot. While its moving capabilities might not be the most useful in this situation, being able to see through its camera as a form of spying, surveillance, or scoping out could be a serious issue if left unchecked, especially as there is no easy way to lock programs out of it other than ending the program. Loomo also has been shown capable of being remotely controlled with controllers, making it capable of being controlled in real time with live video

feed. Again, there are no security measures in place (nor could they easily be put in) to prevent this or other issues.

Conclusions

By the end of the semester, the progress made to creating a foundation for the locomotion services using Segway's Loomo was successful. It provided an understanding of Loomo's internal grid-based navigation capabilities as well as insight into how powerful search algorithms can be in controlled environments. It also supplies future groups with a source to branch from, so they can add onto its concierge capabilities. Spending the time to develop the said foundation for Loomo paid off, as our group won Best Overall Project at the Engineering Design and Innovation Showcase. That being said, there are still improvements that need to be implemented into Loomo. Future groups that work with Loomo should focus on these issues, and strive to consider engineering standards, constraints, and security in every decision they make.

Regardless of the improvements, Loomo's contribution to society can be a major payoff. Working with CVG to further their fleet of autonomous robots helps passengers with disabilities or other incapacities create a more enjoyable travel experience. It can also help lost passengers find and successfully board their flight on time. This will also save the passenger money and make them more likely to travel through the CVG airport.

With Loomo's locomotion services being the key to supplying CVG with their passenger concierge robot, it needed to efficiently guide travelers to their respective gates. Pairing the locomotion application with the a* algorithm paid off tremendously, and worked flawlessly with Loomo's SDK after only a few adjustments were made. Future progression should be working towards the same goal. We, as a team, believe that Loomo can make its way into airports and serve its newly acquired purpose.

Recommendations for Future Work

Loomo can be improved upon in many areas. Our recommendations are for the next group to start by improving the locomotion first. Loomo's x, y, and z coordinates are set each time the base is reset, so they are constantly changing. To further explain, call Loomo's starting positive x North and positive y West. When Loomo takes its turn from rooms 205 – 206 to 208 – 209, Loomo's new x positive is West and y positive is South. If you run the application now, you will notice Loomo adjusts its base to always face looking in the same direction as the starting direction. This is because when Loomo travels backwards from rooms 208 – 209 to 206, instead of checking Loomo's current direction, the code becomes universal. Otherwise, Loomo would be at 208 with its positive x facing West and when directed to go back to 206, it will need to traverse in the negative x direction but is fed negative y coordinates from the a* algorithm. This is where things get confusing, and at almost every point that Loomo is no longer facing its initial direction, a new check would have to be run.

With all that being said, Loomo's coordinate system can be better handled than adjusting to face the same direction as it initially was. We experimented with creating properties that acted as flags for each time Loomo ended facing a different direction. This created a lot more work to evaluate and handle each flag, when we no longer had that time to do so. Another idea was to rotate the grid in the a* algorithm. Not a lot of experimentation went into this idea other than conceptualizing it, but by rotating the a* grid to match Loomo's direction could return the exact path Loomo needs to move in any direction correctly.

Our group tried to eliminate any hardcoding possible, but the starting and end point had to be. A way to improve this would be to introduce GPS capabilities in Loomo. Loomo can

connect to the internet, and GPS should be very accurate to determine Loomo's starting point on the grid. This is an area worth looking into, and whether that means to supply a future group with the research or full implementation, anything would help push Loomo's development further. With GPS, Loomo is capable of much more.

Loomo has no obstacle avoidance at this moment. However, it is well documented on Segway's website how to implement it. Due to Loomo's drifting issues, we had to adjust Loomo to give it as much room as possible to avoid running into walls. If Loomo's obstacle avoidance is implemented, Loomo can be smart enough to deal with its mechanical issues, as well as CVG's issues with avoiding non-stationary objects, such as humans. However, to successfully implement obstacle avoidance, we believe that Loomo needs GPS assistance. If Loomo must go off-track of its route, then Loomo needs to regenerate a new route to take from its current position to the ending. By utilizing GPS, Loomo may be able to capture the new initial point, that is currently only tracked by updates from when Loomo finishes its current route (when Loomo reaches a door, it gets a new initial point; otherwise, Loomo either doesn't know or incorrectly captures its new initial point). *

Smaller areas of future development could be developing a mobile application for Loomo, where a traveler could call it from their cell phone. Improvements to prevent theft by Loomo can be explored. For example, Loomo may travel faster than the traveler and in result steal their luggage. To prevent this, a fob could be developed to measure the distance between the traveler and Loomo. When the distance reaches its threshold, Loomo can either stop or slow down to allow the traveler to catch up.

* After discussions with professors at the Engineering Design and Innovation Showcase, GPS can have difficulties in an airport setting. I would still give the idea thought, but maybe think about the idea of beacons as well.

