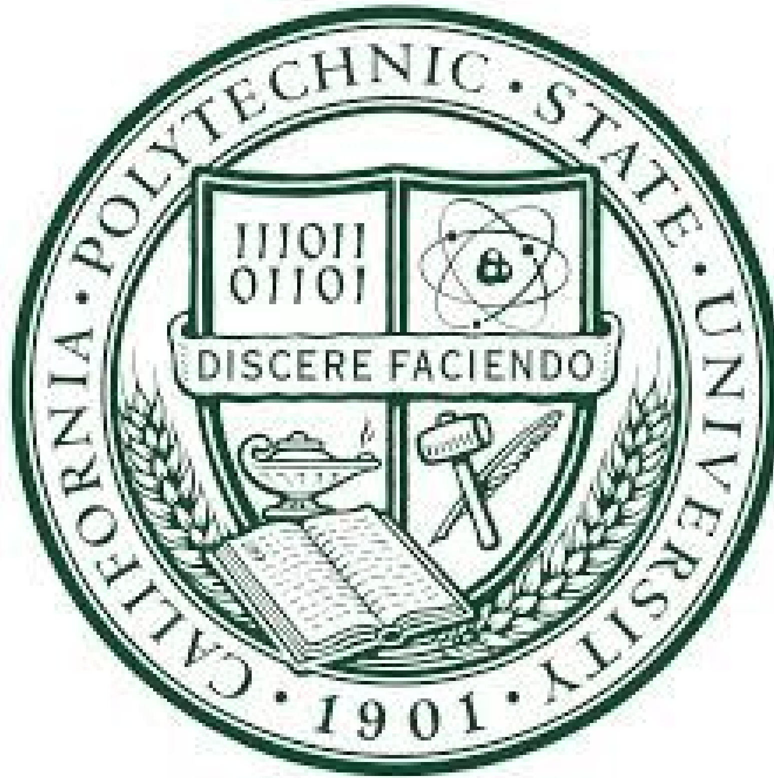


IoT Asset Tracker

By

Matt Murray & Jonny Erickson

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

November 2020

TABLE OF CONTENTS

Section ***page***

Acknowledgement.. 4

Abstract.. 4

Chapter 1: Introduction.. 5

Chapter 2: Customer Needs, Requirements and Specifications.................................... 6

Chapter 3: Functional Decomposition... 8

Chapter 4: Project Planning... 11

Chapter 5: Hardware Design... 13

5.1: BG96.. 13

5.2: Microcontroller... 16

5.3: User Interface... 16

Chapter 6: Software Design... 18

6.1: Finite State Machine Using GUI... 18

6.2: Application Pages.. 19

6.3: Embedded Modules.. 25

6.4: Globally Declared Functions... 27

Chapter 7: Conclusions.. 30

References... 31

Appendices

A. Senior Project Analysis.. 32

B. Project Code... 37

LIST OF FIGURES

Figure ***page***

Figure 1: IoT Asset Tracker Level 0 Diagram.......... 8
Figure 2: IoT Asset Tracker Level 1 Diagram.......... 9
Figure 3: Main Hardware Functional Blocks.......... 13
Figure 4: Quectel BG96 EVB.......... 13
Figure 5: Elecrow 5in Touch Screen.......... 17
Figure 6.1: Overall Application Flowchart.......... 18
Figure 6.2: Start Page.......... 19
Figure 6.3: Initialization.......... 19
Figure 6.4: Phone Number Request.......... 20
Figure 6.5: Invalid Phone Number.......... 20
Figure 6.6: Mode Select.......... 21
Figure 6.7: Tracking Selected.......... 21
Figure 6.8: Geofence Selected.......... 22
Figure 6.9: Tracking Active.......... 23
Figure 6.10: Geofence Active.......... 24

LIST OF TABLES

Table ***page***

Table I: IoT Asset Tracker Specifications 7
Table II: IoT Asset Tracker Level 0 Definition.......... 8
Table III: IoT Asset Tracker Level 1 Definition.......... 9
Table IV: IoT Asset Tracker Timeline.......... 11
Table V: IoT Asset Tracker Gantt Chart.......... 11
Table VI: IoT Asset Tracker Cost Estimate.......... 12
Table VII: Initialization AT-Commands Description.......... 14
Table VIII: Send Text AT-Commands Description.......... 15
Table IX: Location and Geofence AT-Commands Description.......... 15

Acknowledgements:
This project was partially sponsored by Quectel North America with their donation of the BG96 cellular and GNSS module. Without them this project would not have been possible. Furthermore we would like to acknowledge the following people for their support

Dr. Callenes (Cal Poly) - For advising and supervising our project through multiple changes
Dave Hanrahan (Quectel) - For assisting with FAE support on BG96 integration

Abstract

This senior project will be the completion of an IoT asset Tracker. In today's day and age, technology is weaving its way into every aspect of life. Because of this, we demand to know more and more information about everything that is going on in our lives. One of these needed pieces of information is knowing the location of our most valuable assets. This is the main goal of this project: to build a portable SMS driven GNSS tracking device. The device is designed to provide two methods of tracking for any mobile asset.

The device will utilize a large touchscreen for easy user manipulation and a Cat-M1 cellular module for wireless communication over SMS. The first method of tracking uses a repeated location ping that uses only a user defined phone number and interval to begin operation. The second method of tracking utilizes a geofence to detect whether the device has left a predetermined radius based off of the initial location of initialization. Using either of these methods the device will provide the user an easy and reliable solution to any of their tracking needs.

