Spring 2018

Navigational Heads-Up Display

Alex Walenchok
The University of Akron, atw29@zips.uakron.edu

Nicholas Seifert
The University of Akron, nms84@zips.uakron.edu

Joshua Reed
The University of Akron, jar207@zips.uakron.edu

Joshua Humphrey
The University of Akron, jch100@zips.uakron.edu

Navigational Heads-Up Display

Senior Design Project Final Report

Design Team: 3

Alex Walenchok

Joshua Humphrey

Joshua Reed

Nicholas Seifert

Faculty Advisor: Dr. Nghi Tran

April 30, 2018

Table of Contents

Table of Figures

Table of Tables

1. Abstract

One problem drivers face is distraction from looking at their mobile device while navigating rather than watching the road. This problem can be solved with a heads-up display placed directly on the driver's windshield. By using a mobile device with a custom GPS application, the following design will be able to send GPS data to a device that will display navigational information on a car windshield. The design includes two primary components, a mobile device and a System Unit, where the System Unit is composed of a portable power supply, a single board computer, and a display. For the design, the mobile device is an android device, the portable power supply is a battery, and the display is a small projector. The design has a very strong software focus, and the main intention of the design is to produce a fully functional Android mobile application that pairs along with prototype hardware elements. In final implementation, the mobile application sends data in the JSON format over a Bluetooth connection. The key features of the project are as follows:

- A custom GPS Android Application that can generate routes from the user's current location to destinations entered by the user
- A System Unit, which is a device that has the capability to be mounted on a vehicle's dashboard, connect to the user's mobile device, and generate a display that contains navigation information
- A projected display that is shown on the vehicle's windshield

2. Problem Statement

2.1 Need

(AW)

Modern mobile technology is often a distraction for drivers. One major source of distraction is mobile navigation applications (Google Maps for example). While mobile navigation applications are great tools for drivers, they sometimes require drivers to take their eyes off of the road to see information such as the next instruction, the distance to the next instruction, or the distance to their destination. Most, if not all navigation applications offer some type of voice navigation so that the driver can simply listen for the next instruction. However, if the driver mishears the instruction or forgets what the next instruction is, then they still must take their eyes off the road to look at their screen.

One possible solution to these types of problems are mounts that either stick to the car's dashboard or clip to an air vent, that way the mobile device is at least close to eye level. These, however, are not perfect solutions because the driver's view is obstructed, or the clip is placed in a location that diverts the driver's focus from the road.

For these, and many other reasons, there is a current need for a user friendly and efficient way for drivers to view navigation and current trip information without having to completely divert their gaze from the road.

2.2 Objective

(AW/NS)

This project's goal is to provide a way for drivers to view navigation information and trip information without having to divert their gaze from the road in front of them. This information will be present in a heads-up display (HUD) directly on the user's windshield.

A mobile application will also be required to communicate with the display. This application will contain a map, as well as the list of navigation instructions. As the user begins to navigate toward their destination, the app will be sending real time instructions to the HUD display system for the instructions to be processed and displayed on the windshield.

A major design goal that will be associated with this project is making a display that fits all angles of windshields. There are some solutions that currently exist to display navigation instructions on a windshield, but this project intends to improve upon that existing technology by producing a HUD that can be mounted and displayed on any kind, or shape, of windshield. This means it will produce a clear image on any windshield without requiring any sort of manual adjustment.

2.3 Research

(NS/JR)

The dangers associated with distracted driving are very well known. However, using GPS is often looked over as a risk because of how directly it is associated it with driving. When traveling on the highway at 55 mph, a driver travels the length of a football field in only five seconds [4]. This means that what seem like quick glances onto a car mounted GPS or mobile phone become very risky when traveling at high speeds. The proposed design intends to solve this problem. With GPS data placed directly on the windshield, drivers will be spending much less time with their eyes off the road and missing the audio for an instruction won't cause the

driver to become completely distracted with their navigation. A HUD placed directly on the windshield can be a seamless part of the driving experience, not another distraction.

(JR)

A study from Carnegie Mellon University had 24 subjects, 12 elder and 12 younger drivers, participate in a virtual driving evaluation using a display on the windshield, and then the same evaluation using a more standard GPS device. Results from this study showed that the drivers using a display system similar to the proposed idea have significantly fewer navigation errors and divided attention related issues when compared to using a more standard GPS device [3]. This means that there is already evidence that such a device could be useful to drivers in the real world.

(AW)

The proposed design requires communication between the user's mobile device, which will be providing the navigation, and the proposed device, which will be displaying the navigation and trip information. After taking an in-depth look the four main wireless communication protocols (Bluetooth, UWB, ZigBee, and Wi-Fi Direct), we found two viable options for the design. The first option would be to use Bluetooth connection, and the second option would be using a Wi-Fi Direct connection. Both Bluetooth and Wi-Fi Direct meet the design's power consumption requirements, but there are several other factors that must be considered, such as transmission time, data coding efficiency, complexity, and other performance metrics [6].

(JR/JH)

One of the key elements of the proposed design is the mobile application that will be used to communicate with the System Unit. This application will need to be fast and will need to

consumer a low amount of power. Using JavaScript Object Notation (JSON) to parse through the data will be one way to speed up processing time for the application. JSON is an extremely efficient data-interchange format that allows for easy storage of Key-Value pairs. A big advantage of JSON is its usability for programmers while remaining an easy structure for computers to handle. According to a case study done by Montana State University, JSON is much faster and less resource intensive option to transmit data objects [8].

On the power consumption side, there are several factors to consider while developing a mobile application, specifically to ensure low power consumption. The article "Native or Web? A preliminary Study on the Energy Consumption of Android Development Models" details some of these factors while attempting to provide methods for improving energy efficiency. In Android development, the two main programming languages used are Java and JavaScript. The article concludes that to minimize power consumption by an application, a hybrid solution using both Java and JavaScript, provides the largest improvement in energy efficiency [9]. This is something that will be considered while implementing the proposed design to provide the best possible user experience.

(JR)

The paper "Perceptual Issues in Augmented Reality" describes some of the challenges that may be faced when designing the Navigation HUD. Within this article the research goes on to specifically discuss projector-camera systems. Depth distortion is one of the most common problems with Augmented Reality. This will not be a problem for the Navigation HUD. Most navigation Augmented Reality overlay the entire windshield while highlighting the upcoming streets needing to drive towards. This would raise the concern for potential depth issues. The design suggested in this proposal will be displaying directions in the corner of the windshield

which will help eliminate most risk for having depth issues. Another issue presented was visibility of the image, specifically dealing with colors. Different light changing conditions and the changing weather patterns could hurt the quality of our image [5].

(JH)

While the initial plan for designing the HUD is to use projection for the display, it may also be possible to achieve the same result using an LCD. Thin film transistor Liquid Crystal Displays are still relatively common despite the recent technological advancements in the last few years. Because of their low power use relative to other higher resolution alternatives this type of screen makes a very good fit when designing an embedded system that requires a display component. Although regular LCD screens are usually not paired with a backlight the thin film transistor works very well with illumination from directly under the screen. This allows for an embedded system to display information in many different conditions such as darkness or stormy weather. This could serve as a potential alternative for the HUD design [1].

(AW)

One of the key elements of the proposed design is the mobile application that will be used to communicate with the System Unit. Also, one of the main objectives/marketing requirements is that the mobile application will have low power consumption. There are several factors to consider while developing a mobile application, specifically to ensure low power consumption. The article “Native or Web? A preliminary Study on the Energy Consumption of Android Development Models” details some of these factors while attempting to provide methods for improving energy efficiency. This article focuses mainly on Android development, and how the two main programming languages used in Android development (Java and JavaScript) can be used to either increase or decrease an application’s power consumption. The authors found that a

hybrid solution, which uses both Java and JavaScript, provides the largest improvement in energy efficiency [9]. This is something that will have to be considered while implementing the proposed design to provide the best possible user experience.

2.4 Patents

(NS)

A patent search for similar design ideas yielded several results. Patent #US6735517B2 shows a very basic navigation display on a windshield with only a single arrow as part of the HUD. The idea suggested in this proposal differs from this design because the proposed design will display more than just the next navigational instruction as part of the projection. Also, this patent does not describe the functionality of the projecting device. A large portion of the design in this proposal is the projector's ability to automatically adjust to different windshields in order to produce a clear picture, which is not mentioned in this patent [2].

The patent search also provided U.S. Patent 20130051615, which is for an "apparatus and method for providing applications along with augmented reality data". This apparatus works by "seeing" objects via an augmented reality unit. The object data that is gathered by the unit is processed to see if the object is recognizable. Once the object is recognized, a search term (tag) is created. This tag is then utilized to distinguish which application must be opened on the mobile terminal. The application will then "utilize tag information in response to the application being executed." Then, whatever application functionality is associated with that particular word is displayed on the augmented reality screen [7].

2.5 Marketing Requirements

(AW/NS)

Clarification on terminology used in the marketing requirements:

- "Navigation Information" refers to the next navigation instruction, the distance to the next instruction, and the name of the street involved in the next instruction.
- "Trip Information" refers to the current weather conditions, warnings regarding road hazards such as icy roads or accidents ahead, and an indication of how much gas is remaining in the vehicle.
- "HUD" refers to a "heads-up display" that is projected onto the car's windshield. This display will contain the navigation and trip information.

Marketing requirements for this project:

1. The mobile application used to interface with the System Unit will work on Android devices.
2. The device will be powered by a battery that is held within the device.
3. The device's battery will be rechargeable.
4. The display will be unobtrusive and easily readable
5. The entire device will be cost efficient and affordable.
6. The display will contain accurate, and desirable, navigation information.
7. The display will contain accurate, and desirable, trip information.
8. The device will be small in size and will attach directly to the vehicle.
9. The display will be visible regardless of external conditions (e.g. sunny, cloudy, etc.).
10. The mobile phone will connect to the System Unit wirelessly.

2.6 Objective Tree

(AW)

The Objective Tree for this project, as seen in Figure 1, shows that the primary goal for the system is to create a pleasant, and comprehensive, user experience. Most of the time spent developing the system will be spent on the creation of the Mobile Application, for the user's Android device, and the software to handle the display, which shows the useful information. The primary goals for the User Experience portion of the design are to provide an easily visible display that shows both useful trip and navigation information. We also want to provide a Mobile Application that is aesthetically pleasing and easy to use. With the physical device, the main objective is to create a device that is easy to install, is compact, and has a long lasting, rechargeable battery.

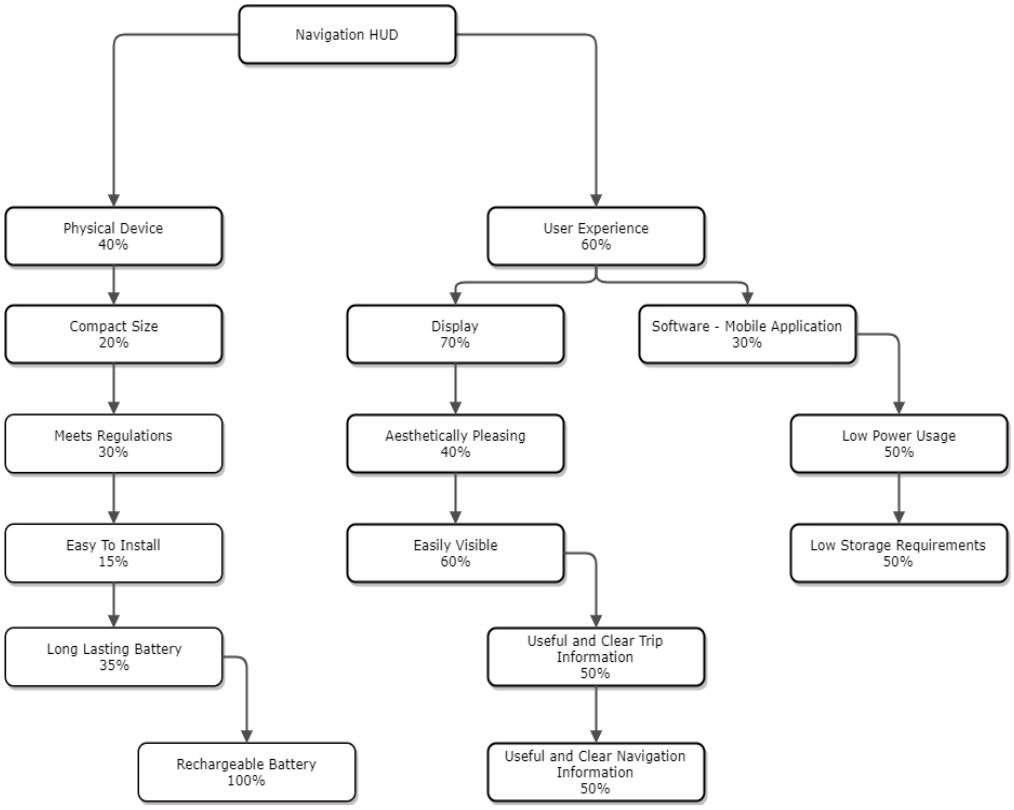


Figure 1: Objective Tree

3. Design Requirement Specifications

(NS/AW/JH/JR)

The design requirement specifications are 11 standards that the Navigation HUD needs to meet for the project to succeed. These requirements are engineering requirements that are helping to guide the project and ultimately achieve the marketing goals. The justification examples the thought process behind the engineering requirement. The full list of design requirements is listed in Table 1.