Appendices

Customer Contact Information

Manny Adams

madams@cvgairport.com

linkedin.com/in/brianpcobb

IT Support Supervisor

CVG Airport - Cincinnati/Northern Kentucky International Airport

Brain Cobb

bcobb@cvgairport.com

linkedin.com/in/manny-adams-67b0215

Chief Innovation Officer

CVG Airport - Cincinnati/Northern Kentucky International Airport

Additional Drawings and Diagrams

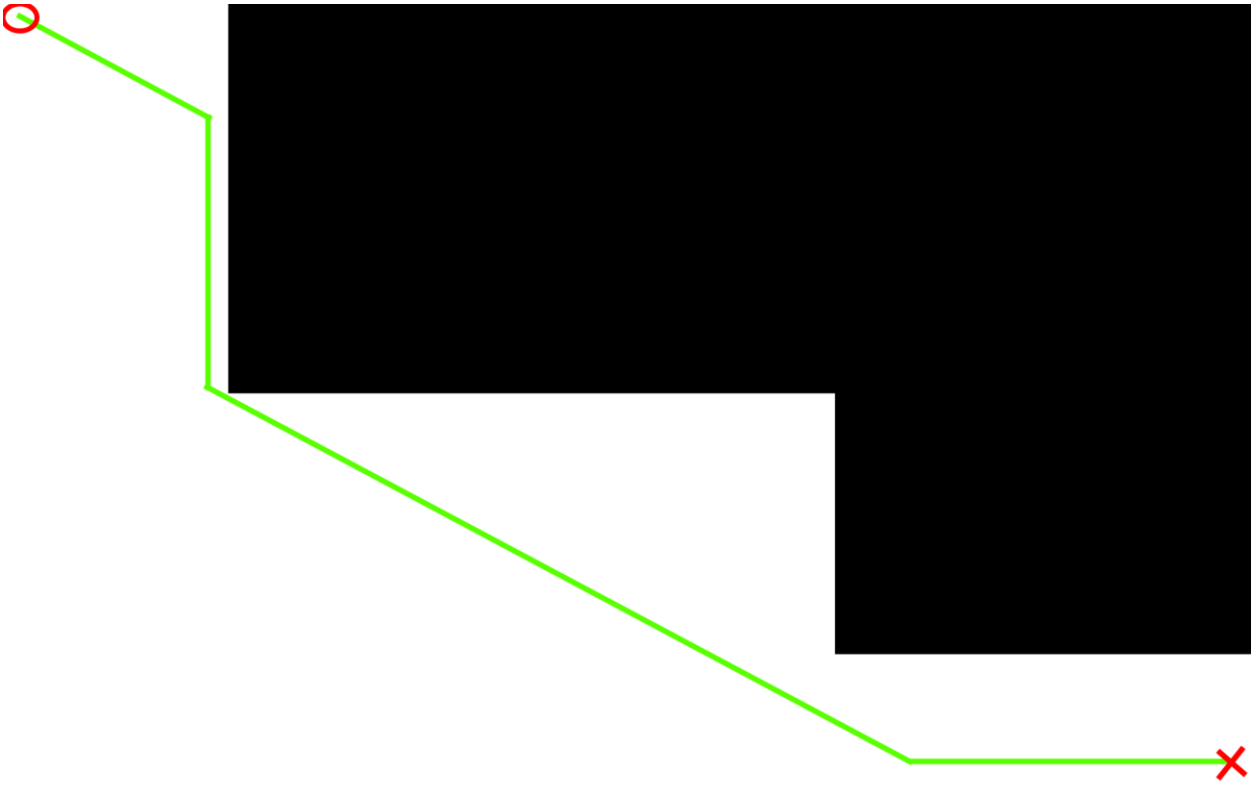


Figure 27: An unused map representing a path chosen by A Algorithm*

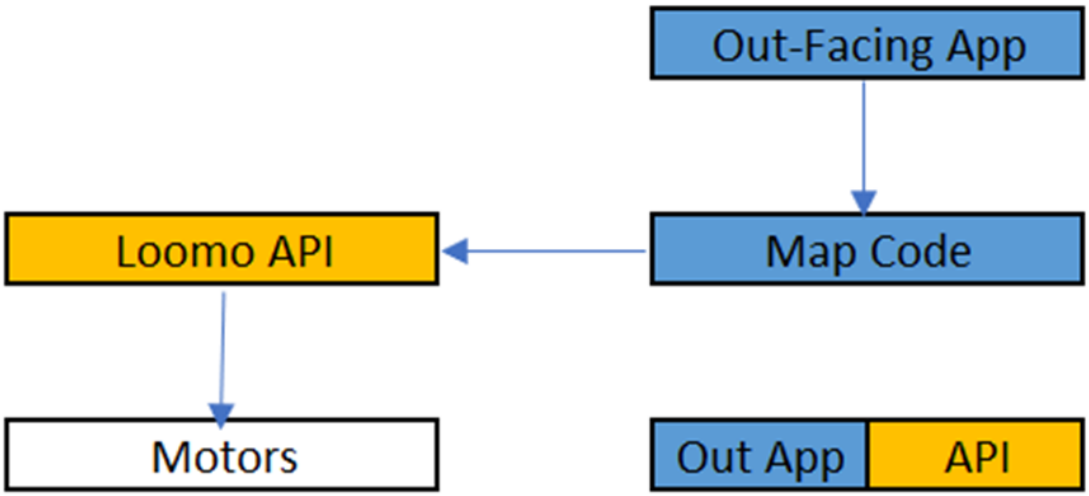


Figure 28: Interactivity Diagram

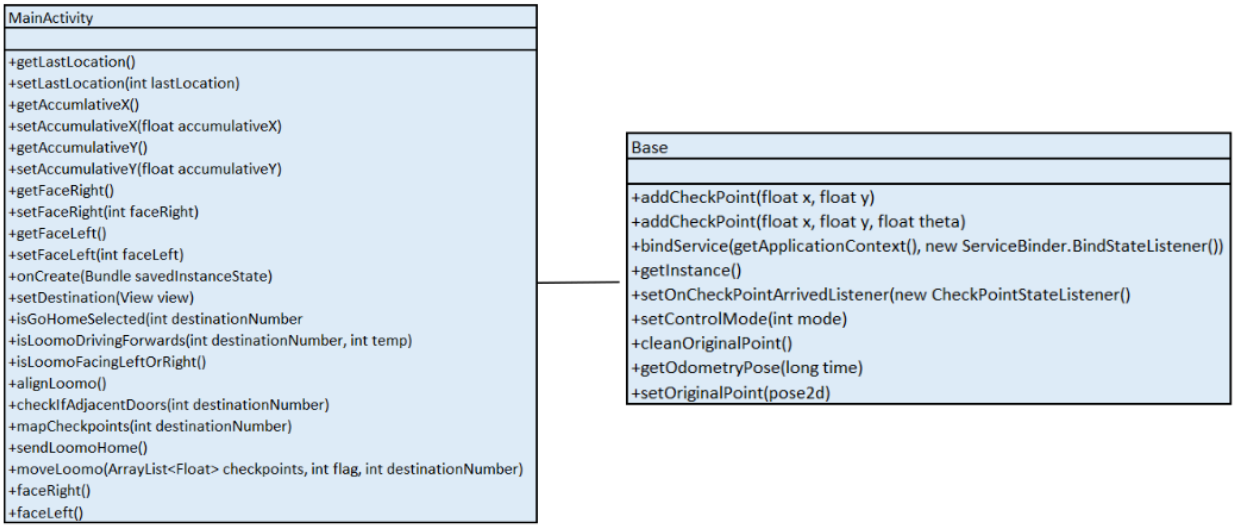


Figure 29: CVG-Demo1 Class Diagram

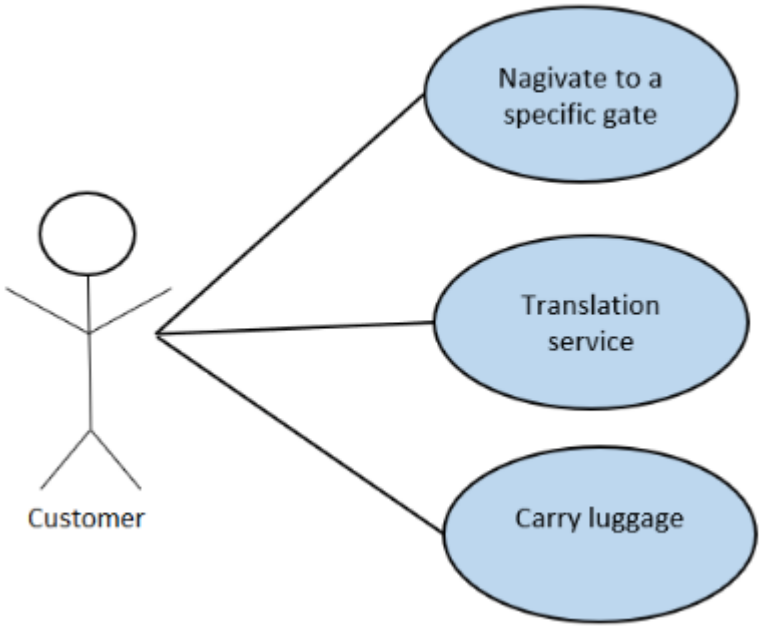


Figure 30: Use Case Diagram

Source Code (could be included in Electronic version if lengthy)

The source code for this project including the Loomo navigation application the A* algorithm can be found at the following GitHub repository.

https://github.com/lamiku01/CSE596_OKAY_LOOMO

Experimental and/or Simulation Test Results

The following figures show a simulated progression of Loomo when the A* algorithm returns an optimal path. In these figures, the simulated path shows how Loomo will choose to navigate from the Home position to Room 208.