Chapter 1: Introduction

Asset tracking is one of the many possible solutions provided by GPS technology. For Matt and Jonny, GNSS and cellular technologies seemed like an interesting subset of electrical engineering to learn more about. As we discussed possible projects, we decided that we wanted to work on something that would more or less encompass a large part of the curriculum we have studied here at Cal Poly. There are obviously numerous possible projects, but creating an asset tracker seemed to touch on a lot of these areas we have been exposed to. Each of these areas would influence a fundamental piece of the end project. Our asset tracker was defined to have a custom GUI built in Python on a Raspberry Pi, which was controlled using a touch screen and communicates over a serial port to a GPS and cellular chipset, provided with power by a battery system and encapsulated in an acrylic housing. We were able to draw on experience from the early CPE programming series for the GUI, MCU-based system design for our serial communication and hardware level commands, power systems courses for the battery-life analysis, and IME156 for the housing. However, there were still many areas that we needed to learn entirely new skills. The cellular and GPS chipset are controlled using AT-commands, which we had not been previously exposed to. We also needed to learn the fundamentals of the Raspberry Pi in order to initialize communication ports, run programs on start-up, and calibrate the touch screen. All in all, we were able to combine these different knowledge sets and build a fully functioning asset tracker using all of the above functionality.

Chapter 2: Customer needs, Requirements and Specifications

Customer Needs Assessment

This project can be boiled down to a GPS tracking device and as such it should be effective at finding and reporting location data. Due to the nature of this product and how it pertains to consumer electronics we have decided that our product needs to be easy for the customer to operate and set up. The customer also needs to not log onto a computer to check the location of the asset, therefore the reporting of the location needs to be over SMS to decrease customer interaction time. This is a fairly passive tracking device and it needs to be stable and only require a few parameters from the user to operate for hours on end without fault. Furthermore, because we are tracking assets of many different shapes and sizes we need the device to be as small as possible while still being able to provide an accurate location. Finally, because this is a portable IoT device, it needs to have a large battery to decrease the impact on the customer when it comes to charging.

Requirements and Specifications

The IoT Asset Tracker has two main functions, The first being that it needs to accurately find and report location data. This works by using a new gnss module that uses the GPS, GLONASS and Galileo constellations. By using multiple satellite systems it is able to find a more accurate location fix. To further add to this functionality we must also easily report this data to the user. By using a cellular module we are able to text the location data to the user on their phone number. This eliminates the need to go online and check the location on a computer and making it easier to track the device. The second function is to have a very easy and logical way for the user to set up the device. To solve this the device will use a GUI on a 5in touchscreen to lead the user through the initialization steps without the need of a mouse or a full keyboard. This will allow the user to easily understand how the device operates and what information is needed from them to set the device up.

With these two goals and the requirements needed to accomplish them, we have compiled a list of required components that will complete the design of the device. One of these is a large battery that will provide enough power to run the device for up to 12 hours. Furthermore, we will want this device to be as small as possible. To accomplish this we will design a housing to secure our modules in a stacked orientation to save space. By incorporating all of these requirements we will create the functionality needed to build a working IoT tracking device

TABLE I

IoT Asset Tracker Specifications

Marketing Requirements	Engineering Specifications	Justification
1,4, 6	5in LCD touch screen with resistive touch and number keypad	The user will need a large screen to interact with and get information about the device. they will also need a keypad to input data
2	Integrated Cellular and GPS module (Quectel BG96)	High accuracy gps and cellular module is needed for location tracking and reporting
5	1700mAh Battery	A large battery is needed for long use case applications and increased charging intervals
3, 4	Compact and secure housing	Small housing is needed to make the device portable and compact
1, 4	Graphical User Interface	The device needs to run on a GUI so the user can easily understand and device initialization
Marketing Requirements 1. Understandable user interface 2. Accurate Location Data 3. Small Size 4. Easy setup 5. Long Battery Life 6. Text based Reporting		

The requirements and specifications table format derives from [1], Chapter 3.

Chapter 3: Functional Decomposition

IoT Asset TrackerLevel 0 Block Diagram

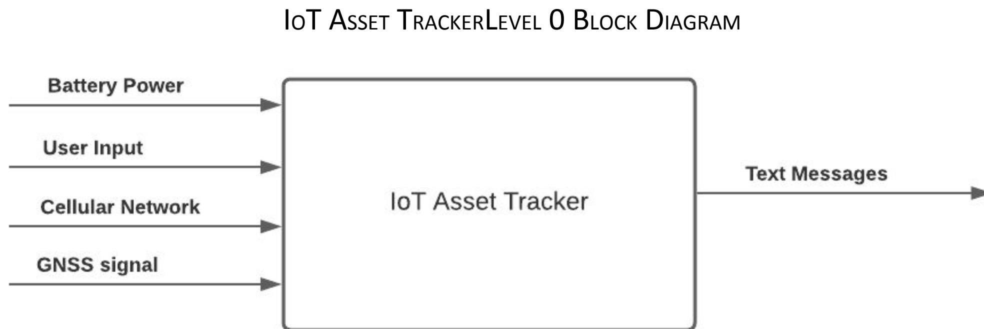


Figure 1: IoT Asset Tracker Level 0 Diagram

TABLE II
IoT Asset Tracker Level 0 Definition

	IoS Asset Tracker
Input:	User input – the user will input parameters to personalize functionality Cellular Network – The device uses the cellular network to send SMS GNSS signal – The device takes in GNSS satellite signals to find locations Battery Power- Battery input power for the device
Output:	Text Messages – The device outputs text messages
Functionality:	Using a one time setup from the user the device will run on the battery power to take in GNSS signals and locate the device. Then by using the data imputed by the user the device will send a text with the location of the device to the users phone number