Marketing Requirements	Engineering Requirements	Justification
2,3	The system will be able to operate for 3 continuous hours on one charge.	The System Unit will need a portable power source to operate. The power supply should last a reasonable amount of time.
1,10	The mobile device and the System Unit will communicate wirelessly, within a range of 10 feet.	For the system to remain unobtrusive, the connection between the mobile device (phone) and the System Unit should be wireless. The connection should also be stable regardless of the position of the phone within the vehicle.
8	The System Unit, located on the dashboard, will measure no more than 4 inches in width, 4.5 inches in length, and 3.25 inches in height.	For the system to be unobtrusive it should be small in size.
6,7	The heads-up display will show information containing the street name, the turn direction, and the distance, in	Having the street name, the turn direction, and the distance until the next navigation

	feet, until the next navigation event onto the windshield.	event is what is needed at minimum to give the driver clear navigational instructions.
6,7	The system will be able to process, and display, data with a delay of no more than 0.5 seconds after receiving the updated GPS position.	For information to be useful, it must be made available to the driver at a rate such that maneuvers can be made safely. Information must be given to the driver well before the maneuver needs to be made.
4,9	The display on the windshield will be visible to a user with 20/20 vision from a distance of 6 feet.	The information displayed to the user should be easy to understand so that the driver is not distracted by the displayed information, and so that only a small amount of time is spent looking at the display.
1	The Mobile Application portion of the system will require no more than 100MB worth of storage on the user's mobile device. The application will also be able to function properly while installed on an external storage device (e.g. an SD card).	100MB is the maximum amount of space that an application can take up to be eligible for download from the "Google Play Store". This makes 100 MB an appropriate upper bound for application size. It is also recommended by Google that applications exceeding 10MB in storage space be installed externally on an

		SD card.
4, 9	The heads-up display on the windshield will measure no larger than 3 inches in height and 6 inches in width.	The display is intended to be unobtrusive for the driver, so an appropriate upper limit has to be set for the display size. 3 inches by 6 inches is large enough to make a clear display and small enough to keep the windshield clear for driving.
6,7	The mobile application will be able to produce GPS instructions that, when being followed by a user, correctly navigate the user from their starting location to their destination.	The intention of the HUD design is to be able to produce a GPS device that functions correctly. It is necessary that a user can test the device and successfully get to their destination by following the instructions on the windshield. This is the best way to prove that the GPS instructions are accurate.
1, 6	The Mobile Application will provide the user with an optional offline mode that preloads instructions so that navigation may be done without the use of location services. The application will automatically switch to offline mode if location services are lost for more than 2 minutes.	In areas with minimal cellular coverage, such as tunnels, it is vital that navigation can still be completed successfully and safely. Providing an offline mode will ensure that the user can still follow the navigation instructions in case of GPS failure.

1	The Mobile Application will cache the user's 10 most recent, and 10 most used, destinations.	To provide the user with a fast, convenient UI, recent and common destinations will be preloaded so that the user can easily access them.

Table 1: Design Requirement Specifications

4. Accepted Technical Design

4.1 Software Theory of Operation

4.1.1 Software Level 0 Block Diagram

(AW)

In a very general sense, the software for the System Unit uses user input for a destination and outside map data to generates the display information. These inputs and the corresponding output can be seen in Figure 2.

Figure 2: Software Level 0 Block Diagram

Module	HUD
Designer	AW / JR
Inputs	User Input (Desired Destination) & Here Maps Data
Outputs	Display Data

Description	A desired location/destination is input by the user. This data, along with information gathered from the Here Maps API, is used to generate directions and trip information. This information is then sent to the display.

Table 2: Software Level 0 Block Diagram

4.1.2 Software Level 1 Block Diagram

(AW)

Looking at Figure 3, the Software Level 1 Block Diagram, we can see that the software design must be divided into two general subparts. The first section, the mobile application, takes in the same data discussed at level 0. Once the information is pulled from the map service, it is then sent to the second section, the display controller. This display controller, otherwise known as the System Unit, must parse the data generated by the mobile application and generate the display that the user will see.

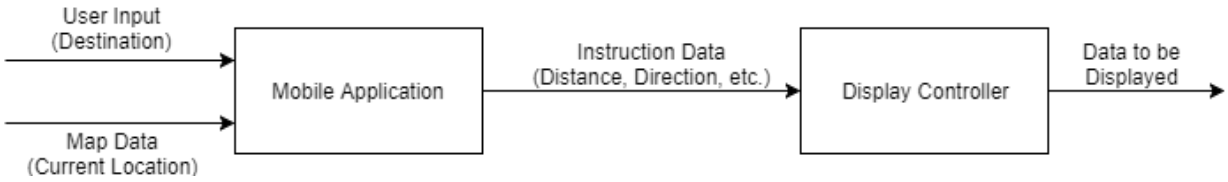


Figure 3: Software Level 1 Block Diagram

Module	Mobile Application
Designer	AW / JR
Inputs	User Input (Desired Destination) and information from the Maps SDK
Outputs	Instruction Data (Distance, next instruction, information to be displayed)
Description	The mobile application takes in a destination from the user, and uses information gathered from the Maps SDK to generate directions, distance to next step, current location, and trip information. This information (Instruction Data) is then sent out to the next stage.

Table 3: Software Level 1 Block Diagram – Mobile Application

Module	Display Controller

Designer	AW / JR
Inputs	Instruction Data (From Mobile Application)
Outputs	Display Data
Description	The input consists of the next direction, the distance to the next maneuver, and simple arrow of next instruction. This data is broken down into a format that can be parsed. This parsed data is then sent to the display.

Table 4: Software Level 1 Block Diagram – Display Controller

4.1.3 Software Level 2 Block Diagram

(AW)

In Figure 4, the Software Level 2 Block Diagram, it can be seen that the mobile application must be split into three parts: the Search UI, the Map UI, and the communication service, which is used to communicate with the display controller. The Search UI is required to allow users to search for, and select, destination addresses. After selecting, or entering, a destination, the user is redirected to the Map UI and Instruction List portion of the application. This portion shows the user's current location, the route on a map, and the full list of instructions along the route. Upon map load and instruction list generation, the current location and the current instruction information is sent to the Communication Service. This service parses the relevant information into JSON format, and then sends the JSON to the display controller via the wireless connection, where it is received in the display controller's Communication Service. The communication services are possibly the most important sections in the design, as a fast, almost instant, transfer of data is required so that information is there when it is needed. After parsing the received JSON, the display controller displays the relevant information to the user.

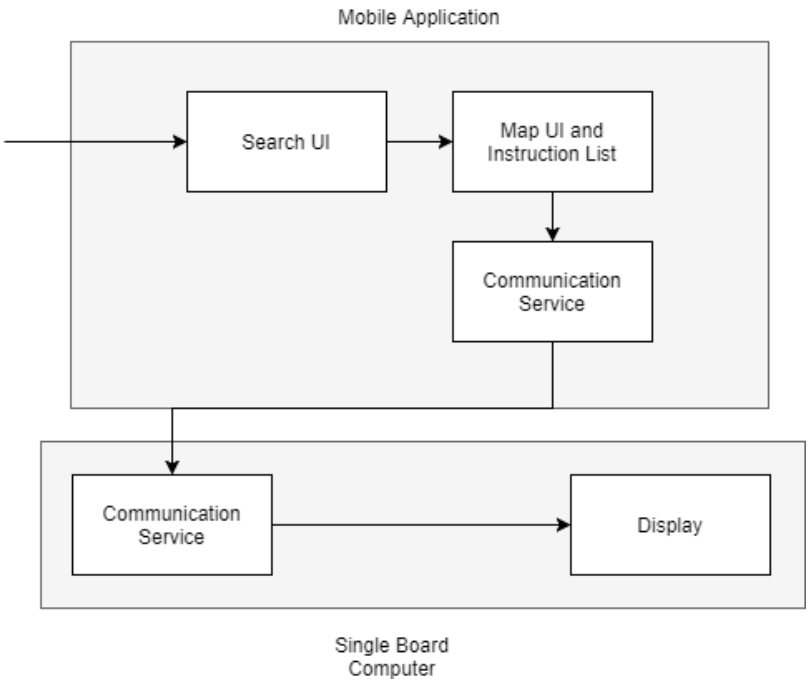


Figure 4: Software Level 2 Block Diagram

Module	Search UI
Designer	AW / JR
Inputs	User input (desired location) and search history from the User History/Location Cache module
Outputs	The desired location, selected by the user, is sent to the Map UI
Description	Allows the user to select/search for a desired location, and passes the selected location on to the Map UI

Table 5: Software Level 2 Block Diagram – Search UI

Module	Map UI and Instruction List
Designer	AW / JR
Inputs	Desired destination that was selected by the user
Outputs	A map with the route highlighted and the instructions for the route
Description	Takes in data from the Search UI, displays the route to the desired location and the user's current location on a map, and displays a list of the instructions

Table 6: Software Level 2 Block Diagram – Map UI and Instruction List

Module	Communication Service (Mobile Device)

Designer	AW / JR
Inputs	Relevant instruction data to be sent to the display
Outputs	The same instruction data that it takes in
Description	Sends data to the microcontroller over the wireless link

Table 7: Software Level 2 Block Diagram – Communication Service (Mobile Device)

Module	Communication Service (System Unit)
Designer	AW / JR
Inputs	Relevant instruction data to be sent to the display
Outputs	The same instruction data that it takes in
Description	Service will constantly be running, looking for information from user device

Table 8: Software Level 2 Block Diagram – Communication Service (System Unit)

Module	Display
Designer	AW / JR
Inputs	Instruction data
Outputs	Display
Description	Displays the instruction data to the user

Table 9: Software Level 2 Block Diagram - Display

4.1.4 Software Level 3 Block Diagram

(AW)

Figure 5, the Software Level 3 Block Diagram shows even more detail into the workings of the software applications. Two of the main differences in the Mobile Application is the addition of a search cache, a JSON parsing service, and an Offline Mode. The search cache is used to store, and display, recent searches and selections to make it easier for the user to find their favorite, or most used, destinations quickly. The JSON parsing service is used in both the Mobile Application and the System Unit, and is used solely for putting the data in a format that is easy to send, receive, and read. These JSON parsing services are responsible for parsing the information brought back from the Map SDK, for compressing the data to be sent to the System

Unit into a form that is easy to understand, and for unpacking that information on the System Unit. Once the information is unpacked by the controller, a custom python UI will display the information from the information in a neat format. The Offline Mode is used when Location Services are lost. If Location Services fail on the mobile device, then Offline Mode will be enabled in order to ensure that navigation can still be carried out successfully. While in Offline Mode, the Mobile Application takes in a user input in the form of a tap on the screen in order to move through the navigation instructions. When Offline Mode is enabled, and the user taps the screen, the next instruction is sent to the JSON parsing service in the Mobile Application, which sends the instruction to the System Unit just as it would any other instruction. This mode allows users to continue using the system without having to divert their gaze from the road.

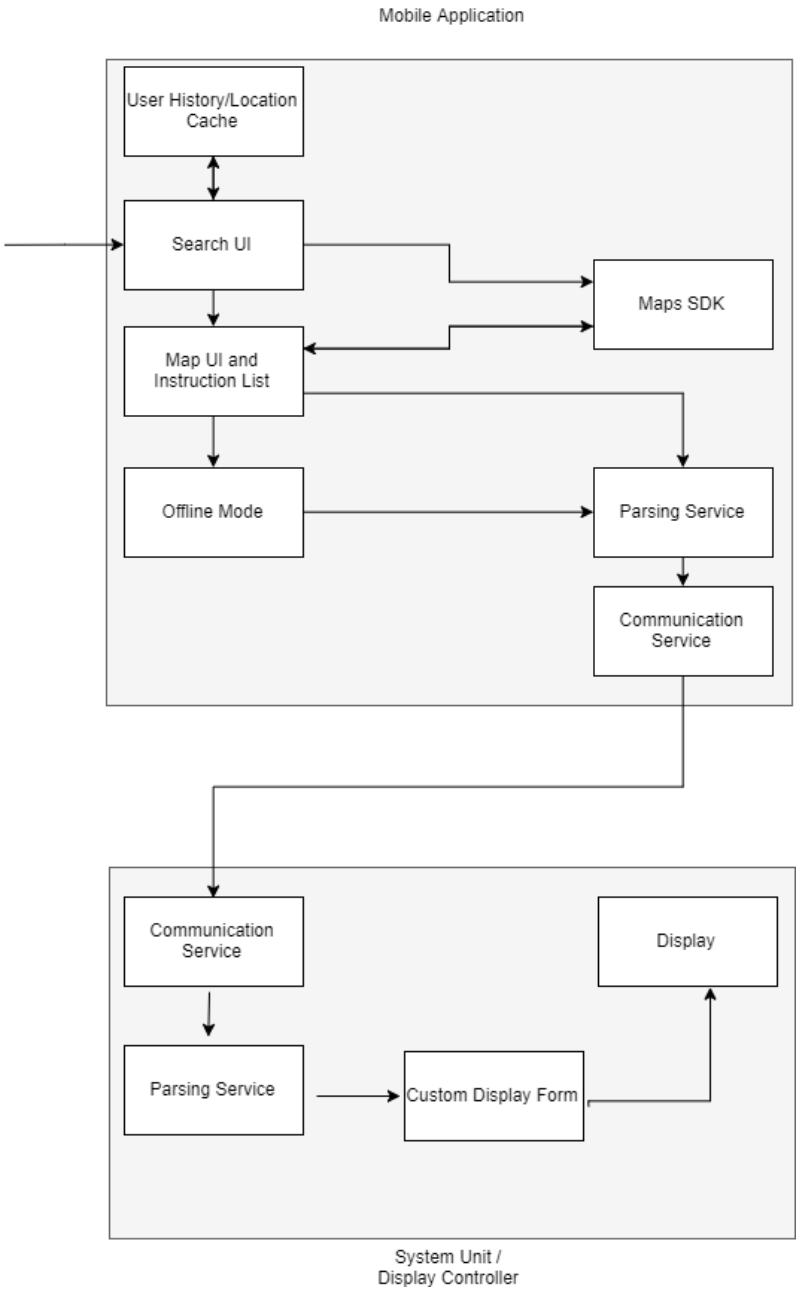


Figure 5: Software Level 3 Block Diagram

Module	User History/Location Cache
Designer	AW / NS
Inputs	Searches conducted in the Search UI
Outputs	Locations that have previously been selected/searched for by the user(s)
Description	Cache to store user's search history and selected locations. Used to provide a smoother, and more user friendly, UI.