Figure 31: Simulated Loomo Image 1: Loomo is at Home position. The path to be followed is shown by the bright green line. The red 'X' represents the destination

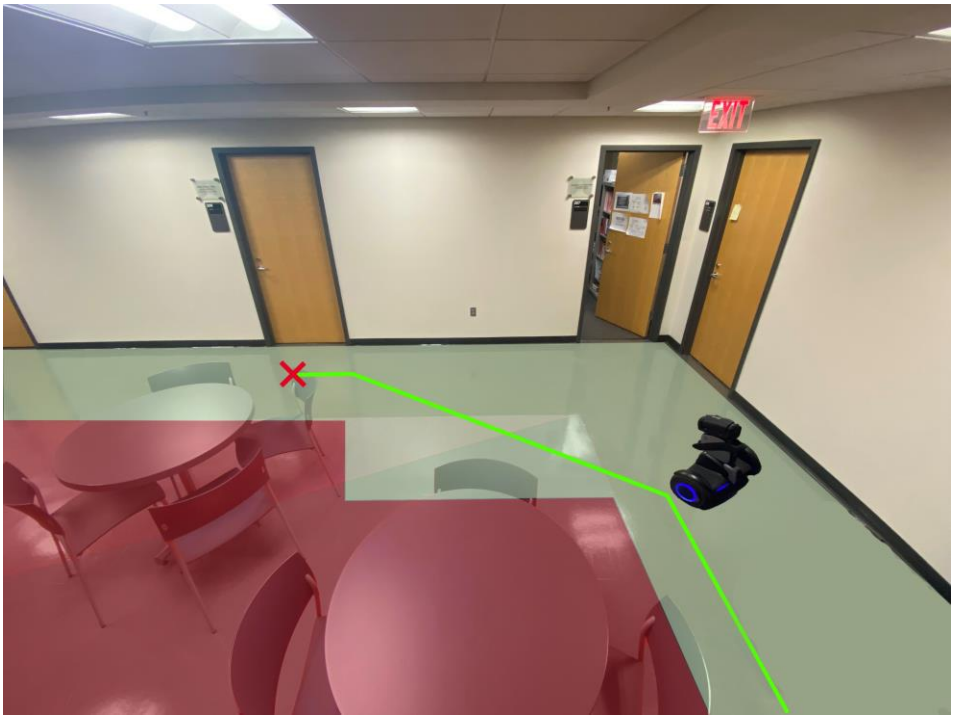


Figure 32: Simulated Loomo Image 2: Loomo is halfway to the destination. It decides to angle towards the destination since it minimizes cost

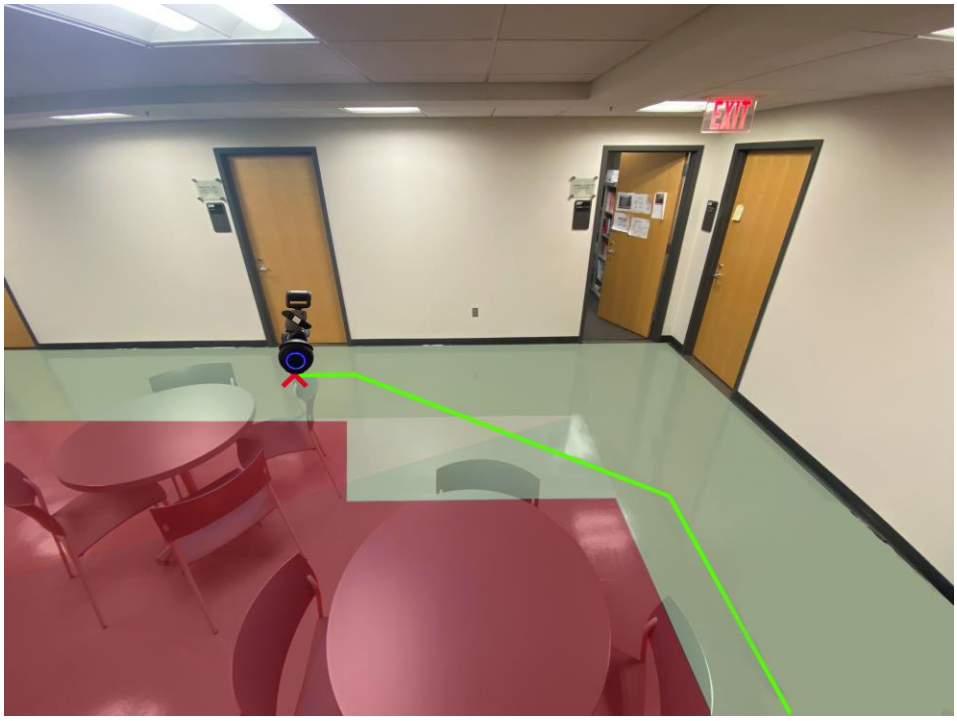


Figure 33: Simulated Loomo Image 3: Loomo reaches destination

Software Installation instructions

To install software onto Loomo, first create an application using Android Studio. Once the application is ready for deployment, connect your computer to Loomo via a usb-c cable. Loomo's usb-c port is located on the back of its head. Once connected, Loomo's device name will appear under available devices. Then, click the run button. This will install the current application onto Loomo, and begin running automatically. If you would like to debug the application, instead of clicking the run button, click the debug button. To close the application, tap the side of Loomo's head.