IoT Asset Tracker Level 1 Block Diagram

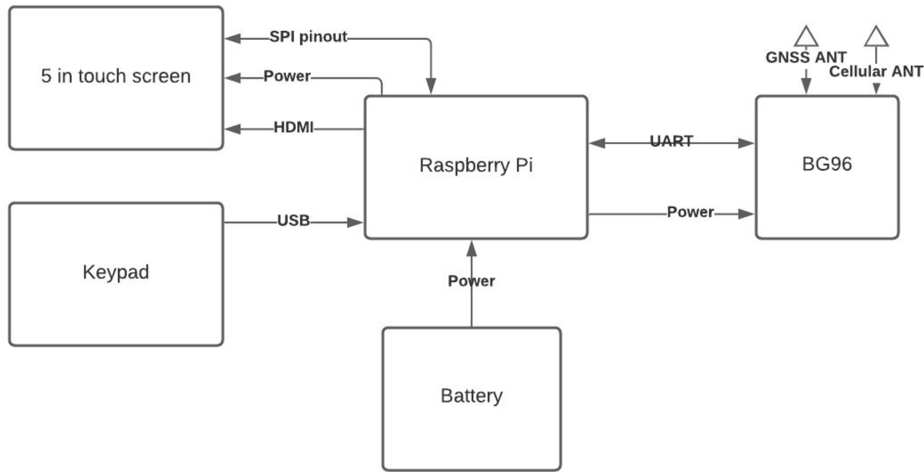


Figure 2: IoT Asset Tracker Level 1 Diagram

TABLE III
IoT Asset Tracker Level 1 Definition

	IoS Asset Tracker
Module 1	Raspberry Pi 3B+
Input(module1)	-battery power -USB input from user via the keypad -SPI communication buss from the touchscreen -UART responses from BG96 AT-Commands
Output(module 1)	-UART AT-Commands to control BG96 -HDMI output to control display -power to BG96 and Display
Functionality (module 1)	The Raspberry Pi is the brains of the project. It is driven by the inputs from the user via the touchscreen and keypad. It manages the power output to the devices as well as all initialization and commands to both the display and BG96
Module 2	BG 96 (integrated cellular and GNSS chipset)
Input(module2)	-UART AT-Commands -Power from raspberry pi -Cellular and GNSS antenna signals
Output(module 2)	-UART AT-Command responses
Functionality (module 2)	The BG96 is completely controlled by the raspberry pi. Upon receiving the correct sequence of commands the device can Locate a position or send a text.
Module 3	5 inch touchscreen
Input(module3)	-Power -HDMI input -SPI communication for initialization

Output(module 3)	-SPI output for touch screen output
Functionality (module 3)	the 5 in touch screen and display is used to display the GUI and to convert touches into selections for the raspberry pi. This device also creates a user defined state machine
Module 4	Keypad
Input(module4)	-User Input
Output(module 4)	-USB(UART) communication of button presses
Functionality (module 4)	The keypad is used to input data such as phone numbers, delays and radiuses that are needed for correct usage of the device.

Chapter 4 Project Planning

TABLE IV
IoT Asset Tracker Timeline

Delivery Date	Deliverable Description
4/10/2020	Design Review
5/22/2020	EE 461 demo
5/28/2020	EE 461 report
6/6/2020	Design revision 1 finished
9/14/2020	Design revision 2 finished
11/2/2020	ABET Sr. Project Analysis
11/16/2020	EE 462 Report

[1] R. Ford and C. Coulston, *Design for Electrical and Computer Engineers*, McGraw-Hill, 2007, p. 37
[2] *IEEE Std 1233, 1998 Edition*, p. 4 (10/36), DOI: 10.1109/IEEESTD.1998.88826