Table 10: Software Level 3 Block Diagram – User History/Location Cache

Module	Search UI
Designer	AW / NS
Inputs	User input (desired location searches) and search history from the User History/Location Cache module
Outputs	The desired location, selected by the user, is sent to the Map UI and the Maps SDK
Description	Allows the user to select/search for a desired location, and passes the selected location on to the Map UI, where it is displayed, and the Maps SDK, where the route is calculated

Table 11: Software Level 3 Block Diagram – Search UI

Module	Map UI and Instruction List
Designer	AW / NS
Inputs	Desired destination from the Search UI, and the Instruction List from the Maps SDK. Also takes in location information from the Maps SDK
Outputs	A map with the route highlighted and the instructions for the route, which is displayed to the user. Also outputs the instruction list to both the Offline Mode and the Parsing Service
Description	Takes in data from the Search UI and Maps SDK, displays the route to the desired location and the user's current location on a map, and displays a list of the instructions within the route

Table 12: Software Level 3 Block Diagram – Map UI and Instruction List

Module	Offline Mode
Designer	AW
Inputs	Instruction List from the Maps UI and a User Input
Outputs	Next Instruction from the Instruction List
Description	Contains the entire instruction list, and sends the next instruction to the mobile device's Parsing Service each time the user taps the mobile device's screen

Table 13: Software Level 3 Block Diagram – Offline Mode

Module	Parsing Service (Mobile Device)
Designer	AW / NS
Inputs	The current instruction from either the Offline Mode or the Map UI/Instruction List

Outputs	List of instructions and route information in JSON format
Description	Takes in the next instruction from the instruction list in either the Map UI or the Offline Mode. Parses the instruction in to JSON format and sends the relevant information to the Communication Service. This module is responsible for gathering, parsing, and generating instruction data that will be displayed to the user.

Table 14: Software Level 3 Block Diagram – Parsing Service (Mobile Device)

Module	Maps SDK
Designer	Here Maps - Provider of the SDK
Inputs	Any and all calls required
Outputs	Responses to the calls made by the application
Description	Generates instruction lists and supplies location information

Table 15: Software Level 3 Block Diagram – Maps SDK

Module	Communication Service (Mobile Device)
Designer	AW / NS
Inputs	Relevant instruction data to be sent to the display in JSON format
Outputs	The same instruction data that it takes in bytes
Description	Sends data to the System Unit over the wireless connection.

Table 16: Software Level 3 Block Diagram – Communication Service (Mobile Device)

Module	Communication Service (System Unit)
Designer	JH / JR
Inputs	Instruction data in bytes
Outputs	Display data in JSON format
Description	Takes in data from the Mobile Application over the wireless connection and sends that data to the System Unit’s Parsing Service

Table 17: Software Level 3 Block Diagram - Communication Service (System Unit)

Module	Parsing Service (System Unit)
Designer	JH/JR
Inputs	Instruction data in JSON format
Outputs	An object containing the relevant instruction information

Description	Takes the data sent over from the Mobile Application (in JSON format), and generates an object containing all of the relevant instruction information to be displayed

Table 18: Software Level 3 Block Diagram – Parsing Service (System Unit)

Module	Custom Display Form
Designer	JH/JR
Inputs	Object containing instruction information
Outputs	Display
Description	UI Application running on the System Unit that takes the object from the System Unit's Parsing Service and displays the information

Table 19: Software Level 3 Block Diagram – Custom Display Form

Module	Display
Designer	JH / JR
Inputs	Display Data
Outputs	Display
Description	Displays the instruction data to the user

Table 20: Software Level 3 Block Diagram - Display

4.1.5 Mobile Application Software Flow Diagram

(AW)

This section describes the general logic of the software behind the Mobile Application. The software flow diagram listed at the bottom of this description follows these steps after the app is initially loaded:

1. After the application is initially launched, two processes are started:
    a. The application checks for a default System Unit to connect to. This default device is set in the User Settings page within the application. Depending on whether the default device is set, or within range, one of the following two steps is taken:

i. If the default device is set, and the default device is in range, then a connection attempt is made.

ii. In initial design if the default device was not set, or the default device was not in range, then the application would search for nearby Wi-Fi direct enabled devices. In implementation, the application searches for nearby Bluetooth devices and completes the following step:

1. The application displays a list of discovered devices that match the signature of the System Unit and awaits a user selection. Once the user selects the System Unit, the application waits for the user to input the destination address (see Step 2).

b. The application loads the Map UI and centers the map around the user's current location. The map is cleared of all markers, and all lists are initialized. After the Map UI is initialized, and the map is cleared, the application awaits a user input for the destination address (see Step 2).

2. After both Step 1a and Step 1b are completed, the user can input a destination address. The user may input a new destination, or they may select from a list of recent and common destinations. After the input is entered and finalized, the application goes to Step 3.

3. After the input is finalized, the address is searched for using the Maps SDK. Depending on if the entered destination is found (is a legitimate address), one of the two following steps is taken:

a. If the destination address is found, then the route is calculated using the Maps SDK. The route is then stored in a list, which is then displayed to the user on the mobile device. The application then goes to Step 4.
b. If the entered destination is not a legitimate address, or the address cannot be found, then the application returns to Step 1b.

4. The top instruction (i.e. the next instruction) in the queue is transmitted to the connected System Unit. After a successful transmission, the application goes to Step 5.
5. The application must check to ensure that location services are available, and that the GPS position is available. Depending on the status of the location services, one of the following steps takes place:
    a. If the location services are available, the application awaits an update on the GPS position of the user's mobile device. After the position is updated, the application goes to Step 6.
    b. If the location services are not available, then the application enters Offline Mode. While in Offline Mode, the UI switches to a state that only takes in a single user input, which is a simple tap on the screen. The following two steps may occur while in Offline Mode:
        i. If the user taps the screen, then it is assumed that they have completed the current instruction, and the application goes to Step 6.
        ii. If the Location Services are restored while in Offline Mode, then the application goes back to Step 3a.

6. After the GPS position is updated, the application checks the next instruction in the queue. Depending on what the next instruction step is, one of the two following steps may occur:
    a. If the next instruction does not exist, meaning that the user has reached their destination, then the application notifies the user and returns to Step 1b. This signifies the end of the navigation, and the application is prepared for a new destination to be entered by the user.
    b. If the next instruction does not indicate that the destination has been reached, then the next instruction is pulled to the front of the queue, and the application returns to Step 4.

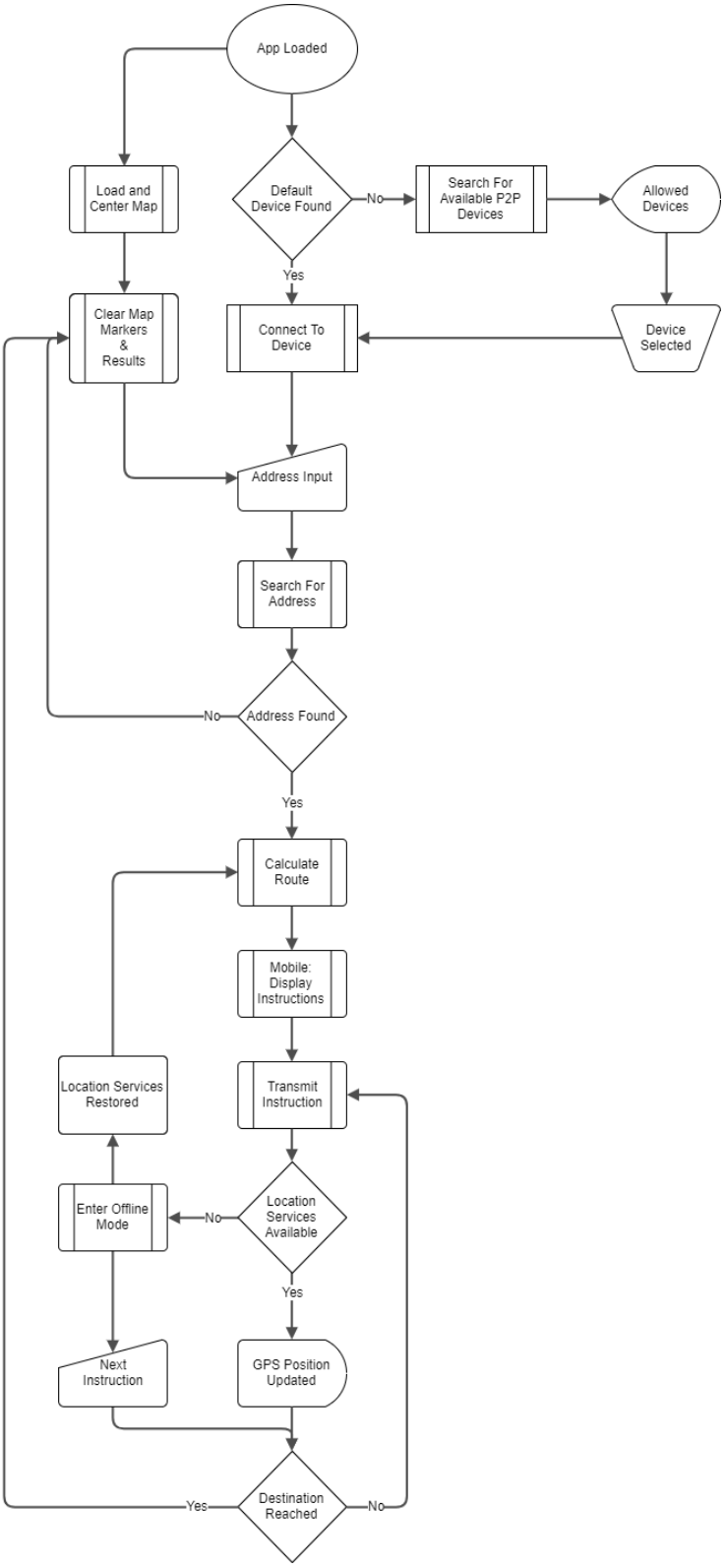


Figure 6: Mobile Application Software Flow Diagram

4.1.6 Mobile Application Class Definitions

(AW)

There are several classes that must be defined to gather, and handle all of the data within the mobile application. The first class that must be defined is the AddressResult class, which stores and provides the addresses selected by the user. Descriptions and explanations for this class' members, methods, and functions can be seen in Table 21.

class AddressResult	
private String	**DisplayText**
	Private member variable; Text to be displayed to the user. Represents the Address, or Description, of the location.
private GeoCoordinate	**coordinates**
	Private member variable; Coordinates of the address.
public AddressResult	**AddressResult(String Display)**
	Constructor that sets the display text.
public AddressResult	**AddressResult(String Display, GeoCoordinate geoCoordinate)**
	Constructor that sets the coordinates and the display text.
public void	**setDisplayText(String Display)**
	Setter for private member variable 'DisplayText.'
public String	**getDisplayText()**
	Getter for private member variable 'DisplayText.'
public void	**setCoordinates(GeoCoordinate geoCoordinate)**
	Setter for private member variable 'coordinates.'
public double	**getLatitude()**
	Retrieves the latitude of the address.
public double	**getLongitude()**
	Retrieves the longitude of the address.

Table 21: Mobile Application Class Definition - AddressResult

After defining the AddressResult class, an AddressResultListViewAdapter must be defined so that a collection of AddressResult objects can be properly bound to a ListView within

the mobile application. This allows for search results to dynamically populate a ListView that can then be shown to the user. Descriptions for this class' members, methods, and functions can be seen in Table 22.

class AddressResultListViewAdapter	
private AddressResultListViewAdapter	**AddressResultListViewAdapter(Context context, int textViewResourceId)**
	Public constructor.
private AddressResultListViewAdapter	**AddressResultListViewAdapter(Context context, int resource, List<AddressResult> items)**
	Public constructor. Takes in the list of AddressResult objects to be displayed to the user.
public View	**getView(int position, View convertView, ViewGroup parent)**
	Sets the display text of the ListView Item for each AddressResult object that is to be displayed.

Table 22: Mobile Application Class Definition - AddressResultListViewAdapter

Like the AddressResultListViewAdapter, a RouteStepListViewAdapter must be defined for the display of steps within a given route. This allows the steps for a generated route to be displayed to the user in a concise, easy to read, list. Descriptions for this class' members, methods, and functions can be seen in Table 23.

class RouteStepListViewAdapter	
private RouteStepListViewAdapter	**RouteStepListViewAdapter(Context context, int textViewResourceId)**
	Public constructor.
private RouteStepListViewAdapter	**RouteStepListViewAdapter(Context context, int resource, List<Maneuver> maneuvers)**
	Public constructor. Takes in a list of Maneuver objects that are to be displayed to the user.
public View	**getView(int position, View convertView, ViewGroup parent)**
	Sets the display text of the ListView Item for each Maneuver object that is to be displayed.

Table 23: Mobile Application Class Definition - RouteStepListViewAdapter

After being given a destination and generating a route to that destination, the mobile application must then send the current location and navigation information to the System Unit (as discussed in Section 4.1.2 Software Level 1 Block Diagram).

In the initial accepted design, this task was to be handled by a series of classes, interfaces, and activities, all of which are described below. To start, a WiFiDirectBroadcastReceiver class would be defined. This class would extend the BroadcastReceiver class provided in the standard Android SDK, which would allow the application to monitor changes in the device's Wi-Fi adapter.

class WiFiDirectBroadcastReceiver	**extends BroadcastReceiver**
public WiFiDirectBroadcastReceiver	**WiFiDirectBroadcastReceiver(WiFiP2pManager manager, WiFiP2pManager.Channel channel, Handler peerHandler, Context context)**
	Public Constructor that instantiates the Context, WiFiP2pManager, Channel, and Handler. Calls the setListener() method.
private void	**setListener()**
	Overrides the ConnectionInfoListener, which allows for the monitoring of the Wi-Fi Direct connection.
public void	**onReceive(Context context, Intent intent)**
	Overrides the onReceive method of BroadcastReceiver. This method handles events related to the P2P connection, such as: Peers Changed, Connection Changed, and Device Wi-Fi State Changed. Responsible for adding peers to the UI, and responsible for attempting to regain connections when the connection fails.
protected void	**showAlert(String message)**
	Shows a toast notification to the user, with the text set to the passed in message.