User's Manuals

To access the installed application on Loomo, first a user must connect Loomo to an internet connection. After connecting to the internet, the user must enter developer mode from settings. Once a user enters developer mode, the user can simply find the "CVG-DemoFinal" application on the screen. Clicking onto the "CVG-DemoFinal" application will launch the application and the user will be able to test the application.

References

"Create presentations, Infographics, Design & Video," *Visme*. [Online]. Available: https://www.visme.co/. [Accessed: 28-Apr-2022].

"CVG celebrates 75 years," *Home*. [Online]. Available: https://www.cvgairport.com/. [Accessed: 28-Apr-2022].

"Home," *Segway*, 12-Apr-2022. [Online]. Available: https://www.segway.com/. [Accessed: 28-Apr-2022].

"Northern Kentucky International Airport," *Wikipedia*, 30-Jan-2008. [Online]. Available: https://en.wikipedia.org/wiki/Cincinnati/Northern_Kentucky_International_Airport. [Accessed: 28-Apr-2022].

S. J. Russell and P. Norvig, *Artificial Intelligence: A modern approach*. Harlow: Pearson, 2022.

Segwayrobotics.com. [Online]. Available: https://www.segwayrobotics.com/. [Accessed: 28-Apr-2022].

Segway Robotics, "Loomo Product Information Guide" [Online]. Available: https://www.segwayrobotics.com/. [Accessed: 28-Apr-2022].

"Quick Guide For Android Tablet Ninebot VA50EC" [Online]. Available: https://fccid.io/2ALS8VA50EC/User-Manual/Users-Manual-3393307 [Accessed: 28-Apr-2022].

"Read 'future flight: A review of the small aircraft transportation system concept -- special report 263' at nap.edu," *3 - Air Transportation Challenges | Future Flight: A Review of the Small Aircraft Transportation System Concept -- Special Report 263 |The National Academies Press*. [Online]. Available: https://nap.nationalacademies.org/read/10319/chapter/5. [Accessed: 28-Apr-2022].

"Autonomous Mobile Robots in airports," *YUJIN ROBOT*, 20-Dec-2021. [Online]. Available: https://yujinrobot.com/how-todays-challenges-are-driving-demand-for-amr-in-airports/. [Accessed: 28-Apr-2022].

Ottonomy, "Creating a contactless travel experience: Ottonomy launches first fully autonomous delivery robots at CVG Airport," *Creating a Contactless Travel Experience: Ottonomy Launches First Fully Autonomous Delivery Robots at CVG Airport*, 16-Dec-2021. [Online]. Available: https://www.prnewswire.com/news-releases/creating-a-contactless-travel-experience-ottonomy-launches-first-fully-autonomous-delivery-robots-at-cvg-airport-301446573.html. [Accessed: 28-Apr-2022].

E. DeLetter, “CVG Airport now has Food Delivery Robots,” *The Enquirer,* 19-Dec-2021. [Online]. Available: https://www.cincinnati.com/story/news/2021/12/16/cvg-cincinnati-airport-ottobot-food-delivery-robots/8925637002/. [Accessed: 28-Apr-2022].

Stone Barrett

CSE 596 - Capstone

“Okay Loomo”

Barrett, Stone; Malyala, Karthik; Mikula, Lauren;

Patel, Saurin; Secor, Alex; Wedding, Donald

Overview of task assignment and project activities

Throughout the Spring 2022 academic semester, I completed the Capstone design course for the Computer Science Engineering (CSE) undergraduate degree at the University of

Louisville's J.B. Speed School of Engineering. In January, a list of projects to choose from was given to students in the course. All of these projects involved a company to provide hardware, access, or other necessities, a categorization of disciplines needed, and a real-world applicable need that could be met through experimental design of some sort. The projects included in this list were all very interesting. There was a project for a pizza company's automated inventory system, there was a project for software to control drone swarms, and even a project for an automated lawn mower. There was, however, one project that piqued my interest more than the rest. This project was in collaboration with the CVG airport in northern Kentucky, and the description ran as follows:

> "Airport Concierge Description With people entering the airport from around the world, how can CVG utilize a robot to provide a concierge service. Anything from language translation, navigating an airport, and/or to providing an extra set of hands by carrying your bag. This project will utilize a Segway Loomo and the open architecture through the Android SDK. Benefits 1. Provide research and development services to the sponsor 2. Deliver a language translator to help with the international passengers 3. Transport passengers to their destination throughout the airport Goal Train CVG's Segway Loomo to interact with passengers through language translations and navigation throughout the airport."

Segway Robotics is the controlling company of the well-known transportation device producer, Segway. While Segway focuses on human-controlled personal electric vehicles, Segway Robotics develops intelligent software to guide their own robotic transportation devices. According to their website, Segway Robotics was founded in 2012 and has its headquarters located in Beijing, China. The Loomo Robot is a "self-balancing transporter" that runs on a

custom build of the Android version 5.1 (also known as Lollipop) operating system. Loomo can follow a person using computer vision, record video and take pictures, respond to voice and gesture commands, and be ridden by standing atop the two foot platforms. Loomo's carrying capacity is 220 pounds (lbs) or 19 kilograms (kg); its battery range is 30 miles (mi) or 48 kilometers (km). Loomo is IPX4 splash proof rated and can traverse multiple types of terrain. Loomo weighs 42 lbs or 19 kg. Loomo sports a 4.3 inch (in) liquid crystal display (LCD) where users can interact with the operating system using touch controls. Loomo recognizes gesture controls and perceives information through two cameras – one is a high-definition 1080p camera with 30Hz streaming refresh rate and a 104 degree field of view (FOV) and the other is a 3-dimensional camera from Intel called the RealSense ZR300. The former camera is used for recording video and taking pictures while the latter is used for sensing depth in 3D space and tracking motion. Loomo also perceives its environment using multiple ultrasonic sensors, infrared distance sensors, touch sensors, and other monitoring tools. Loomo operates using an Intel Atom x7-Z8750 central processing unit (CPU). This processor has 4 cores running at 2.56 gigahertz (GHz) on an x86-64 architecture and integrates the Intel HD Graphics 405 graphical processing unit (GPU). Loomo carries 4 gigabytes (GB) of random access memory (RAM). Loomo's digital storage capacity is 64 GB. Loomo can travel at a maximum speed of 4.3 miles per hour (MPH) or 8 km per hour (km/h) when in "robot mode" and 11 MPH or 18 km/r when in "self-balancing vehicle mode". Loomo can travel up and down slopes only less than 15 degrees of inclination, roll over obstacles only 1 centimeter (cm) or less in height, and cross gaps only 3 cm or less in width. Loomo has a microphone array of 5 microphones that allows for beam-forming which enables voice localization and voice command recognition. Loomo is available for purchase from Segway Robotics' website for $2299.99 United States Dollar (USD) –

however, as stated previously, the CVG airport would be providing this hardware to our team. (Segway)