TABLE V
IoT Asset Tracker Gantt Chart

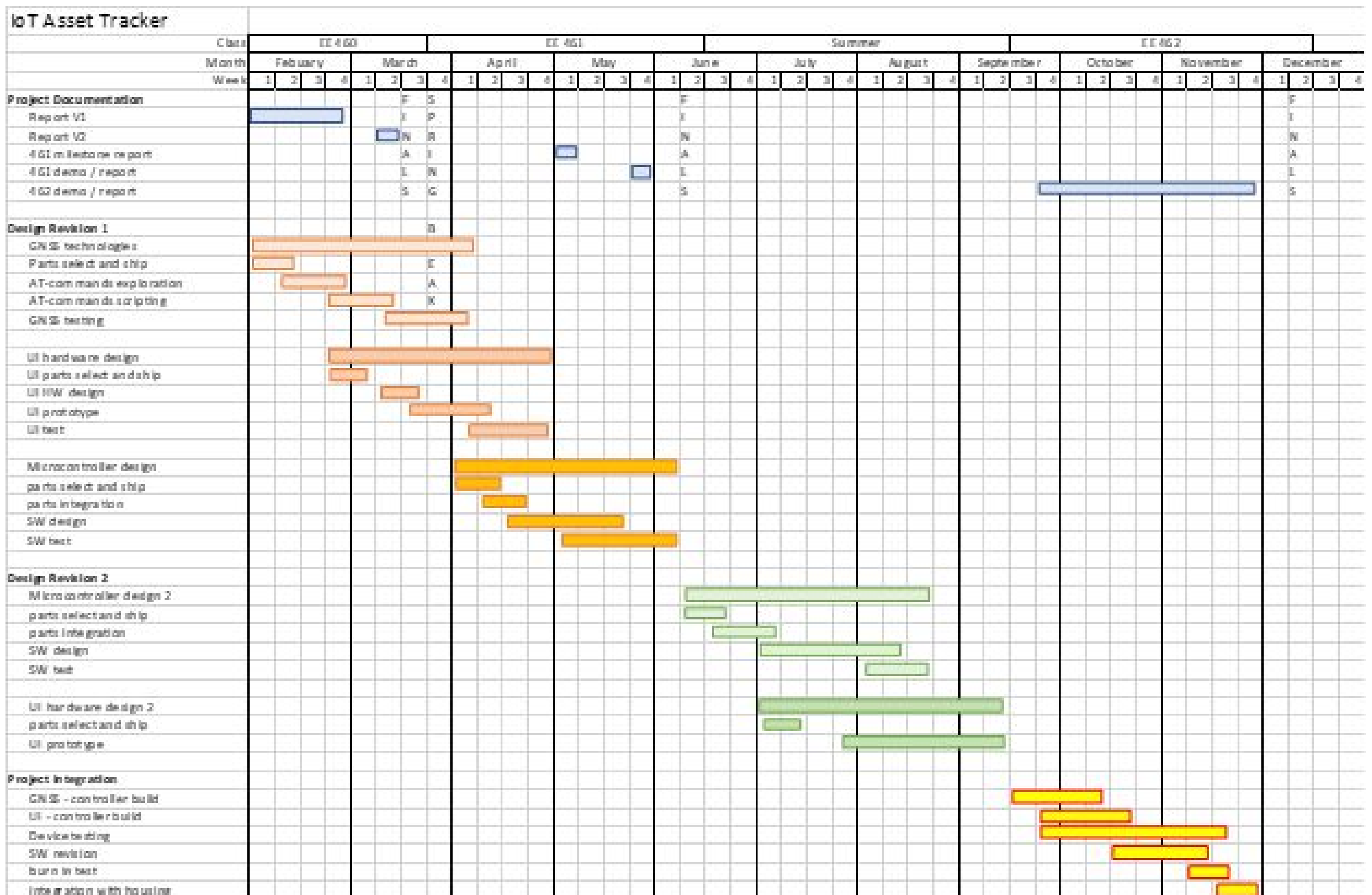


TABLE VI

IoT Asset Tracker Cost Estimate

Materials Costs

Manufacturer	Part Number	Description	Quantity	Cost per Unit	Total Cost
Quectel	BG96-TE-A+UMTS-LTE-EVB-KIT	LTE/GIS Evaluation Board	1	$ 116.00	$ 116.00
Element	Raspberry Pi 3 B+	Raspberry Pi Motherboard	1	$ 38.91	$ 38.91
Elecrow	RPA05010R	5 Inch RPI Touchscreen	1	$ 38.99	$ 38.99
Jelly Comb	B07CQGWNP2	Mini Number Pad - USB	1	$ 10.99	$ 10.99
RAVPower	RP-PB19	16750 mAh Power Bank	1	$ 23.99	$ 23.99
Benfei	000151BLACK	USB to Serial Cord (RS-232)	1	$ 8.59	$ 8.59
Optix	MC-17	20"x30"x0.093" Acrylic Sheet	1	$ 19.98	$ 19.98
GE	2708920	Clear Silicone Sealant Caulk	1	$ 6.57	$ 6.57
Total Materials Cost:					**$ 264.02**

Research and Development Costs

Job	Description	Hourly Rate	Number of Hours	Total Cost
Front-End GUI Design		$ 45.00	20	$ 900.00
Backend Software Design		$ 50.00	25	$ 1,250.00
Hardware Interfacing		$ 40.00	45	$ 1,800.00
Mechanical Design		$ 40.00	5	$ 200.00
Total Research and Development Cost:				**$ 4,150.00**

Manufacturing Labor Costs

Job	Description	Hourly Rate	Number of Hours	Total Cost
Enclosure Construction	Cutting and drilling of acrylic sheet, assembly of enclosure	$ 15.00	3	$ 45.00
Touch Screen Calibration	Downloading necessary drivers to Raspberry Pi and calibrating	$ 15.00	0.5	$ 7.50
Raspberry Pi Startup Initialization	Adding initialization code to the startup sequence on Rpi	$ 20.00	0.25	$ 5.00
Electrical Assembly	Wiring between RPi, BG96, Power Bank, Touch Screen, Keypad	$ 20.00	0.5	$ 10.00
Full Assembly	Attaching electrical assembly to the enclosure	$ 15.00	0.5	$ 7.50
Total Manufacturing Labor Cost:				**$ 75.00**

Total Costs

Type	Total Cost
Materials	$ 264.02
Manufacturing Labor	$ 75.00
Total Unit Cost:	**$ 339.02**
Research & Development	$ 4,150.00
Project Total Cost (Single Unit):	**$ 4,489.02**
Breakeven at 100 Units:	$ 380.52
Breakeven at 1,000 Units:	$ 343.17
Breakeven at 10,000 Units:	$ 339.44
Units sold to breakeven at $350 per Unit	**378 Units**

Chapter 5: Hardware Design

The design of the hardware falls into three major areas. They consist of the BG96, the Microcontroller and the User interface. These three blocks of hardware can be seen below in figure 3:

Figure3: Main Hardware Functional Blocks

5.1 BG96

Overview:
The core functionality of the IoT Asset tracker is the usage of a Quectel BG96 integrated Cellular and GNSS chip. The BG96 is an LTE Cat M1/Cat NB1 EGPRS module. The cellular portion of this chip allows for ultra low power consumption while providing access to global frequency bands for cellular applications. The GNSS functionality of the chip also provides access to the GPS, GLONASS, BeiDou, Galileo and Compass satellite constellations. Due to the abundance of satellites the chip is able to provide a DGNSS location fix that increases the accuracy of location data to <1m.

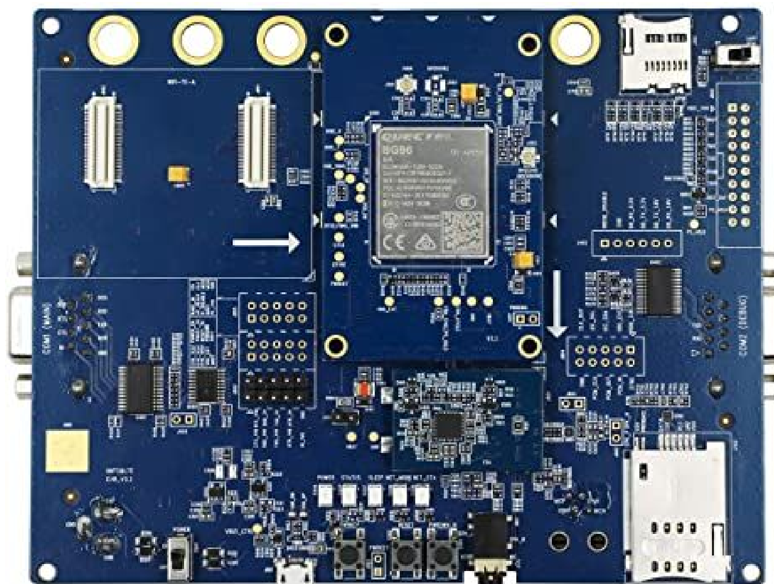


Figure 4: Quectel BG96 EVB

As seen above in figure 4 the BG96 is implemented on an evaluation board that allows us to use industry standard interfaces such as (USB, UART, SPI, I2C). In our project the BG96 will be controlled using AT-Commands on the main UART port. These AT-Commands are the way in which the chip is controlled[1][4].