Table 24: Mobile Application Class Definition – WiFiDirectBroadcastReceiver

After defining the WiFiDirectBroadcastReceiver class, the P2P (or "Peer-to-Peer") class would be defined. This class utilizes the WiFiDirectBroadcastReceiver object to find peers, display peers, and make connections to the peer that is selected by the user.

class P2P	
private void	**createIntent()**
	Sets up the intent filter. This defines which intents the WiFiDirectBroadcastReceiver will override.
public P2P	**P2P(Context context, Handler peerHandler)**
	Public Constructor.
private void	**initializeWiFiDirect()**
	Initializes the Wi-Fi Direct manager and channel. This also sets up a loop that will attempt to reconnect after a Wi-Fi Direct connection fails.
private void	**createReceiver()**
	Instantiates the private WiFiDirectBroadcastReceiver with the manager and channel generated in the initializeWiFiDirect method.
private void	**registerReceivers()**
	Registers receivers within the correct context.
private void	**unregisterReceivers()**
	Unregisters the receives when the application is closed.
private void	**discoverPeers()**
	Initializes the discovery process.
public void	**connect(WiFiP2pDevice device)**
	Establishes a Wi-Fi Direct connection with the specified device.

Table 25: Mobile Application Class Definition – P2P

(NS)

In the actual design and implementation of the heads-up display, a Bluetooth connection was used to send data from the mobile phone over to the system unit. For this communication protocol there were some differences in the mobile application class definitions.

class BTService	
public BTService	**BTService(Context context)**
	Public Constructor.
Public boolean	**isDiscovering()**
	Returns true if the adapter is currently discovering. Otherwise, returns false.
public void	**discover(Boolean autoConnect)**

	Begins discovery of Bluetooth devices. If autoConnect is true, an automatic connection will be made with the System Unit based on the System Unit's Bluetooth device name.
public void	**connect(BluetoothDevice target)**
	First pairs with the target device, then establishes the connection based on the UUID that both the mobile application and the System Unit share.
public boolean	**isConnected()**
	Returns true if the mobile device currently has an established Bluetooth connection, otherwise returns false.
public void	**transmit(String message)**
	Converts a string to a byte array, then sends it over the established Bluetooth connection.

Table 26: Mobile Application Class Definition - BTService

4.1.7 Mobile Application Pseudocode

(AW)

The Mobile Application is responsible for displaying the map, connecting to the System Unit, and generating the route based on the user's input. The first step in this process is connecting to the System Unit. In the initial design of the heads-up display, the connection between the system unit and the mobile device would be made using the following procedure:

In the planned design, the Mobile Application initiates the connection process by first using the mobile device's Wi-Fi Broadcast Receiver to search for Wi-Fi Direct enabled devices. The search is conducted so long as no connection is made. Once a connection is made, the application terminates the search, as it is no longer necessary. During the searching process, once a device is found it is displayed to the user in a user-friendly list. After the user selects a device from this list, the Mobile Application uses the mobile device's Wi-Fi Manager to establish the connection. If the connection succeeds, then the application proceeds to the next stages - taking in user input and generating the route. If the connection fails, then the application attempts the connection again. This process is explained in further detail in Section 4.1.5 Mobile

Application Software Flow Diagram, and the pseudocode for this process can be found in Figure 7.

```
Search For Devices Using Wi-Fi
Broadcast Receiver:
While (Connection_Status != Connected) {
        Search_For_P2P_Devices();
        If (Device_Found) {
                Display_Device_Name();
        }
}

Device Selected From List of Available Devices:
new WifiP2pManager.ActionListener() {
      public void onFailure(int reason) {
                Log_Failure(reason);
                Retry_Connection();
        }
};
```

Figure 7: Device Search Pseudocode

In implementation, a similar procedure was performed using Bluetooth instead of Wi-Fi Direct, using the functionality and class described in Table 26: Mobile Application Class Definition - BTService.

After a connection is made and the user inputs a destination address, the Mobile Application moves to the Route Generation process. This process starts with a search for the entered address. Using the Maps SDK, the application generates a list of all addresses that match the user's input and displays these results in a user-friendly list. For each result, the application stores the latitude and longitude coordinates of the result and places a marker on the map at those coordinates. After displaying the results, the application awaits the user's selection. Once a selection is made, the application gets the coordinates of the selected result, sets the route options (e.g. does the user prefer a route without tolls?), and adds waypoints at the user's current position and the result's coordinates. Once these initial steps are completed, the application calls upon

the Maps SDK to generate the route. If a route is found, then the application stores the instruction list, displays the route on the map, and displays the instruction list. This process is explained in further detail in Section 4.1.5 Mobile Application Software Flow Diagram, and the pseudocode for this process can be found in Figure 8.

```
User Searches For an Address or Place:
Search_Address(User_Entry);
Center_Map();
For (Address a : Search_Results) {
        Get_Coordinates(a);
        Place_Marker_At_Coordinates();
        Add_Address_To_Results_List();
}

User Selects a Marker or a List Entry:
Get_Selected_Coordinates();
Set_Route_Options();
Add_Waypoints();
Calculate_Route();
If (Route != null) {
        Display_Route_On_Map():
        Show_Route_List();
        Center_Map();
}
```

Figure 8: Address Search Pseudocode

After a connection is made and an instruction list is generated, the next step for the Mobile Application is to parse the current instruction into JSON format and transmit that instruction to the System Unit. The first step in this process is to create the JSON object. The JSON object is created by adding several key-value pairs to a string-like structure. The pairs that will be added to this object include the Street Label/Name, the Distance to Next Instruction in units of feet, and the Direction of the next instruction (i.e. left, right, straight, etc.). After the JSON object is created it is then converted to a byte array, which can be placed in the data buffer and transmitted to the System Unit over the wireless connection. This process is explained in

further detail in Section 4.1.5 Mobile Application Software Flow Diagram, and the pseudocode for this process can be found in Figure 9.

```
Create the JSON Object:
While (Confirmation_of_Pi_Service == True) {
    JSON_Object Direction_Data;
    Direction_Data.add(Street_Label, Street_Data);
    Direction_Data.add(Distance_To_Next_Direction, Distance_Data);
    Direction_Data.add(Direction_Arrow, Arrow_Data);
    String JSON_String = Covert_To_String(JSON_Object);
}

Convert and Send the Data to the Pi
While (End_Of_Data_Stream == False) {
    Data_To_Send = Convert_To_Byte_Array(JSON_String);
    Andoid_Data_Buffer.Add(Data_To_Send);
    Send_Data_Over_Bluetooth(Android_Data_Buffer);
}
```

Figure 9: Data Transfer Pseudocode

An extra piece of the Mobile Application software that must be considered is the Offline Mode. This mode allows navigation to take place even when the mobile device loses access to location services, GPS position, or cellular service. If the mobile device does lose access to one of these services, it is automatically placed in to Offline Mode, which loads a new UI that consists of only one pane that takes in only one user input - which is a simple tap on the device's screen. When the user taps on the screen it is assumed that they have completed the previous instruction and are ready to view the next instruction. This input triggers the same mechanisms detailed in the above sections - the next instruction is pulled from the instruction list, compressed into a format that can be sent, and is then transmitted to the System Unit. The only difference between Online and Offline Mode is that the application does not update the current instruction upon GPS position update, but rather it updates the next instruction upon a user input. While in Offline Mode, the application continuously listens for the restoration of location services. If location services are restored, then the application returns to Online Mode by finding the user's

current position, generating a new route, and resuming normal functionality. This process is explained in further detail in Section 4.1.5 Mobile Application Software Flow Diagram, and the pseudocode for this process can be found in Figure 10.

```
Location Service Listener:
If (!PositionAvailable) {
    Set_Offline_UI();
    Return_To_Listening();
} Else If (PositionAvailable) {
    If (In_Offline_Mode) {
        Set_Online_UI();
        Regenate_Route();
        Return_To_Online_Functionality();
    }
}

Offline Mode Activity:
void User_Input_Detected() {
    Get_Next_Instruction();
    Generate_JSON();
    Send_Instruction();
    Update_Instruction_List();
    Await_User_Input();
}
```

Figure 10: Offline Mode Pseudocode

4.1.8 System Unit Class Definitions

(JR/NS)

The class definitions listed below represent the planned design for supplying information about the necessary functions and data types required to properly implement the software running on the System Unit. The SBCP2P ("Single Board Computer Peer-to-Peer") class is used in this design to allow the System Unit's single board computer to broadcast and establish a Wi-fi Direct connection.

In implementation, very similar methods were used to those in Table 27, however they utilized Bluetooth functionality instead of Wi-Fi Direct. Rather than broadcasting an IP address

and Port information, the System Unit broadcasts its Bluetooth adapter and socket. The Mobile Device then detects the System Unit through a discovery process, both devices open a Bluetooth socket, and the connection is then finalized. Data is then transmitted via a byte array within the devices' buffer.

class SBCP2P	
public void	**broadcastIPAndPort()**
	Broadcast IP and Port information
public void	**discoverPeers()**
	Initializes the discovery process.
public void	**receiveBytes()**
	Sets the System Unit up to receive data over the Wi-Fi Direct connection. Data is received as a byte array within a buffer on the device.

Table 27: System Unit Class Definition – SBCP2P

The display class, shown in Table 28, holds the necessary data for producing a display with the information required by the design. It also contains the capacity to take in a JSON object and parse out the correct data to be displayed.

class Display	
public Display	**Display(JSON_Object json)**
	Public constructor that produces a display object based off of a JSON Object.
private Image	**turnImage**
	Private member that holds an image, which corresponds to the current turn direction.
private string	**streetName**
	Private member that holds the data for the street name so that it may be displayed.
private string	**turnDirection**
	Private member that holds the data for the direction of the next turn so that it may be displayed.
private int	**Distance**
	Private member that holds the data for the distance until the next navigational instruction so that it may be displayed.

Table 28: System Unit Class Definition - Display

4.1.9 System Unit Pseudocode

(NS)

The System Unit is responsible for receiving the navigational data from the mobile application, and then using that data to produce the heads-up display. To begin this process, first the System Unit must establish a connection with the mobile device. The pseudocode sample in Figure 11 shows the planned design of this process.

```
Service_Start(){
        SBCP2P p2p = new SBCP2P;

        Broadcast the Wi-Fi Direct Service:
        While (Connection == False) {
                p2p.broadcastIPAndPort();
        }

        Returning Service Looking for the Mobile Device IP and Port:
        While (Connection == True && Confirmation==False) {
                p2p.sendConfirmation();
        }

        Receiving Service Taking in Bits Over Connection:
        While (Connection == True && Confirmation == True ) {
                p2p.receiveBytes();
        }
}
```

Figure 11: System Unit Data Retrieval Pseudocode

This design begins by making the System Unit broadcast its IP address and port number. When this happens, the System Unit is making itself visible to other devices with Wi-Fi Direct support. This means that the mobile device can discover the System Unit and start to establish a connection.

Once the mobile device begins its attempt to connect to the System Unit, the System Unit needs to send some sort of confirmation back over Wi-Fi Direct to inform the mobile device that it agrees to receive data through this newly established network. Through this new network, both

the mobile device and System Unit become paired with one another through Wi-Fi Direct and the two can now freely send data back and forth. The pseudocode sample in Figure 12 details the process of receiving data on the System Unit and using that data to produce a display.

In implementation, a similar procedure was performed using Bluetooth instead of Wi-Fi Direct.

Receiving and Parsing the JSON Data:

```
While (End_Of_Data_Stream == False) {
        ByteArray Received_Data = Receive_Data_Over_WiFiP2p(Pi_Data_Buffer);
        String JSON_String = Convert_To_String(Received_Data);
        JSON_Object Direction_Data_Object = Convery_To_JSON_Object(JSON_String);
        Display display = new Display(Direction_Data_Object);
        Project_Output(Display);
}
```

Implementation of Display Constructor that Transforms the Parsed Data Into a Display:

```
public  Display(JSON_Object json){
        this.turnDirection = JObject.Direction_Arrow;
        this.streetName = JObject.Street_Label;
        this.distance = JObject.Distance_To_Next_Direction;
}
```

Figure 12: System Unit Display Generation Pseudocode

When data is received by the System Unit, it comes through as a byte array in a buffer. This data needs to be converted to information that the System Unit can use to produce a display. In this case, a unique class was created that has a constructor that can take in a JSON object and use it to create a new unique object specifically for producing the display. This process involves first taking the data in through the buffer and receiving it as an array of bytes. That byte array first needs to be converted to a string. Since the data was sent over as a string in the JSON format, when converted it will be received with that same formatting. As a result, this string can easily be parsed by the System Unit by constructing a JSON object.

As a JSON object, the data is now organized as name-value pairs. This pairing allows the System Unit to select each piece of received data individually, and correctly parse out the street

name, distance to next instruction, and turn direction with ease. This process is performed by the constructor for the display object and produces cohesive data for outputting the display. The output will use Tkinter, which is a GUI toolkit that is included in Python. The idea is to have a label associated with each JSON element. Figure 13 shows a potential design for the projected display, and Figure 14 shows the final display implementation.

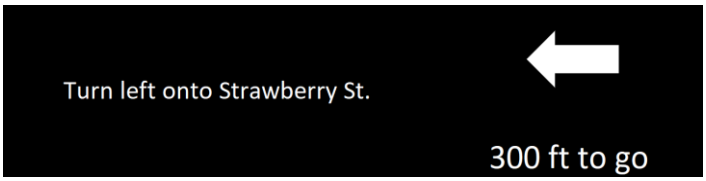


Figure 13: Initial Display Design

Figure 14: The Implemented Display

4.2 Hardware Theory of Operation

4.2.1 Hardware Level 0 Block Diagram

(NS)

The fundamental concept of the heads-up display hardware is that the user of the heads-up display submits some sort of input with a directional request, then GPS information is displayed on the user's car windshield Hardware Block Diagram 0, shown in Figure 15, shows this basic element of the design.

Figure 15: Hardware Level 0 Block Diagram

Module	HUD
Designer	NS / JH
Inputs	User Input
Outputs	Displayed GPS Visuals
Description	The function of the HUD is to take the GPS request made by the user and transform it into clear displayed information right in front of them. The user submits a destination and the HUD shows directions in a display on their windshield.