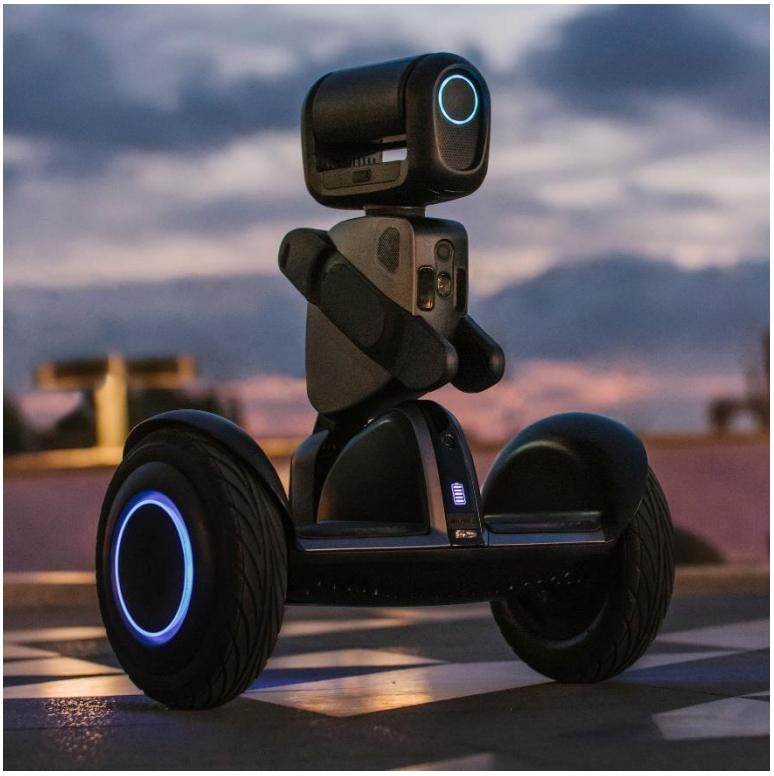
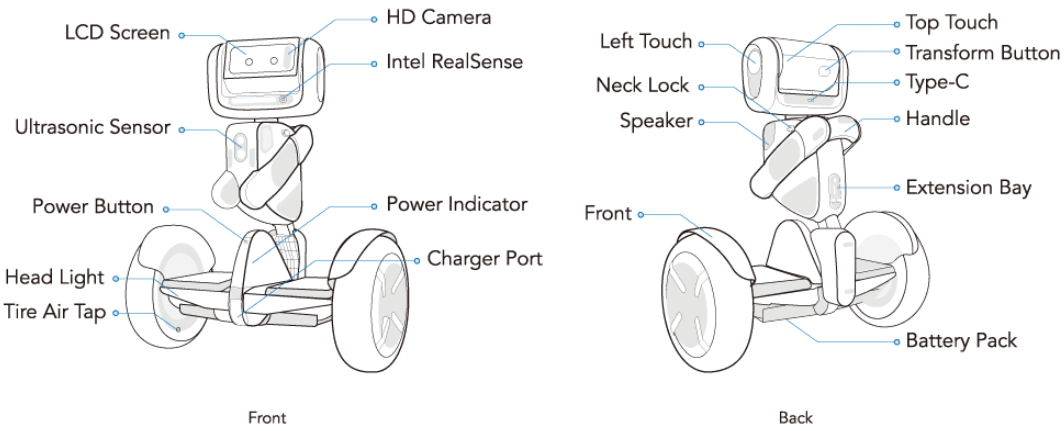


Figure S.1: Loomo product photo from Segway Los Angeles



Hardware Specs	Descriptions
Dimension	Height 0.64m, length 0.57m, width 0.28m
Weight	17.5kg, including the battery pack
Battery Capacity	310Wh
Speed Limit	8 kilometers per hour (software limit for the Alpha edition)
Pan Tilt Unit Range	Pan ±150 degrees, Tilt -90 ~ +180, zero is horizontal facing front
LCD Screen	4.3 inch

Platform Specs	Descriptions
Processor	Intel Atom Z8750, 4 cores, 2.4GHz, x86-64 architecture
Operating System	Customized system based on Android 5.1
Memory	4 GB
Storage	64 GB
USB port	USB 3.0 Type-C cable (compatible with USB 2.0)
Extension Bay	USB 2.0 and 24 voltage power (1A limit)

Sensor Specs	Descriptions
RealSense	Real time 30Hz RGB-Depth streaming for both indoor and outdoor uses
HD Camera	High resolution with 104° FOV (Pending fine turning. Image quality is not optimal)
Mic Array	Beamforming and voice localization, powered by 5 microphones
Ultrasonic Sensors	Obstacle distance calculation
Touch Sensors	Located at robot head left, right & back
Wheel Odometry	Approximately 4 degrees precision
Base IMU	6-axis IMU

Figure S.2: Loomo diagram and specifications sheet from Segway Robotics

The Cincinnati/Northern Kentucky International Airport is located in Hebron, Kentucky. According to Wikipedia, the code "CVG" was derived from the name of a nearby city called Covington. CVG was opened for commercial flights in 1947, and has an interesting origin story:

> "President Franklin D. Roosevelt approved preliminary funds for site development of the Greater Cincinnati Airport on February 11, 1942. This was part of the United States Army Air Corps program to establish training facilities during World War II." (Wikipedia)

While the history of the location is certainly a topic for discussion all on its own, CVG's efforts to cement its place in the future are where the Capstone project comes into place. In recent years, CVG has undertaken multiple projects that each have the goal of modernizing its customer experience, including food delivery robots that serve customers their remotely ordered food anywhere in the airport. On top of this achievement, CVG also wanted to solve other customer experience shortcomings through the use of robotics. One of these shortcomings was the commonality for passengers to get lost in the facility. Being a relatively large airport at 7,700 acres in land area (BizJournals), passengers tend to have a difficult time finding their way to their correct gates in a timely manner. With former CVG IT Support Supervisor Manny Adams at the helm, the airport began searching for cost-effective solutions to this conundrum. This led Mr. Adams to his friend, machine learning researcher, and former Chair of the CSE department at the University of Louisville Dr. Adel Elmaghraby.