AT-Commands:
The chip is controlled entirely by AT-Commands sent over the UART communication bus[4]. by using these commands we are able to perform three major functions on our BG96, these are initialization, sending SMS messages, and acquiring location data. These AT-commands can be seen below in table VI, VII, and VIII respectively.

The initialization sequence below is only executed once on the startup of the program. There are many different Initialization commands that can be given, but on either a warm or cold boot the device will retain its default parameters. This is especially useful when dealing with the UART port parameters as they do not need to be set and the device can receive commands as long as they come in with 115200 baud,8 data bits, 1 stop bit, no parity and no flow control[1].

TABLE VII
INITIALIZATION AT-COMMANDS DESCRIPTION

AT-COMMAND	DESCRIPTION
AT	Wake up UART channel
ATV1	Set response to verbose
ATE1	Enable Echo
AT+CMEE=2	Set error messages to verbose values
AT+IPR?	Query Baud rate (115200 standard)
ATI	Display product identification information
AT+GSN	Request International IMEI Number
ATI	Display product identification information
AT+CPIN?	Query need for SIM pin number
AT+CIMI	Return international mobile subscriber info
AT+CSQ	Return Signal Quality Report
AT+CREG?	Query Network Registration status
AT+CGREG?	Query Network Verbose registration
AT+COPS?	Display current cellular network
AT+QPSCFG=”OUTPORT”,”NONE”	Configure NMEA sentence outport
AT+QGPS?	Turn on GNSS

[1] “BG96 AT-Commands Manual.” *Quectel BG96*, 2018, www.quectel.com/product/bg96.htm.

TABLE VIII

SEND TEXT AT-COMMANDS DESCRIPTION

AT-COMMAND	DESCRIPTION
ATE1	Enable Echo
AT+CMGF=1	Set message format to text mode
AT+CSMP = 17,167,0,0	Set first octet to 17 and validity period to 167ms
AT+CSCS=”GSM”	Enable character set to GSM alphabet
AT+CMGS=”+1{PHONE NUMBER}	Enter recipient phone number
{MESSAGE}	Input given message
\x1A	Send message character

[1]“BG96 AT-Commands Manual.” *Quectel BG96*, 2018, www.quectel.com/product/bg96.htm.

TABLE IX

LOCATION AND GEOFENCE AT-COMMANDS DESCRIPTION

AT-COMMAND	DESCRIPTION
AT+QCFGEXT="ADDGEO",0,3,0,{LAT},{LON},{RADIUS}	Add geofence 0 with given lat,lon and radius
AT+QCFGEXT=”DELETEGO”,0	Delete geofence 0
AT+QCFGEXT=”QUERYGEO”,0	Query status of device and geofence
AT+QGPSLOC=0	Determine location fix of device

[2]“Quectel BG96 GNSS AT-Commands Manual.” *Quectel BG96*, www.quectel.com/product/bg96.htm.

As the initialization sequence is done sequentially the device echos the received command as well as any information related to said command. The responses from the device allow us to understand if the BG96 has been set up correctly. The main checkpoint is to see if we are registered to the network and that the carrier network is realized correctly. Once the BG96 is initialized correctly it is ready to receive commands from table VII and VIII to send text messages or manipulate the GNSS module. In our device we will use these AT commands to perform all of the needed functionality of the IoT Asset Tracker. The implementation of these functions can be seen in chapter 6 or in Appendix B

5.2 Microcontroller

Design Revision 1:
The central piece of the IoT Asset Tracker is the microcontroller that runs all of the functionality needed to meet our design requirements. We needed a chip that was easily programmable, low power and with enough functionality to run the user input devices as well as the BG96.

In our first design revision we chose to use the MSP432 due to our common knowledge from other classes. We believed that by using some of our premade modules that we would be able to easily meet all of our design requirements. We bagan and did all of the hardware design needed to incorporate a screen, keypad, buttons and the UART communication bus for the BG96. But after extensive testing we found that the MSP432 was unable to communicate with the BG96. This became a vexing problem as when we compared the waveforms from the MSP and from the computer, they were exactly the same. After many hours of testing we reached out to an FAE from Quectel and described our problem. The FAE indicated to us that because we had to use a UART level shifter, the BG96 would not validate the UART port and would not receive commands. He had also said that the UART port on the evaluation board required a very high output impedance and that our design was possibly not meeting this criteria. Because of this and our need to finish the project within a deadline we decided to move to a 2nd design revision

Design Revision 2:
In our second design revision we decided to use a microcontroller that would guarantee a fix to our UART problem. We chose to use a Raspberry Pi 3B+[6]. By increasing the complexity of our microcontroller we were able to add more features to our device without a drastic increase in cost. Furthermore, by using a raspberry pi we were able to upgrade our user interface to a large touch screen that displays our GUI. This can be seen in chapter 6.

The benefits of using a Raspberry Pi were realized almost immediately when we began to redesign our hardware architecture. We found that all of our devices could receive power and communication via the COM ports on the Pi. We were also able to decrease the software complexity by using python as the main language to run the device. And finally, the ability to debug our project increased drastically as we were able to test and run software developments quickly and effectively [6]

5.3 User Interfaces
The final key component of the hardware design is the user interface modules. These two modules are responsible for providing the user an easy and understandable way to input their data into the device. These two modules can be broken down into a display and a keypad, both attached to the microcontroller.