Table 29: Hardware Level 0 Block Diagram - HUD

4.2.2 Hardware Level 1 Block Diagram

(NS)

More specifically, the HUD consists of two parts: the mobile device and the system unit. The mobile device will be a user's internet-connected smartphone which will receive the GPS data. This data will then be sent over to the system unit for parsing. Figure 16, the Hardware Level 1 Block Diagram, shows these details.

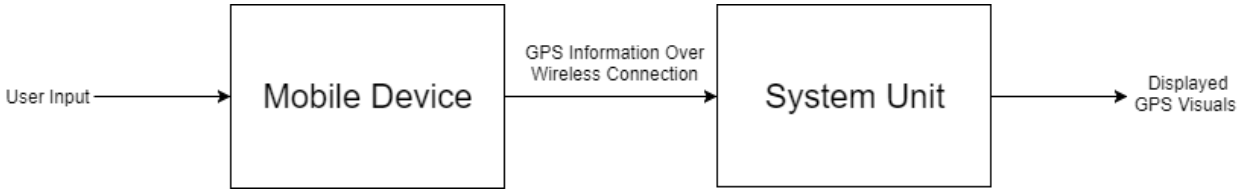


Figure 16: Hardware Level 1 Block Diagram

Module	Mobile Device
Designer	NS / JH

Inputs	User Input
Outputs	GPS Information Over Wireless Connection
Description	The mobile device simply sends the GPS data to the rest of the HUD. It does this over a wireless connection that it makes with the rest of the System Unit.

Table 30: Hardware Level 1 Block Diagram - Mobile Device

Module	System Unit
Designer	NS / JH
Inputs	GPS Information Over Wireless Connection
Outputs	Displayed GPS Visuals
Description	The System Unit is responsible for creating the displayed visuals by interpreting the information that it receives from the mobile device. The System Unit retains a wireless communication with the phone while the two are working in tandem. The System Unit processes and parses GPS information sent over by the mobile device, then it creates clear visuals for the HUD user.

Table 31: Hardware Level 1 Block Diagram – System Unit

4.2.3 Hardware Level 2 Block Diagram

(JR)

The transfer of information between the mobile device and the System Unit is a key component in the operation of the system. We assessed both Bluetooth and Wi-Fi Direct communication protocols for the sending and receiving of our data. After researching the two communication protocols we decided to choose Wi-fi Direct due to a higher capacity for data and broadcast speed while also maintaining a high level of security [10]. Wi-Fi Direct promises AES 256-bit encryption while Bluetooth gives only AES 128-bit encryption [10]. While this may seem like a trivial difference, privacy and security of data transfer is crucial in any consumer-facing application. Responsibly handling location data and making sure that it remains in the right hands is extremely important to ensure user safety and a secure system. Ultimately, Wi-Fi Direct was decided as the data transmission tool for the system. Figure 17, the Hardware Level 2 Block Diagram, shows a more detailed design where the process of using this wireless

connection is exemplified.

In implementation however, a Wi-Fi Direct connection that could send the required data over was unable to be established. Because of this, Bluetooth was chosen as the communication protocol for the final design.

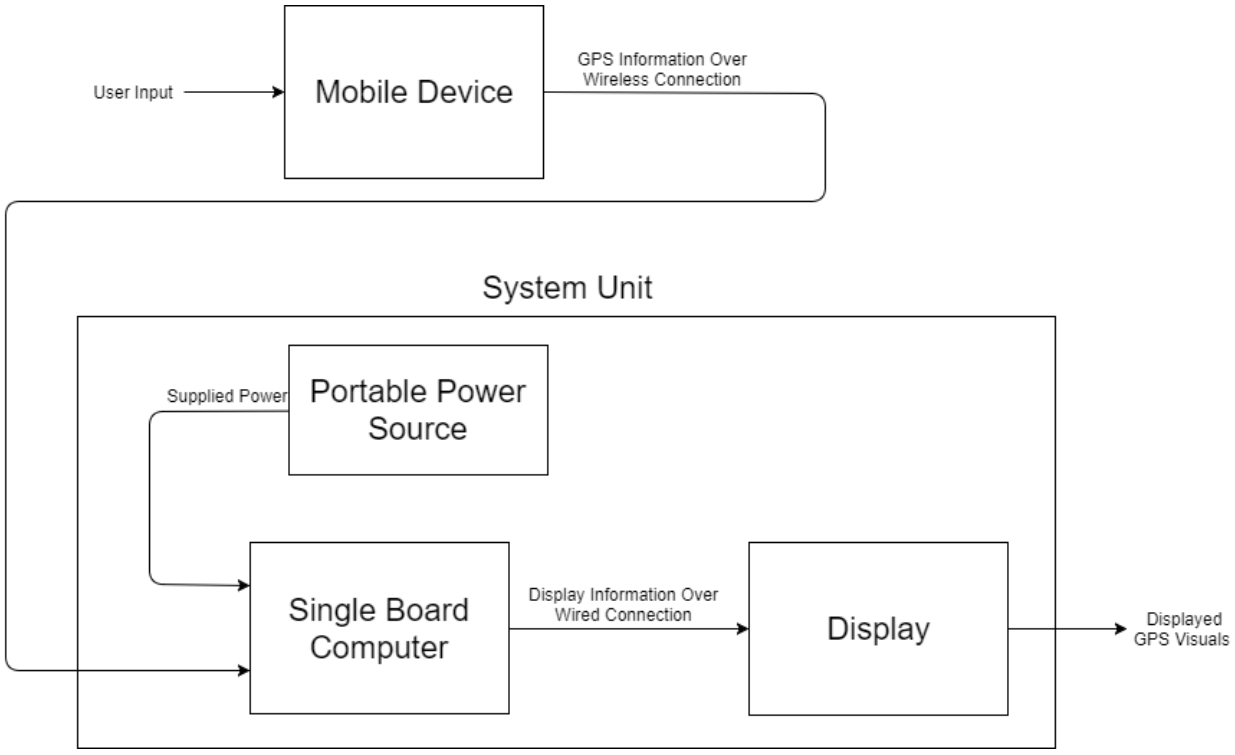


Figure 17: Hardware Level 2 Block Diagram

Module	Mobile Device
Designer	NS / JH
Inputs	User Input
Outputs	GPS Information Over Wireless Connection
Description	The mobile device simply sends the GPS data to the rest of the HUD. It does this over a wireless connection that it makes with the rest of the System Unit.

Table 32: Hardware Level 2 Block Diagram – Mobile Device

Module	Single Board Computer
Designer	NS / JH
Inputs	GPS Information Over Wireless Connection, Supplied Power

Outputs	Display Information Over Wired Connection
Description	The Single Board Computer takes all of the data from the mobile device and then utilizes a power source in order to supply all of the inputs to the display. It takes in data wirelessly from the mobile device and utilizes the portable power source, then uses a wired connection to the display to send over the data that will be shown.

Table 33: Hardware Level 2 Block Diagram – Single Board Computer

Module	Portable Power Source
Designer	NS / JH
Inputs	None
Outputs	Supplied Power
Description	The portable power source is the single element that supplies power to the System Unit.

Table 34: Hardware Level 2 Block Diagram – Portable Power Source

Module	Display
Designer	NS / JH
Inputs	Display Information Over Wired Connection
Outputs	Displayed GPS Visuals
Description	The display is the part of the design that the end user sees. It displays the GPS data that it receives from the single board computer.

Table 35: Hardware Level 2 Block Diagram - Display

4.2.4 Hardware Level 3 Block Diagram

(JR)

After researching different microcontrollers, the Raspberry Pi 3 is the best choice for the controller within the System Unit. The main reason for using the Pi 3 was the dual option to use either Bluetooth or Wi-Fi Direct. Even though the proposed design plans to utilize Wi-Fi Direct, Bluetooth can be a backup option without changing hardware. Another great feature of the Pi 3 is the onboard HDMI port. This gave the device flexibility when completing screen testing. Figure 18 shows more detail of this hardware design.

In our final design we used a Raspberry Pi 0 rather than the Raspberry Pi 3. The Pi 0 had all the same wireless adaptor options with a much lower profile. The low profile was necessary to complete design requirement 6 (see Section 3. Design Requirement Specifications).

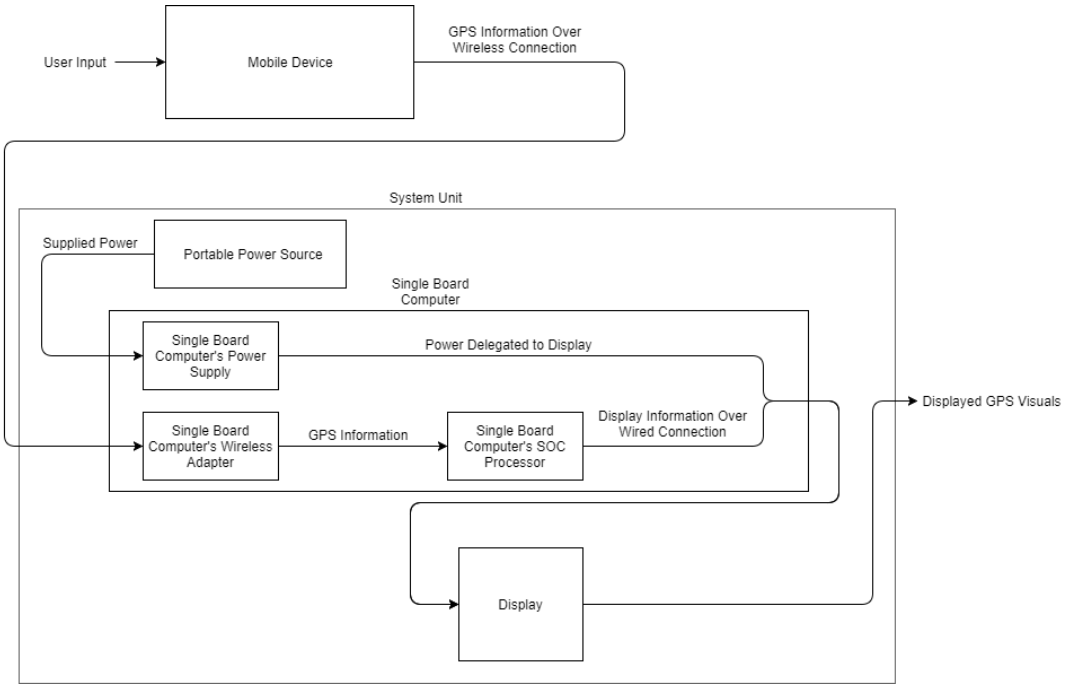


Figure 18: Hardware Level 3 Block Diagram

Module	Mobile Device
Designer	NS / JH
Inputs	User Input
Outputs	GPS Information Over Wireless Connection
Description	The mobile device simply sends the GPS data to the rest of the HUD. It does this over a wireless connection that it makes with the rest of the System Unit.

Table 36: Hardware Level 3 Block Diagram – Mobile Device

Module	Portable Power Source
Designer	NS / JH
Inputs	None
Outputs	Supplied Power
Description	The portable power source is the single element that supplies all of the power to the System Unit.

Table 37: Hardware Level 3 Block Diagram – Portable Power Source

Module	Single Board Computer's Wireless Adapter
Designer	NS / JH
Inputs	Wi-Fi Direct Signal
Outputs	Received Wi-Fi Direct Data
Description	The wireless adapter in the single board computer is responsible for receiving all the GPS data from the mobile device. It then sends the data to the processor so the appropriate GPS data can be sent to the screen display.

Table 38: Hardware Level 3 Block Diagram – Single Board Computer's Wireless Adapter

Module	Single Board Computer's Power Supply
Designer	NS / JH
Inputs	Supplied Power
Outputs	Power Delegated to Display
Description	The power supply is the part of the single board computer that ensures that each individual element of the System Unit gets powered, including the display.

Table 39: Hardware Level 3 Block Diagram – Single Board Computer's Power Supply

Module	Single Board Computer's SOC Processor
Designer	NS / JH
Inputs	Received Data
Outputs	Data Sent to Screen Display
Description	The processor is responsible for managing the received GPS data. It uses its programmed logic to transform the received wireless data into information that creates the appropriate visual display.

Table 40: Hardware Level 3 Block Diagram – Single Board Computer's SOC Processor

Module	Display
Designer	NS / JH
Inputs	Display Information Over Wired Connection, Power Delegated to Display
Outputs	Displayed GPS Visuals
Description	The display is the part of the design that the end user sees. It displays the GPS data that it receives from the single board computer's SOC processor.

Table 41: Hardware Level 4 Block Diagram - Display

4.2.5 Hardware Level 4 Block Diagram

(JH/JR)

Going into more detail, user input will be submitted to the mobile device through the touch screen, and the running application on the phone allows the user to interface with a wireless adapter on the mobile device. Depending on finalized design decisions, this will either be the Bluetooth or Wi-Fi adapter. The mobile device can then establish a connection to the single board computer (Pi 0) through its respective wireless adapter.

Using the onboard processor on the single board computer, data is processed and sent to the input ports (HDMI) of the display (projector) so that the data can be shown on the physical display on the windshield. Figure 19 shows Hardware Block Diagram Level 4, the lowest level visual for the hardware in this design.