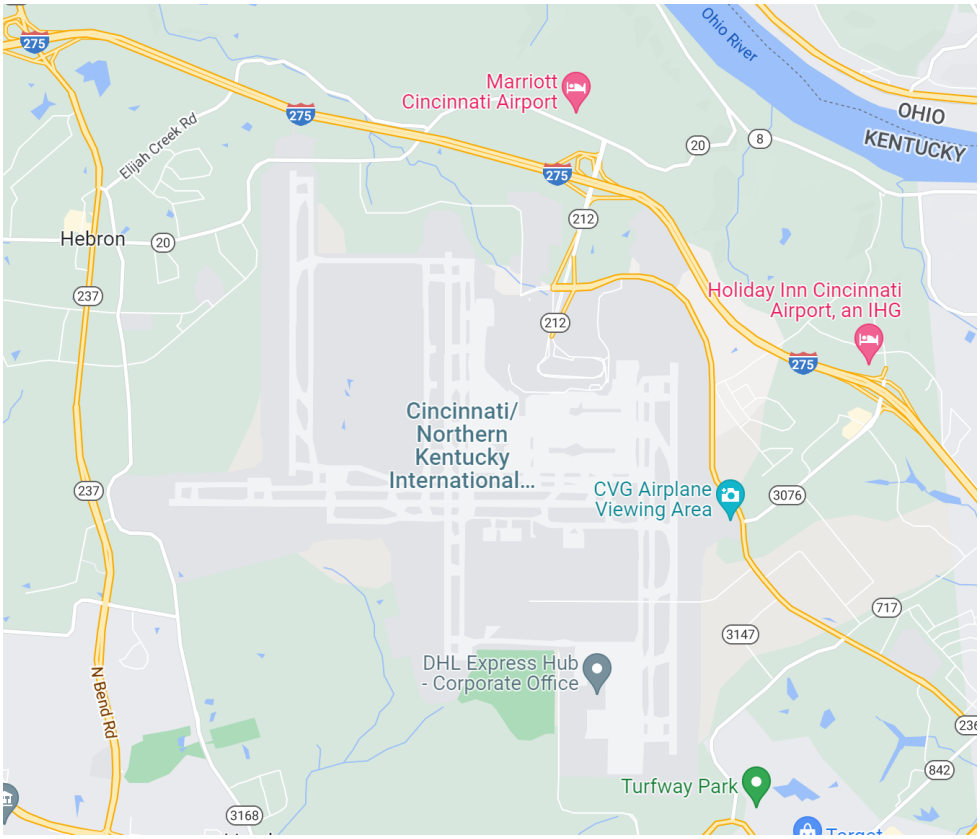


Figure S.3: Google Maps view of CVG Airport

Figure S.4: Google Maps Street View inside CVG, also available on CVG virtual tours

Dr. Elmaghraby worked with the J.B. Speed School of Engineering's Industrial Research and Innovation department to get the project integrated into CSE 596 as a Capstone project, where it still resides now. In keeping with the robotics approach CVG had already implemented to some degree, Mr. Adams and Dr. Elmeghraby agreed that a Loomo robot from Segway Robotics would be the right choice of platform to prototype the solution with. One of the first hurdles to overcome in this project was interlingual communication. No one involved wanted Loomo to only be capable of assisting English-speaking customers. Since Loomo's default configuration settings only allow one language for interaction at a time, the first objective was made clear: integrate real-time translation so that anyone can be assisted by Loomo. This objective was worked on by a previous team in a different semester of CSE 596. The team that worked on it left their work for future teams just as we have. Because of this, when we picked up the project, we were able to see what they had accomplished and even given access to their source code in the event we want to use it or improve upon it. After reviewing their work, we decided that translation for spoken and written language had been accomplished to a suitable degree using the Google Translate Application Programming Interface (API). What needed improvement in this area was the implementation and user interface, but we decided that this could be put on the back burner while we focused on another objective. Dr. Imam suggested the same, saying that our team should choose another goal to start from scratch on. Remembering the project description given to the class by Dr. Park and Dr. Imam, we found our objectives and laid out our plan.

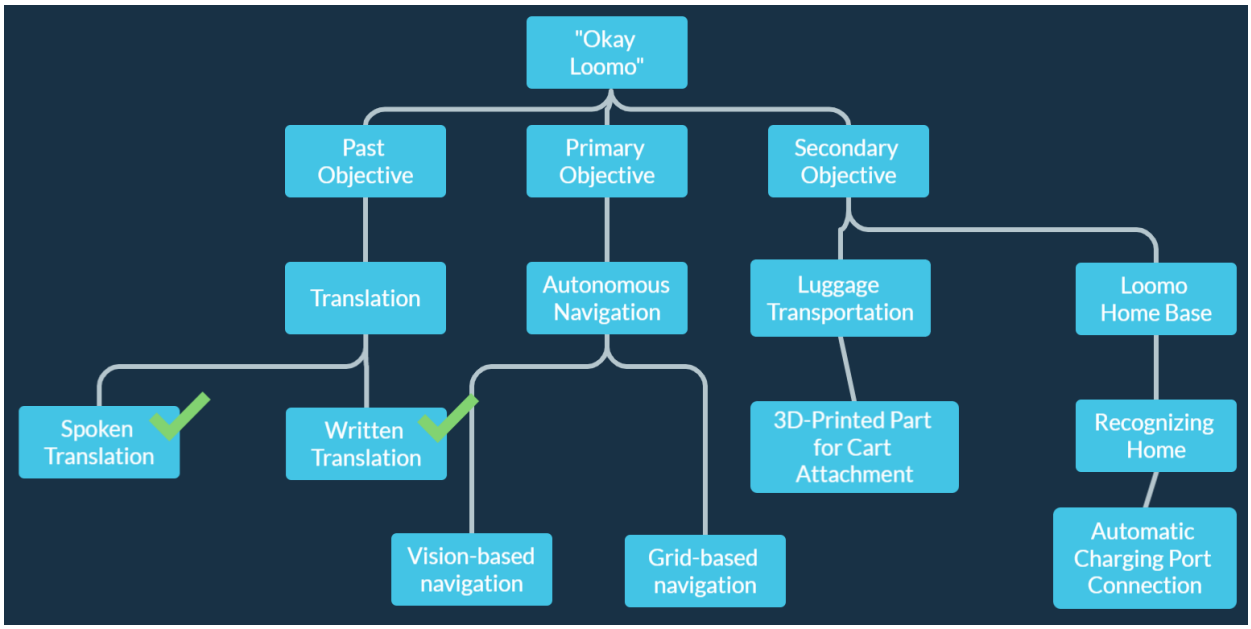


Figure S.5: Objective tree at the beginning of the semester (Visme)