Display:
During our first design iteration we chose to work with the Newhaven 2x20 Character display. this was a screen that was already set up to work with the MSP432. However when we made the switch to the Raspberry Pi we found that it was not sufficient for instructing the user or as a reporting tool. Because of this we decided to switch to the elecrow 5in resistive touch display shown below in Figure 4:

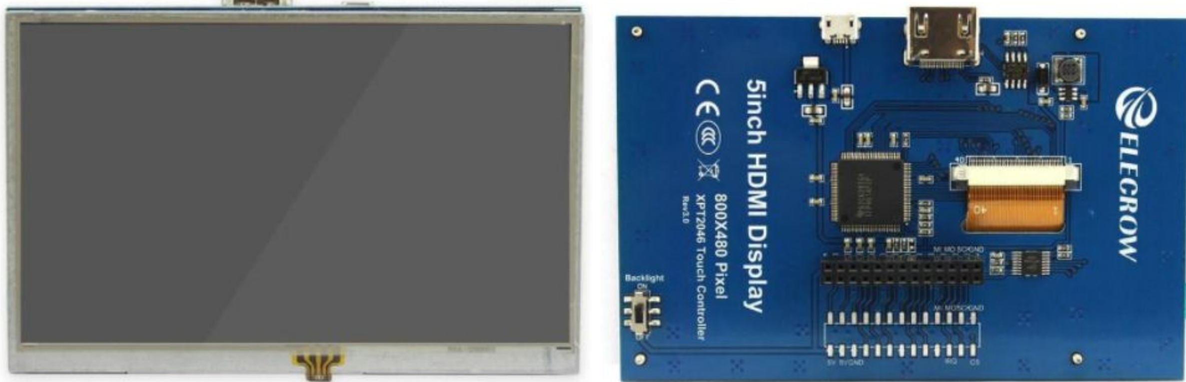


Figure 5: Elecrow 5in Touch Screen

The touch screen has two major ports of interest. The first being the HDMI input for video. and the second being the female pin headers for power and for outputting touch screen data to the Raspberry Pi. The screen has a Raspberry Pi footprint and is easily attached and integrated to the Raspberry Pi[3][6]

We made the decision to go for the lower cost resistive touch compared to the capacitive touch because the scope of our project does not call for a highly accurate touch screen. The main difference between these two technologies is that the capacitive touch consists of two layers of glass with a separating conductor. When a finger touches the screen the electrical properties change and a touch is registered. The use of the glass also adds durability to the screen. The resistive touch screen is very similar in nature but is formed by two synthetic layers separated by air. When touched the two layers collapse and a touch is registered[5]. This touch is not the most accurate, but comes at a much lower cost compared to the capacitive touch screen and was therefore chosen.

Keypad:
The final component of the user interface is the use of a keypad. This keypad is what allows the user to input their desired phone number, delay or radius of the geofence. We found that we needed a keypad with a backspace button so that any errors by the user can be fixed outside of the software. The keypad is attached using a USB port that can be easily integrated into the Raspberry Pi with little effort.

Chapter 6: Software Design

The functional controller for the software is the GUI. THis GUI acts as a finite state machine for different areas of code. Within each of these states are all of the necessary function declarations and requests for communicating over the serial port to the BG96, receiving the returned messages, and controlling the next available options for the user.

6.1 Finite State Machine Operation using GUI

Below are software flowcharts showing the main functionality of the GUI, the means of travelling between pages, and the software embedded within those pages. In Figure 6.1, the flowchart of the overall application (using a GUI library named TKinter) is shown [7]. Figures 6.2 through 6.10 then show the flowchart diagrams alongside a screen capture of each application page, or formally called a frame within TKinter. The symbol key for the software flowcharts in sections 6.1 and 6.2 is on the right. In section 6.3, there are flowcharts describing the embedded modules referenced in 6.2.8 and 6.2.9