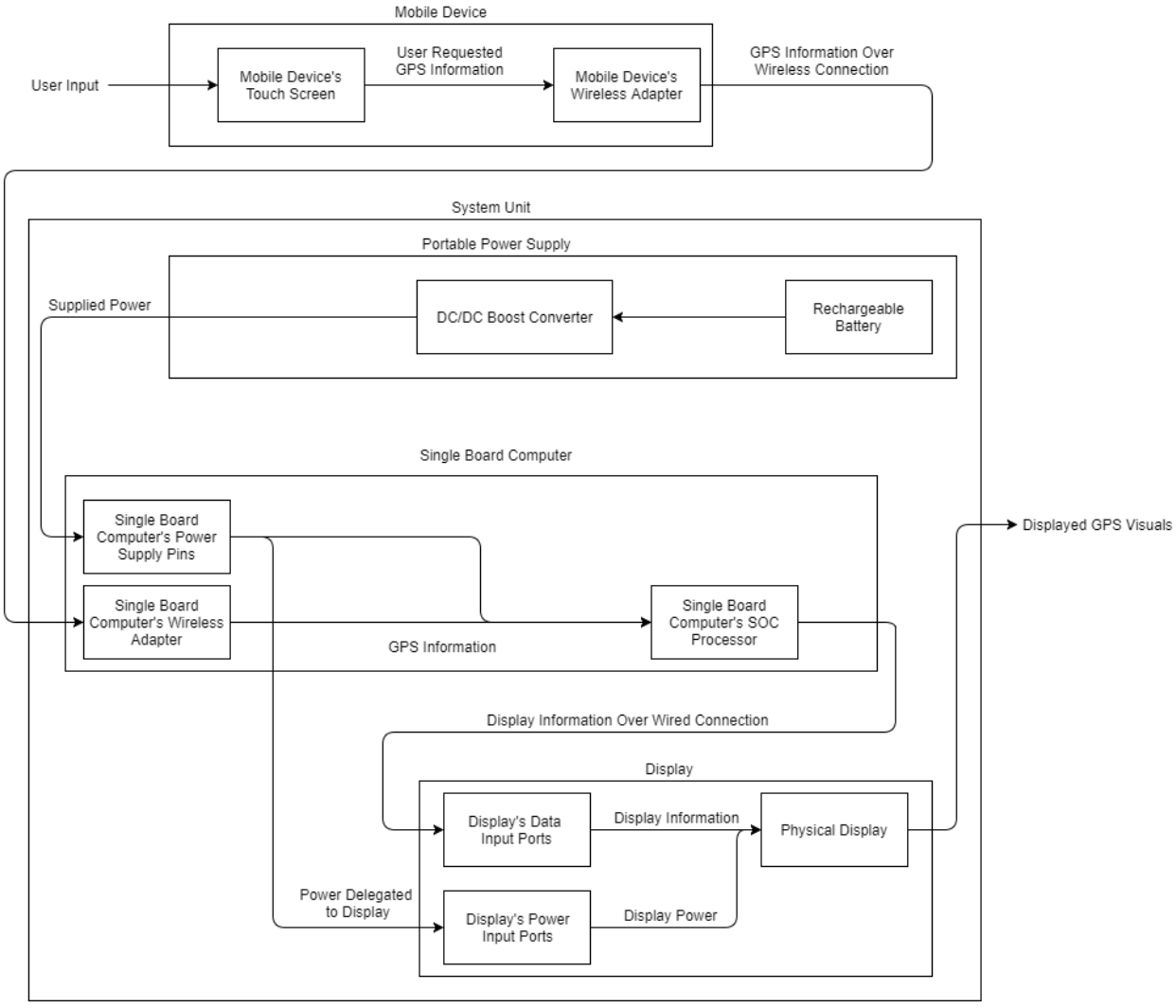


Figure 19: Hardware Level 4 Block Diagram

Module	Mobile Device's Touch Screen
Designer	NS / JH
Inputs	User Input
Outputs	User Requested GPS Information
Description	The touch screen is the element of hardware in the mobile device that takes in user input directly. It translates this input into the user requests for the rest of the phone to process.

Table 42: Hardware Level 4 Block Diagram – Mobile Device's Touch Screen

Module	Mobile Device's Wireless Adapter
Designer	NS / JH
Inputs	User Requested GPS Information
Outputs	GPS Information Over Wireless Connection
Description	The mobile device uses its wireless adapter to send data over to the rest of the HUD. It does this over a wireless connection that it makes with the rest of the System Unit.

Table 43: Hardware Level 4 Block Diagram – Mobile Device's Wireless Adapter

Module	Portable Power Supply
Designer	NS / JH
Inputs	None
Outputs	Supplied Power
Description	The portable power supply is the single element that supplies power to the System Unit.

Table 44: Hardware Level 4 Block Diagram – Portable Power Supply

Module	Single Board Computer's Wireless Adapter
Designer	NS / JH
Inputs	Wi-Fi Direct Signal
Outputs	Received Data
Description	The wireless adapter in the single board computer is responsible for receiving the GPS data from the mobile device. It then sends the data to the processor so the appropriate GPS data can be sent to the screen display.

Table 45: Hardware Level 4 Block Diagram – Single Board Computer's Wireless Adapter

Module	Single Board Computer's Power Supply
Designer	NS / JH

Inputs	Supplied Power
Outputs	Power Delegated to Display
Description	The power supply is the part of the System Unit that ensures that each individual element of the System Unit gets powered, including the display.

Table 46: Hardware Level 4 Block Diagram – Single Board Computer's Power Supply

Module	Single Board Computer's SOC Processor
Designer	NS / JH
Inputs	Received Wi-Fi Data
Outputs	Data Sent to Screen Display
Description	The processor is responsible for managing the received GPS data. It uses its programmed logic to transform the received wireless data into information that creates the appropriate visual display.

Table 47: Hardware Level 4 Block Diagram – Single Board Computer's SOC Processor

Module	Display's Input Ports
Designer	NS / JH
Inputs	Display Information Over Wired Connection, Power Delegated to Display
Outputs	Display Information
Description	The display's input ports take in everything that the display needs to show the proper visuals to the user. The input ports are where the display is powered, and where the data from the rest of the System Unit is received.

Table 48: Hardware Level 4 Block Diagram – Display's Input Ports

Module	Physical Display
Designer	NS / JH
Inputs	Display Information
Outputs	Displayed GPS Visuals
Description	The physical display is the part of the design that the end user is able to see. It displays all of the GPS data that it receives from the input ports.

Table 49: Hardware Level 4 Block Diagram – Physical Display

4.2.6 System Schematic

(JH)

To power the Raspberry Pi at least 5V DC is required to provide the necessary power for operation. However, for the chosen batteries, the best fit is two Lithium Polymer batteries that

both have a maximum voltage of 3.7V DC. By running these two batteries in parallel the operational time of the System Unit can be doubled without using a significant amount of space. The maximum current output of the batteries in parallel is 2.5A, which is higher than what is required. Because of this, a DC-DC boost converter is required to raise the voltage from 3.7V DC to 5V DC. This causes a slight drop in current to around 2A but does not impeded the function of the System Unit. At maximum capacity, the Raspberry Pi and the projector draw a combined 2A, which fits perfectly with the chosen DC voltage amplifier. Figure 20 shows the schematic for the hardware design.

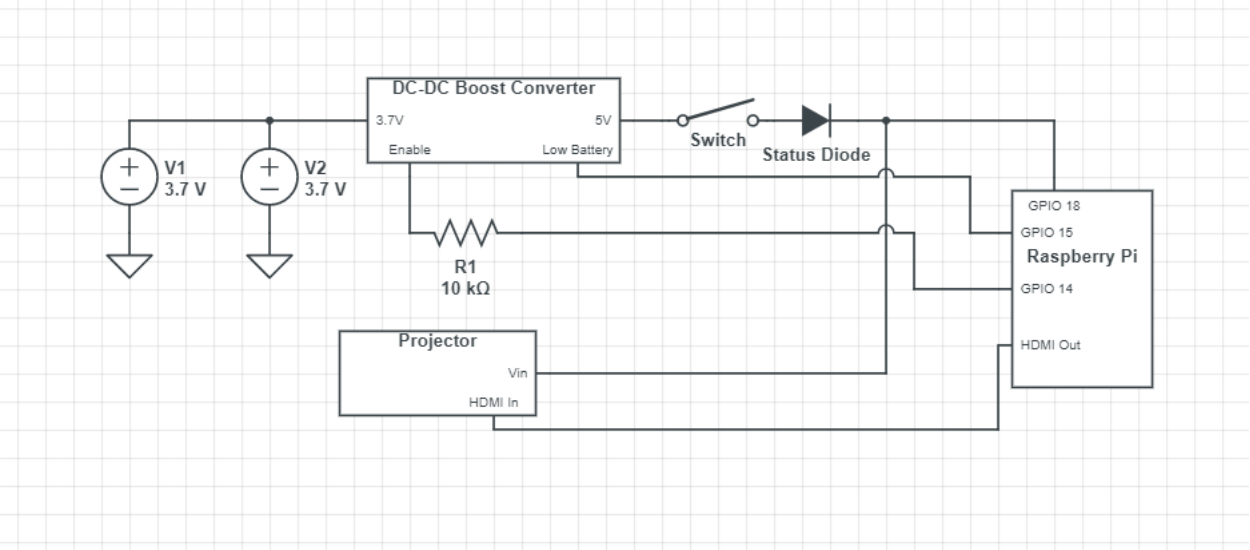


Figure 20: System Schematic

4.3 Engineering Calculations

Estimated Operational Time:

2*3800mAH = 7600mAH = 7.6AH (Maximum Amp Hours of Two Lithium Polymer Batteries in Parallel)

1A (Raspberry Pi Current Draw) + 1A (Projector Current Draw) = 2A

7.6AH / 2A = 3.8 Hours

DC - DC Boost Requirements:

2A Supplied Minimum (Maximum Device Current Draw)

5V Supplied Minimum (Raspberry Pi and Projector Voltage)

4.5 Software Changes in Implementation

(AW)

Changes to the accepted software design were minimal during the implementation process. The primary change was the transition from Wi-Fi Direct (Wi-Fi Peer-to-Peer) to Bluetooth as the communication protocol used between the Mobile Device and the System. These changes are detailed in the above section, Section 4.1 Software Theory of Operation.

4.4 Hardware Changes in Implementation

(JH)

Through the implementation of the design hardware changed from concept to prototype and eventually final product. There were some initial concerns with using the battery that came with the original projector. This led to an attempt to create a DC-DC Boost or Step-Down schematic to level off the voltage of the various batteries we tested. However, final implementation used the most simplistic path and did not require any schematics at all. Instead of boosting and regulating voltage the final implementation used a 5V battery to charge the smaller projector battery. This allowed for the removal of any intricate hardware design and allowed for extended battery life. Initial calculations suggested that that the heads-up display would remain powered for 3.5 hours. With the final, implemented design the device stayed powered for over 4 hours during testing. The other benefit of the 2-battery design was that it was a safer solution. By moving the projector away directly from the 5V battery we no longer had to worry about burning

out its sensitive board. This allowed for not only a safer and more compact design but also simplified assembly.

5. Operation and Maintenance

(AW)

Operation Instructions:

1. Turn on the projector by pressing and holding the top right button on the top of the projector. This is the button located directly above the projector lens. To turn the projector off, press and hold the same button. See Figure 21.
2. Turn on the System Unit by flipping the switch located on the front of the System Unit, directly below the projector lens. Note that the projector must be turned on before this switch is flipped. See Figure 22: System Unit Power Switch.
3. Open the Mobile Application. If a default device has already been set, then the connection between the Mobile Device and System Unit will be automatically established. Otherwise, select the "Manage Connections" option from the app menu, scan for Bluetooth devices, and select the correct device.
4. Enter a destination into the search bar in the Mobile Application and select the desired destination from the results list. See Figure 24.
5. To view the route, select the "Route Steps" option beneath the map. To start navigation either select the "Directions" option. Navigation can be stopped at any time by pressing the "X" button in the top right corner of the screen. Pressing and holding this button forces the application into Offline Mode for testing and demonstration purposes. See Figure 25.
6. To charge the System Unit, plug a Micro-USB cable into the port located on the left side of the System Unit. See Figure 23.