Figure S.5 is an overview of the project objectives. The Past Objective branch details the goals of the previous group that worked on translation. The Primary Objective branch details the main goal of our team to reach by the end of the semester. This branch splits, which will be discussed further on. The Secondary Objective branch details the goals of our team to reach by the end of the semester IF and only if the Primary Objective branch were completed with much time remaining.

For the sake of this section, the goal of the project was to develop self-driving capabilities for Loomo. As the robot stands in production by Segway Robotics, it can follow a person. However, a person cannot ask Loomo to guide them. CVG wanted a way for Loomo to know where a location (either an airport gate or amenity) is and the fastest way to get there, as well as be able to guide the person there that asked for help. Also, as a side note: “Okay Loomo” was chosen as the name of our project because that is the phrase by which a person can get Loomo’s attention.

In the end, the primary objective was achieved using the grid-based navigation approach. About halfway through the semester, we discovered that Loomo had its own internal grid system that could be used when giving instructions through the SDK. A vision-based approach COULD be achieved to a degree, given that Loomo's processor does have an integrated GPU, but that would have required more time than the semester allowed for. The grid system made it very easy to program into Loomo a map of whatever environment one may wish to navigate. The grid system also allows for Loomo to recognize "home" or where it came from. If a Loomo doesn't start at a home base, however, it could be programmed to a "home" location instead of the grid's default (0,0). Another thing that our group found would be possible but did not have time to accomplish was obstacle detection interrupts. While Loomo's grid can be programmed with permanent obstacles to avoid and a path can be created with them in mind, some obstacles are not permanent or predictable. Some obstacles, like a person passing in front of Loomo or a wet floor sign, show up unexpectedly. To avoid this, a future group could implement in interrupt into our existing navigation code that uses the sensors on Loomo to detect objects that are not already programmed into the map. The interrupt could make Loomo stop and wait for a while or maneuver around it by constructing a new path with the obstacle now programmed in place. All of this being said, the most important aspect of our accomplishments this semester is how the paths are created by the software.

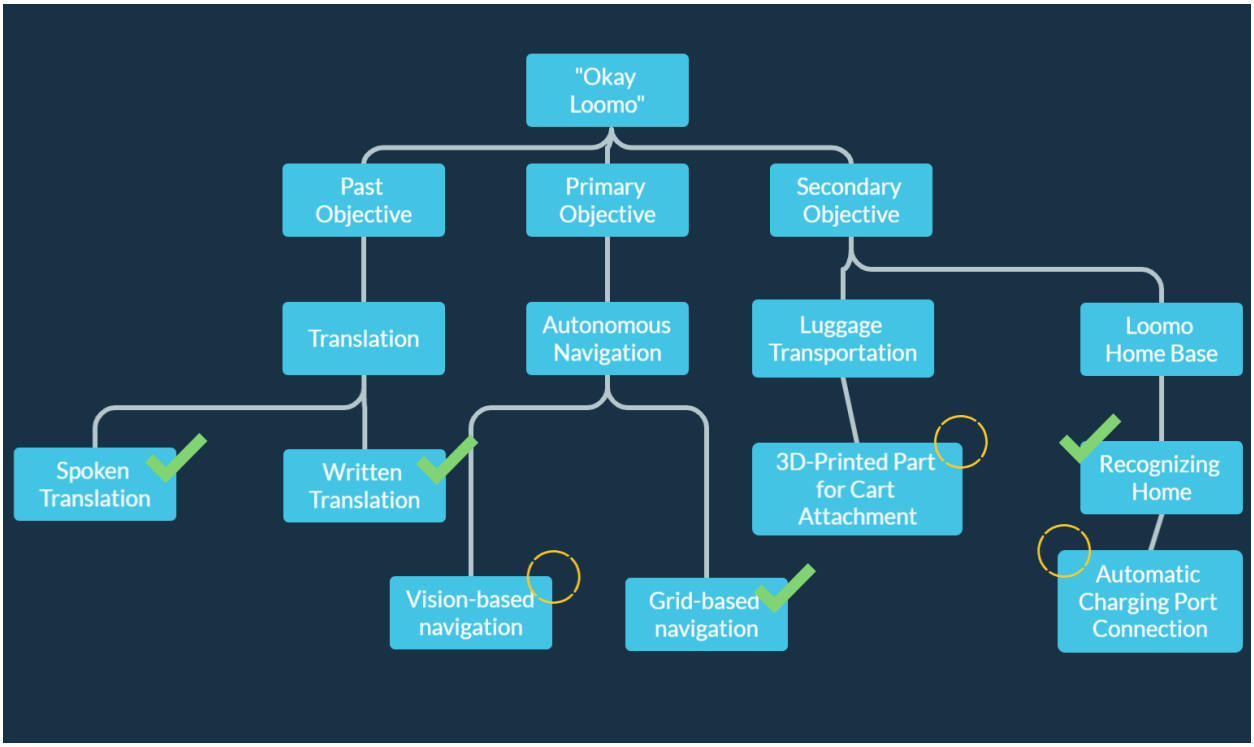


Figure S.6: Objective tree at the end of the semester

Pathfinding is done via the AI search algorithm known as A*. This algorithm uses a function to calculate the distance to the nearest next node, then estimates the Euclidean distance to the goal node, rinse and repeat. I picked up knowledge of this algorithm and how it can be used in a previous semester during the course CSE 545: Artificial Intelligence. In that course, we studied and ran many experiments using lots of different algorithms that can be implemented in path-finding, and due to its simplicity and low computational resource usage, A* seemed like the correct candidate for this project.

$$f(n) = g(n) + h(n)$$

Figure S.7: A Search Algorithm*

Areas of self-development and new learning that were needed/attempted

"Okay Loomo" was a project that had a very middling learning curve. I say this because it brought together many elements of things I have learned throughout my time as a Speed School student but required taking them the extra step. I was familiar with the Android Application Package (APK) we would use in creating the application that would run our navigation software and interact with users. I was familiar with artificial intelligence (AI) search algorithms that could be used in a self-driving vehicle environment. I was familiar with the Java programming language that the Loomo Software Development Kit (SDK) is written in. What I was not familiar with, however, was bringing these things together and implementing a robust solution in an environment where real people and customers would be interacting with and relying on a physical device. Most of the personal learning on my part as a result of this project involved being willing to try integrating components. For some reason, it's always been difficult for me to tie concepts together regardless of how well I know the individual concepts. But, with the help of my teammates, I was able to overcome that adversity for this project.