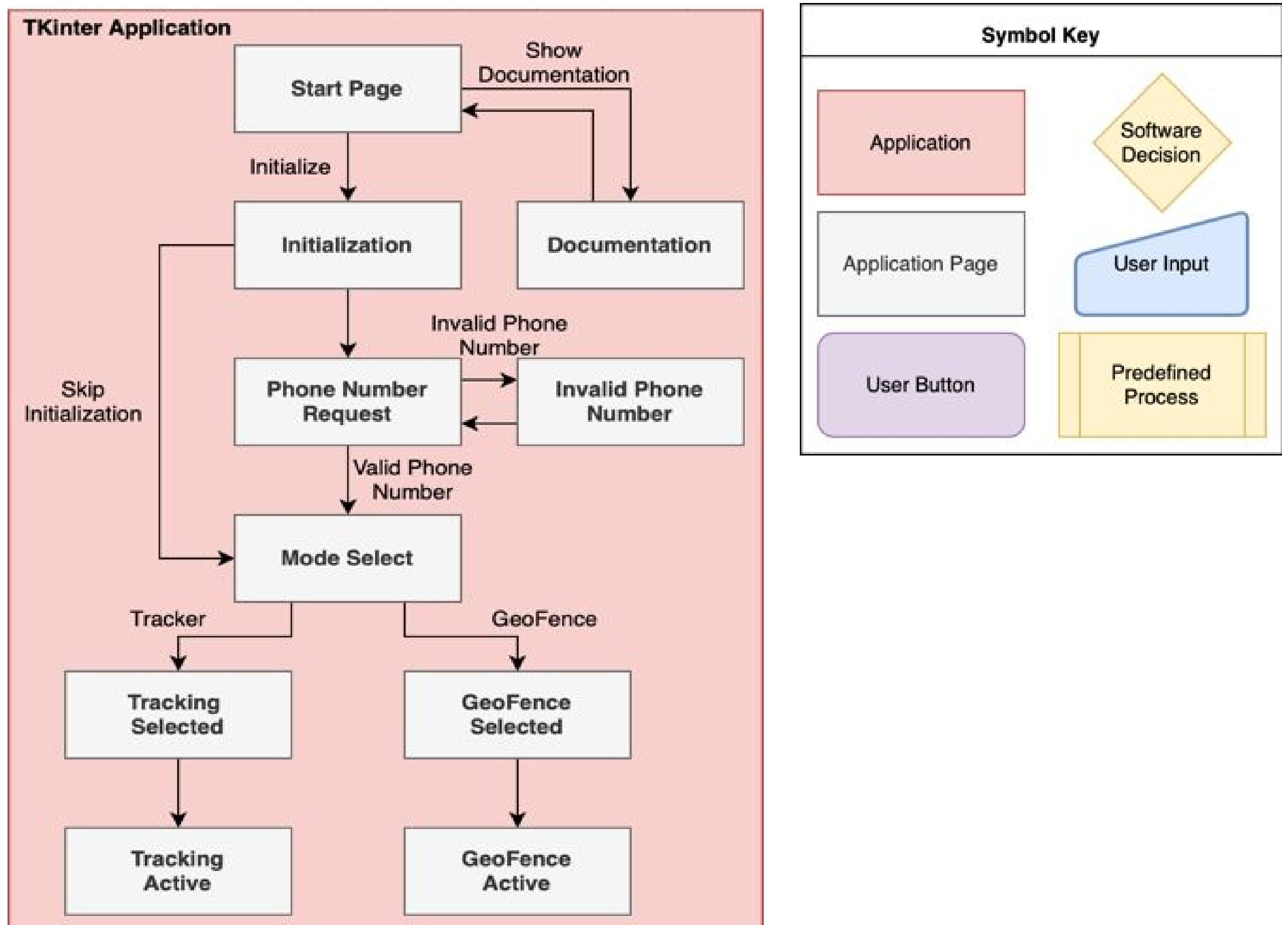
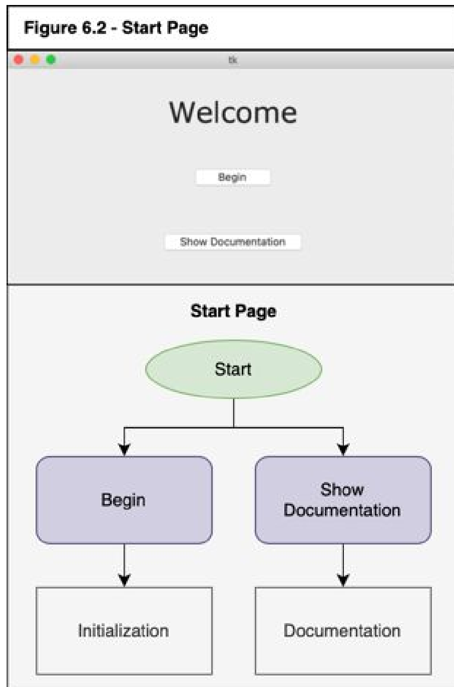


Figure 6.1 - Overall Application Flowchart

6.2 Application Pages

Sections 6.2.1 through 6.2.10 discuss in detail the software embedded within each frame of the application, as well as which frames can be accessed on each page.



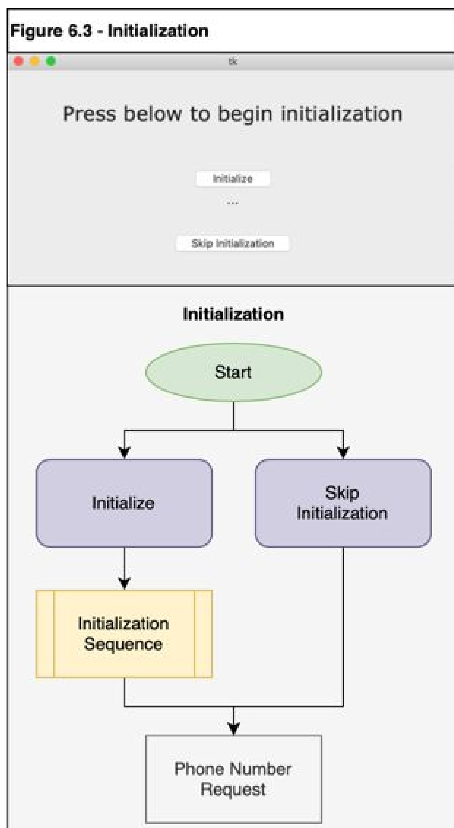
6.2.1 Start Page

Figure 6.2
General Description:

The **Start** page is the landing page upon running the application. Users will have two options navigating away from this page. The first is by pressing the Begin button, which brings the user to the **Initialization** page, outlined in 6.2.2. The second option is accessing the **Documentation** page, which can be selected using the Show Documentation button.

Embedded Modules: None

Global Functions Used: None



6.2.2 Initialization

Figure 6.3
General Description:

The **Initialization** page allows the user to either initialize the BG96 and serial communication port (embedded in the Initialization Sequence module) using the Initialize button, or skip initialization using the Skip Initialization button. Skipping initialization should only be used if the BG96 and serial port are already initialized. Both of these buttons lead to the **Phone Number Request** page outlined in 6.2.3.