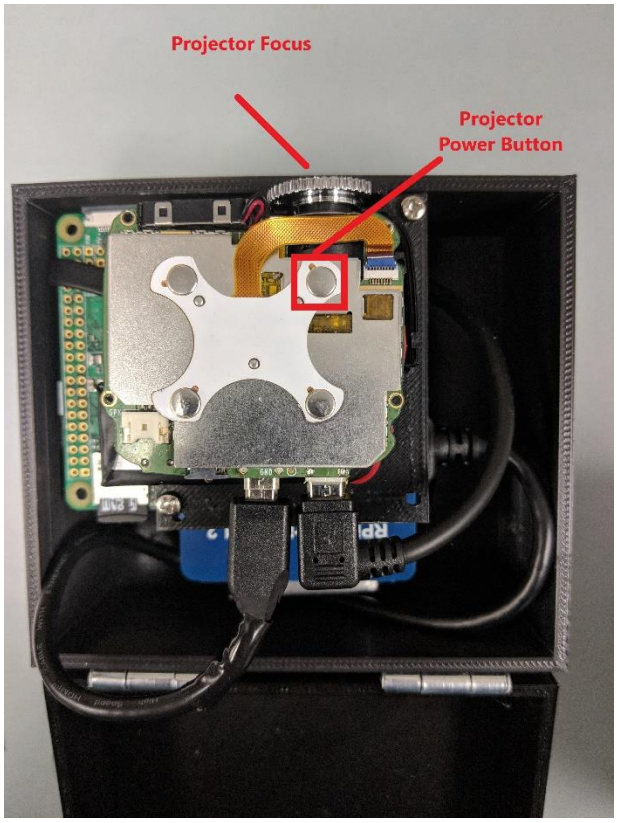


Figure 21: Projector Power Button

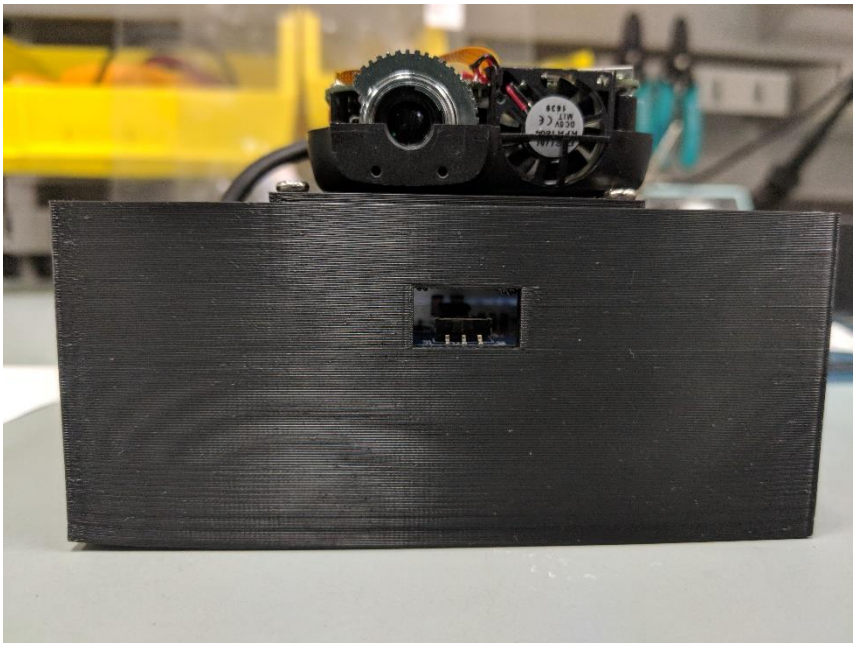


Figure 22: System Unit Power Switch

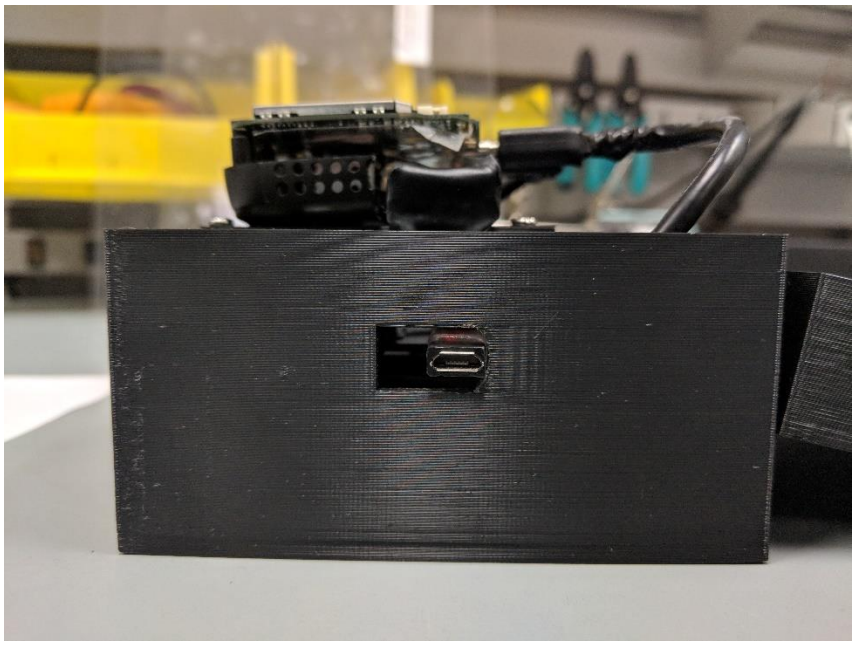


Figure 23: System Unit Charging Port

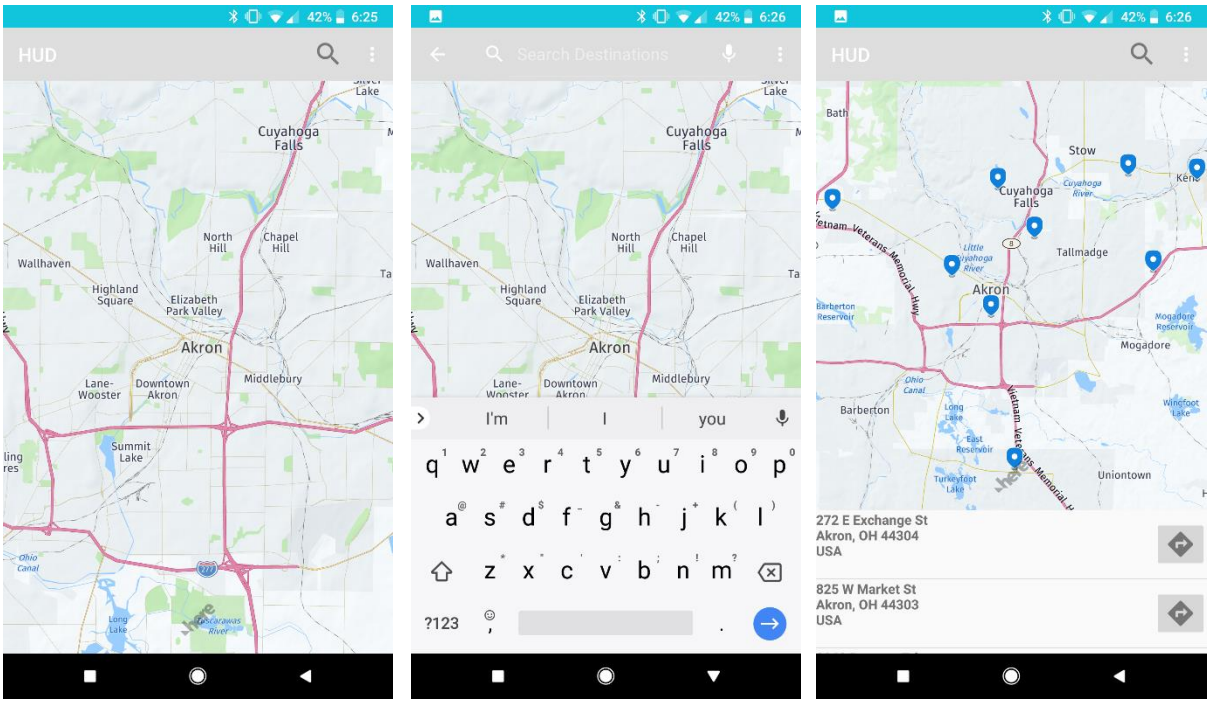


Figure 24: Mobile Application - Search Functionality

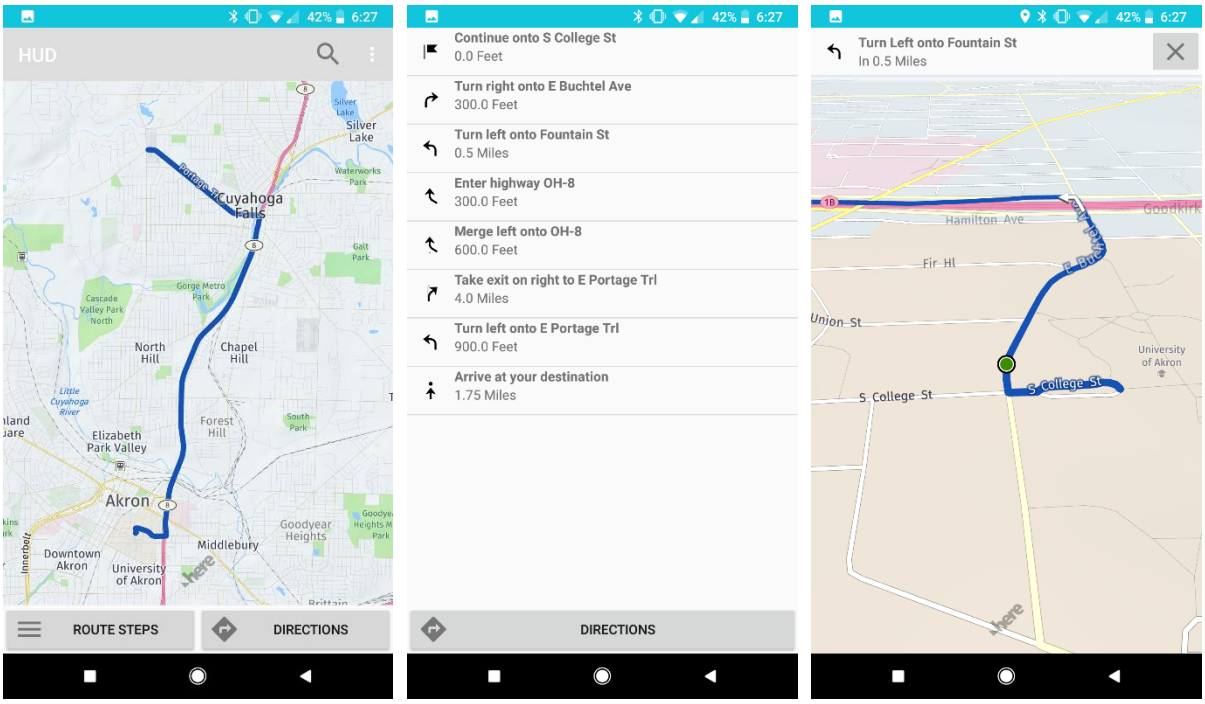


Figure 25: Mobile Application - Navigation

6. Testing Procedures

(AW)

A simulation mode was used within the Mobile Application to test possible navigation scenarios. The simulation mode spoofed the Mobile Device's GPS data in order to make it seem as the though device was moving along a selected route. This would force the application to send instruction information to the System Unit, allowing for verification of data and performance testing. Testing also included forcing instructions to be sent between the two components at varying frequencies, which provide crucial data on the stability, speed, and resiliency of the system. Aside from brute-force testing, unit tests were written to verify that the instructions received at the System Unit were complete and accurate. Timers were also used between the two components to test for latency and to provide crucial metrics that were used to improve performance.

7. Project Schedules

7.1 Gantt Chart Fall 2017

ID	Task Name	Duration	Start	Finish	Resource Names
1	**SDP1 fall2017**				
2	**Project Design**				
3	**Preliminary Design Report**				
4	**Problem Statement**	21 days	Tue 8/29/17	Mon 9/18/17	
5	Need	14 days	Tue 8/29/17	Mon 9/11/17	AW
6	Objective	14 days	Tue 8/29/17	Mon 9/11/17	AW,NS
7	Background	14 days	Tue 8/29/17	Mon 9/11/17	AW,NS,JR,JH
8	Marketing Requirements	14 days	Tue 8/29/17	Mon 9/11/17	AW,NS
9	Objective Tree	14 days	Tue 8/29/17	Mon 9/11/17	AW
10	Preliminary Design Gantt Chart	14 days	Tue 9/5/17	Mon 9/18/17	JR
11	**Block Diagrams Level 0, 1, ... w/ FR tables**	**14 days**	Tue 9/5/17	Mon 9/18/17	
12	Hardware modules (JH and NS)	14 days	Tue 9/5/17	Mon 9/18/17	JH,NS
13	Software modules (AW and JR)	14 days	Tue 9/5/17	Mon 9/18/17	AW,JR
14	Preliminary Design Presentation 9:55am	1 day	Tue 9/19/17	Tue 9/19/17	AW,NS,JR,JH
15	**Midterm Report**	**28 days**	**Tue 9/19/17**	**Mon 10/16/17**	
16	Design Requirements Specification	7 days	Tue 9/19/17	Mon 9/25/17	AW,NS,JR,JH
17	Midterm Design Gantt Chart	28 days	Tue 9/19/17	Mon 10/16/17	AW,NS,JR,JH
18	**Design Calculations**	**28 days**	**Tue 9/19/17**	**Mon 10/16/17**	
19	**Electrical Calculations**	**28 days**	**Tue 9/19/17**	**Mon 10/16/17**	
20	Power, Voltage, Current	28 days	Tue 9/19/17	Mon 10/16/17	JH
21	**Block Diagrams Level 2 w/ FR tables & ToO**	**7 days**	**Tue 9/19/17**	**Mon 9/25/17**	
22	Hardware modules (identify designer)	7 days	Tue 9/19/17	Mon 9/25/17	NS,JH
23	Software modules (identify designer)	7 days	Tue 9/19/17	Mon 9/25/17	AW,JR
24	**Block Diagrams Level 3 w/ FR tables & ToO**	**7 days**	**Tue 9/19/17**	**Mon 9/25/17**	
25	Hardware modules (identify designer)	7 days	Tue 9/26/17	Mon 10/2/17	NS,JH
26	Software modules (identify designer)	7 days	Tue 9/26/17	Mon 10/2/17	AW,NS
27	**Block Diagrams Level N+1 w/ FR tables & ToO**	**7 days**	**Tue 9/19/17**	**Mon 9/25/17**	
28	Hardware modules (identify designer)	7 days	Tue 10/3/17	Mon 10/9/17	NS,JH
29	Software modules (identify designer)	7 days	Tue 10/3/17	Mon 10/9/17	AW,NS
30	Midterm Design Presentations 9:55-11:35am Part 1	1 day	Tue 10/17/17	Tue 10/17/17	AW,NS,JR,JH
31	Midterm Design Presentations 9:55-11:35am Part 2	1 day	Tue 10/24/17	Tue 10/24/17	AW,NS,JR,JH
32	Project Poster	14 days	Tue 10/31/17	Mon 11/13/17	
33	**Final Design Report**	**42 days**	**Tue 10/17/17**	**Mon 11/27/17**	
34	Abstract	42 days	Tue 10/17/17	Mon 11/27/17	JR
35	**Software Design**	**42 days**	**Tue 10/17/17**	**Mon 11/27/17**	
36	**Modules 1...n**	**42 days**	**Tue 10/17/17**	**Mon 11/27/17**	
37	Psuedo Code	42 days	Tue 10/17/17	Mon 11/27/17	AW,NS,JR,JH
38	**Hardware Design**	**42 days**	**Tue 10/17/17**	**Mon 11/27/17**	
39	**Modules 1...n**	**42 days**	**Tue 10/17/17**	**Mon 11/27/17**	
40	Schematics	42 days	Tue 10/17/17	Mon 11/27/17	JH
41	Parts Request Form	42 days	Tue 10/17/17	Mon 11/27/17	JR
42	Budget (Estimated)	42 days	Tue 10/17/17	Mon 11/27/17	AW,NS,JR,JH
43	Implementation Gantt Chart	42 days	Tue 10/17/17	Mon 11/27/17	JR
44	Conclusions and Recommendations	42 days	Tue 10/17/17	Mon 11/27/17	JH
45	Final Design Presentations 9:55-11:35am Part 1	1 day	Tue 11/28/17	Tue 11/28/17	AW,NS,JR,JH
46					