Outline of specific contributions including portions of code and/or hardware if relevant

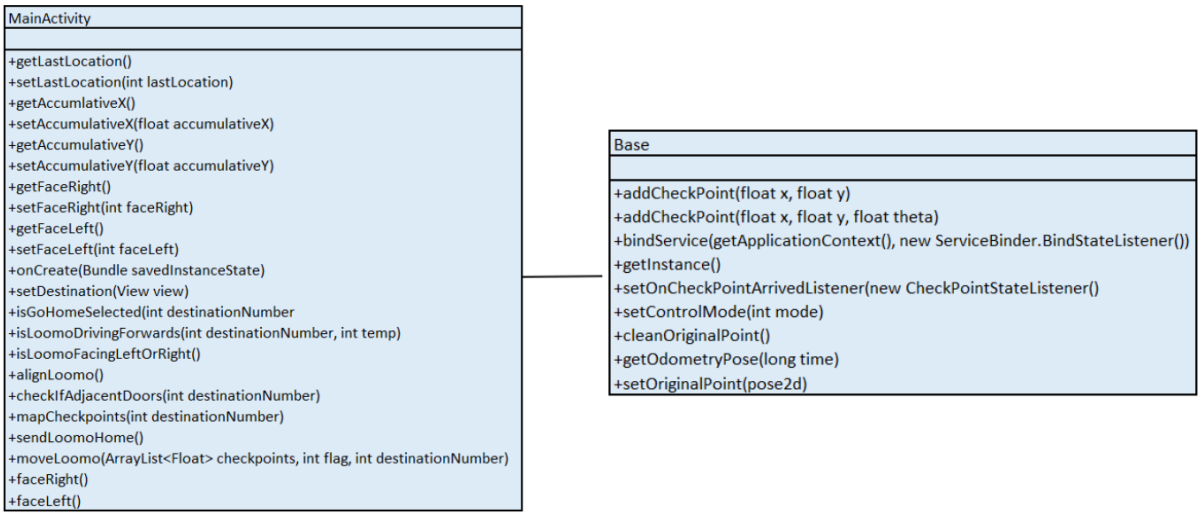


Figure S.8: Class diagram of navigation software onboard Loomo

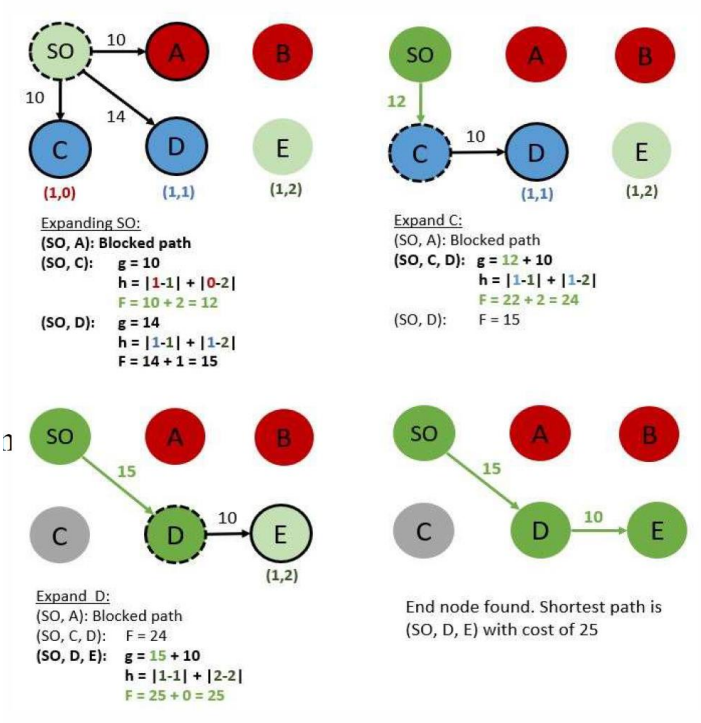


Figure S.9: Example of A Algorithm creating path*

As stated previously, my involvement dealt with researching the methods needed to autonomously navigate on the Loomo platform as well as their implementation in Java since Java

was necessary. I was also heavily involved in physically measuring Loomo's testing areas and verifying the paths Loomo constructed as valid.

How does your project relate to trends and emerging areas of CECS?

Machine learning, artificial intelligence, self-driving vehicles, and "smart" devices have been buzzwords in the world of engineering and computer science for years now. However, even though there is a lot of hype around them and they have fallen short of hopes or expectations a more than a handful of times, the technology is slowly catching up to the wants and needs of customers and engineers alike. Intelligence in computational devices isn't something I think about so much as a "trend" since its been an idea in literature for centuries and a target of engineers and philosophers for decades. More capable and more intuitive devices are always being pushed for and that is exactly what our team has contributed to the field. While the device itself was created by Segway Robotics, the software that allows it to guide and help others was designed by our Capstone team.

References

Bizjournals.com. [Online]. Available: https://www.bizjournals.com/. [Accessed: 28-Apr-2022].

"Create presentations, Infographics, Design & Video," *Visme*. [Online]. Available: https://www.visme.co/. [Accessed: 28-Apr-2022].

"CVG celebrates 75 years," *Home*. [Online]. Available: https://www.cvgairport.com/. [Accessed: 28-Apr-2022].

"Home," *Segway*, 12-Apr-2022. [Online]. Available: https://www.segway.com/. [Accessed: 28-Apr-2022].

"Northern Kentucky International Airport," *Wikipedia*, 30-Jan-2008. [Online]. Available: https://en.wikipedia.org/wiki/Cincinnati/Northern_Kentucky_International_Airport. [Accessed: 28-Apr-2022].

S. J. Russell and P. Norvig, *Artificial Intelligence: A modern approach*. Harlow: Pearson, 2022.

Segwayrobotics.com. [Online]. Available: https://www.segwayrobotics.com/. [Accessed: 28-Apr-2022].