Page 1

Figure 26: Gantt Chart Fall 2017

7.2 Gantt Chart Spring 2018

Task Name	Duration	Start	Finish		Resource Names
SDPII Implementation 2017	**105 days**	**Tue 1/16/18**	**Mon 4/30/18**		
Revise Gantt Chart	14 days	Tue 1/16/18	Mon 1/29/18		JR
Implement Project Design	**97 days**	**Tue 1/16/18**	**Sun 4/22/18**		
Hardware Implementation	**56 days**	**Tue 1/16/18**	**Mon 3/12/18**		
Testing WPF schemes on windshield	22 days	Tue 2/6/18	Tue 2/27/18		JR,NS
Revise Display Hardware	1 day	Wed 2/28/18	Wed 2/28/18	5	NS,JR
Create mount for final display	19 days	Thu 3/1/18	Mon 3/19/18	6	JR
Assemble Hardware	13 days	Tue 3/20/18	Sun 4/1/18	7	JH
Test Hardware	14 days	Mon 4/2/18	Sun 4/15/18	8	JH
Revise Hardware	14 days	Mon 4/2/18	Sun 4/15/18	8	JH
MIDTERM: Demonstrate Hardware	5 days	Mon 4/16/18	Fri 4/20/18	9	JH,NS,JR,AW
SDC & FA Hardware Approval	0 days	Mon 3/12/18	Mon 3/12/18		Dr.Tran
Software Implementation	**102 days**	**Tue 1/16/18**	**Fri 4/27/18**		
Map Display on App	1 day?	Tue 1/23/18	Tue 1/23/18		AW
Connection between Pi and HUD App created and passing data	8 days?	Tue 1/16/18	Tue 1/23/18		NS,JH
App connection settings have been created within the HUD App	15 days?	Tue 1/16/18	Tue 1/30/18		JR
JSON structure created and can parse test.txt file	1 day?	Thu 2/1/18	Thu 2/1/18		JH
WPF can display info from JSON structure	17 days?	Tue 1/16/18	Thu 2/1/18		JR
All data elements are able to be pulled from Here Maps	15 days?	Tue 1/30/18	Tue 2/13/18		AW
All data elements are able to be sent to Pi	10 days?	Tue 2/13/18	Thu 2/22/18		AW
All data elements are able to be parsed by JSON	7 days?	Sat 2/24/18	Fri 3/2/18		JH
Displaying all data elements from WPF template	1 day?	Sat 3/10/18	Sat 3/10/18	21	JR
Working Mobile HUD	67 days	Tue 1/16/18	Fri 3/23/18		JH,NS,JR,AW
Revise (Optimize) Software	16 days	Sun 4/1/18	Mon 4/16/18		JH,NS,JR,AW
MIDTERM: Demonstrate Software	5 days	Tue 4/17/18	Sat 4/21/18	24	JH,NS,JR,AW
SDC & FA Software Approval	0 days	Mon 3/12/18	Mon 3/12/18		Dr.Tran
System Integration	**90 days**	**Tue 1/16/18**	**Sun 4/15/18**		
Test Complete System	21 days	Mon 3/26/18	Sun 4/15/18		JH,NS,JR,AW
Demonstration of Complete System	7 days	Tue 1/16/18	Mon 1/22/18		JH,NS,JR,AW
Develop Final Report	**105 days**	**Tue 1/16/18**	**Mon 4/30/18**		
Write Final Report	98 days	Tue 1/16/18	Mon 4/23/18		JH,NS,JR,AW
Submit Final Report	0 days	Mon 4/23/18	Mon 4/23/18	31	JH,NS,JR,AW
Spring Recess	7 days	Mon 3/26/18	Sun 4/1/18		JH,NS,JR,AW
Project Demonstration and Presentation	0 days	Mon 4/23/18	Mon 4/23/18		

Figure 27: Gantt Chart Spring 2018

8. Parts List

Qty.	Refdes	Part Num.	Description
1	Raspberry Pi 3	3005	The Pi 3 is our choice for a microcontroller.
1	MiniProjector	x	This device will display the HUD metrics on the windshield
1	Backdrop Film	x	Transparent film that will be used as a backdrop for the projector
2	Battery	x	This will power the HUD device and projector.
1	Mount	x	This will properly hold up the projector at the correct angle and keep the HUD secure on the dash.
1	Battery Charger	259	This will charge the battery
1	Through Hole Hobby Board	x	This will be used to help connect the battery to the Pi
1	Power Boost	2465	This will take a 3.7V input and will convert to 5.2V output

Table 50: Parts List

9. Budget

Qty.	Part Num.	Description	Unit Cost	Total Cost
1	3005	The Pi 3 is our choice for a microcontroller.	$35.00	$35.00
1	x	This device will display the HUD metrics on the windshield	15.00	15.00
1	x	Transparent film that will be used as a backdrop for the projector	20.00	20.00
2	x	This will power the HUD device and projector.	23.99	47.98
1	x	This will properly hold up the projector at the correct angle and keep the HUD secure on the dash.		
1	259	This will charge the battery	12.50	12.50
1	x	This will be used to help connect the battery to the Pi		
1	2465	This will take a 3.7V input and will convert to 5.2V output	19.95	19.95

Table 51: Budget

10. Conclusion and Recommendations

The navigational heads-up display is designed with the intention for any driver to be able to utilize their Android smartphone to see GPS information on their windshield instead of just on their phone. The greatest element of design is the software behind the Mobile Application, and the software behind the System Unit. The UI, the communication, and the various services running within the system required a significant amount of design and testing to make them efficient. Time and space complexities are two key concepts that must always be considered while developing software for any system. These concepts just say that software should be as fast, and as lightweight, as possible. These concepts were key in the development of the HUD system, as they were fundamental in providing a clean, seamless user experience. Overall, the

HUD was successfully implemented, and proved to be a viable product. The original design went mostly unchanged and worked as expected.

Recommendations for future implementations of this design involve improving the hardware as technology improves. The main issue with the final implementation of this project was visibility. The projector that was selected and used was not as bright as it needed to be to ensure that the instructions are always visible, regardless of sunlight and other factors. With a brighter projector, the display would have been more visible, but given the time, resources, and available technology, there was not another option. For future design teams interested in designing a similar product, it is recommended that they investigate alternative display options – such as transparent displays. Another issue with the projector is the fact that the chosen projector was large, which caused the System Unit to become oversized. The battery was also a small blemish within the System Unit, as it did not last as long as desired. With more efficient hardware components, the system unit mounted on the user's windshield becomes smaller, more efficient, more user friendly, and ultimately a better product.

11. Design Team Information

- **Alex Walenchok**
    - Software Manager
    - Computer Engineering
    - Primary Focus: Mapping/Routing, Mobile Application, and design for the System Unit casing.
- **Joshua Humphrey**
    - Hardware Manager
    - Electrical Engineering
    - Primary Focus: Hardware design and instruction handling on the System Unit.
- **Joshua Reed**
    - Project Manager
    - Computer Engineering
    - Primary Focus: System Unit UI and connection establishment.
- **Nicholas Seifert**
    - Archivist
    - Computer Engineering
    - Primary Focus: Connection establishment, performance improvements, and instruction handling.

12. References

[1] Choi, Inseok. "Low-power Color TFT LCD Display for Hand-held Embedded Systems."ACM Digital Library. ACM, n.d. Web. 16 Mar. 2017.

[2] Engelsberg, Andreas, Sven Bauer, and Holger Kussmann. Windshield Display for a Navigation System. Robert Bosch GmbH, assignee. Patent US6735517B2. 11 May 2004.

[3] Kim, Seungjun & Dey, Anind. (2009). Simulated augmented reality windshield display as a cognitive mapping aid for elder driver navigation. Conference on Human Factors in Computing Systems - Proceedings. 133-142. 10.1145/1518701.1518724.

[4] Kim, Seungjun, and Anind K. Dey. "Simulated Augmented Reality Windshield Display as a Cognitive Mapping Aid for Elder Driver Navigation." Proceedings of the 27th International Conference on Human Factors in Computing Systems - CHI 09 (2009).

[5] Kruijff, Ernst, Edward Swan, II, and Steven Feiner. "Perceptual Issues in Augmented Reality Revisited" 9th IEEE International Symposium on Mixed and Augmented Reality (ISMAR) (2010): IEEE Xplore, 22 Nov. 2010. Web. 16 Mar. 2017.

[6] Lee, Jin-Shyan, Yu-Wei Su, and Chung-Chou Shen. "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi." Industrial Electronics Society (IECON). 33rd Annual Conference of the IEEE (2007): IEEE Xplore, 2008.

[7] LIM, Sang-Hyeok, Gum-Ho KIM, and Yu-Seung Kim. "Patent US20130051615 - Apparatus and Method for Providing Applications along with Augmented Reality Data." Google Books. N.p., 23 Dec. 2011. Web. 16 Mar. 2017.

[8] Nurseitov, Nurzhan, et al. “Comparison of JSON and XML Data Interchange Formats: A Case Study.” *Comparison of {JSON} and {XML} Data Interchange Formats: {A} Case Study*,8 Mar. 2010, pp. 157–162.

[9] Oliveira, Wellington, Weslley Torres, and Fernando Castor. "Native or Web? A Preliminary Study on the Energy Consumption of Android Development Models." IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) (2016): IEEE Xplore, 2016.

[10] Paul, Ian. “Wi-Fi Direct vs. Bluetooth 4.0: A Battle for Supremacy.” *PCWorld,* PCWorld, 26 Oct. 2010

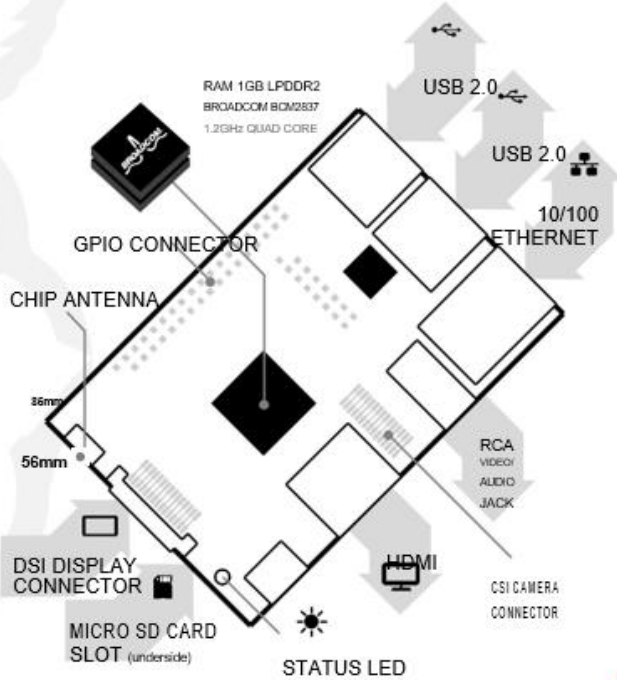
13. Appendix

13.1 Raspberry Pi 3

Raspberry Pi

Raspberry Pi 3 Model B

Product Name	**Raspberry Pi 3**
Product Description	The Raspberry Pi 3 Model B is the third generation Raspberry Pi. This powerful credit-card sized single board computer can be used for many applications and supersedes the original Raspberry Pi Model B+ and Raspberry Pi 2 Model B. Whilst maintaining the popular board format the Raspberry Pi 3 Model B brings you a more powerful processer, 10x faster than the first generation Raspberry Pi. Additionally it adds wireless LAN & Bluetooth connectivity making it the ideal solution for powerful connected designs.
RS Part Number	**896-8660**



RS

Raspberry Pi

Raspberry Pi 3 Model B

Specifications

Processor	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	1GB LPDDR2
Operating System	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V1, 2.5A

Connectors:

Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4 Composite RCA (PAL and NTSC)
Audio Output	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Push/pull Micro SDIO

Key Benefits

- Low cost
- Consistent board format
- 10x faster processing
- Added connectivity

Key Applications

- Low cost PC/tablet/laptop
- IoT applications
- Media centre
- Robotics
- Industrial/Home automation
- Server/cloud server
- Print server
- Security monitoring
- Web camera
- Gaming
- Wireless access point
- Environmental sensing/monitoring (e.g. weather station)

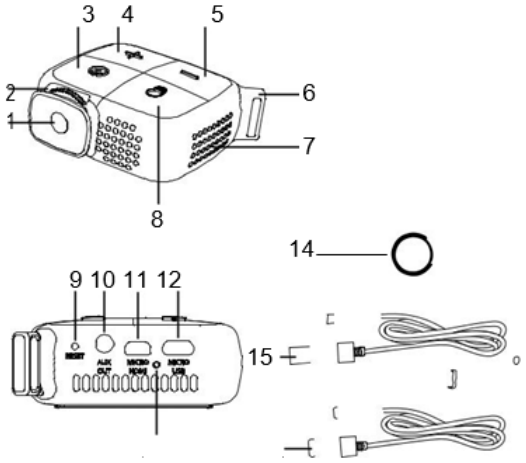
RS

13.2 Projector

LOCATION OF PARTS AND CONTROLS

1. Projector Lens/Flashlight
2. Focus Wheel
3. Power Button
4. Volume + Button
5. Volume - Button
6. Key Ring Holder
7. Speakers
8. Flashlight Button
9. Reset Button
10. Aux Out Port
11. Micro HDMI® Port
12. Micro USB Port
13. Charging LED Indicator
14. Key Ring
15. Micro USB Charging Cable
16. Micro HDMI® Cable

LOCATION OF PARTS AND CONTROLS



SPECIFICATIONS

Battery . . . . . 1000mAh Lithium-Polymer

Brightness . . . . . 5 Lumens

Native Resolution . . . . . 320 x 240 (720p)

Aspect Ratio 4:3

Focus . . . . . Manual

Charging time . . . . . 2 hours (approximately)

Battery life . . . . . 1 hour (approximately)

Weight . . . . . 0.2 lbs (92g)

Dimensions . 2.4"(61.7mm) w x 2.3"(57.7mm) d x 1" (25.5mm) h

13.3 Battery and Power Boost

Parameters	Value

Battery Capacity	3800mAh Maximum
Output Current	2.0A
Output Voltage	5.0V
Output Ports	USB2.0x5
Standard Charging Current / Voltage	1.0A/5.0V
USB Data Ports	USB2.0 X 4 / microUSB x1
Size	86.60mm x 56.00mm x 18.45mm
Weight	104.51g