Embedded Modules: Initialization Sequence

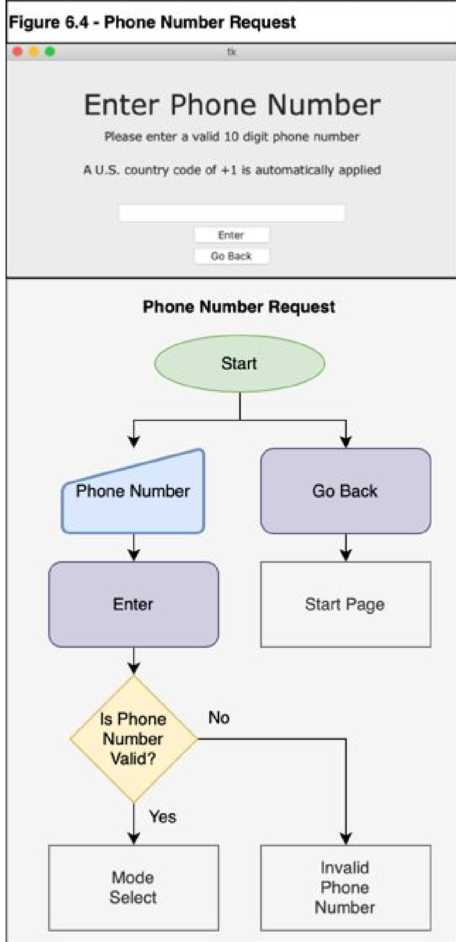
The Initialization Sequence module contains all of the necessary AT-Commands for properly initializing serial communication and the BG96 chipset. The module uses the global SendCmd() function in order to push these commands onto the BG96.

Global Functions Used: SendCmd()

6.2.3 Phone Number Request

Figure 6.4

General Description:



The **Phone Number Request** page is where the user will input the desired phone number for either Location Tracking or Geofence updates. The user will input a number into the textbox and press Enter to save the number. If the number is a valid ten-digit phone number, the number will be saved globally and they will be taken to the **Select Mode** page outlined in 6.2.5. If the number is not a valid ten-digit number, then they will be directed to the **Invalid Phone Number** page outlined in 6.2.4. The user also has the option to escape back to the **Start** page using the Go Back button.

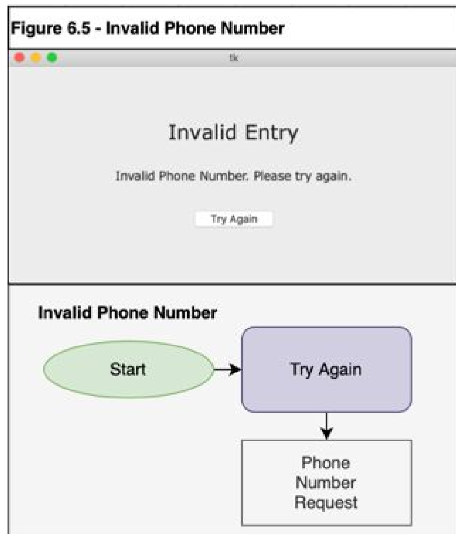
Embedded Modules: None

Global Functions Used: None

6.2.4 Invalid Phone Number

Figure 6.5

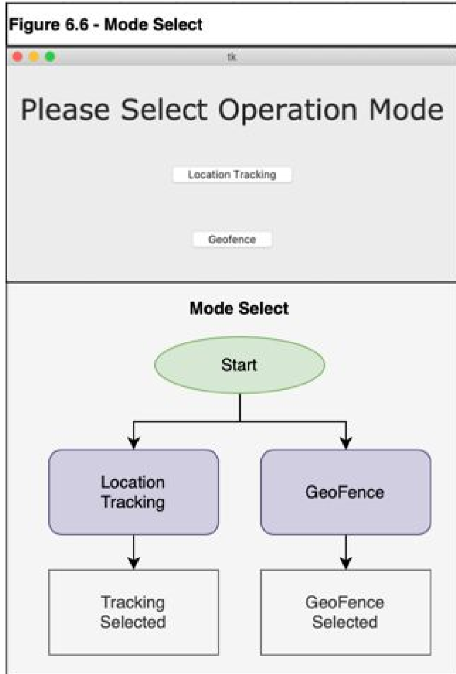
General Description:



The **Invalid Phone Number** page will only be reached if the user inputs an invalid (not ten-digit) number on the **Phone Number Request** page. The user will then be instructed to enter in a valid number, and press the Try Again button. This button leads them back to the **Phone Number Request** page to input a new number.

Embedded Modules: None

Global Functions Used: None



6.2.5 Mode Select

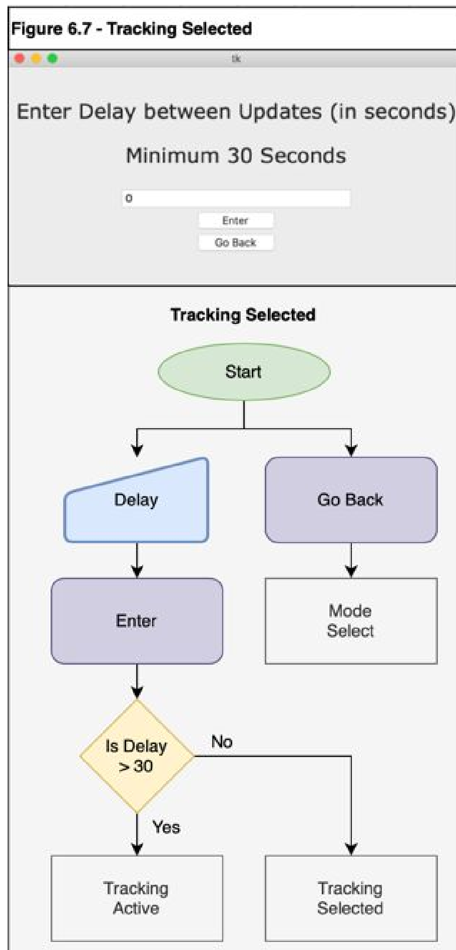
Figure 6.6

General Description:

The **Mode Select** page is where the user decides the true functionality of the IoT Asset Tracker. There are two modes of operation for the system: Location Tracking and Geofence, each with a button selection of the same name. By pressing the Location Tracking button, the user is brought to the **Texting Selected** page (6.2.6). Likewise, the Geofence button brings the user to the **Geofence Selected** page (6.2.7). The two modes of operation are discussed in more detail under sections 6.2.8 and 6.2.9.

Embedded Modules: None

Global Functions Used: None



6.2.6 Tracking Selected

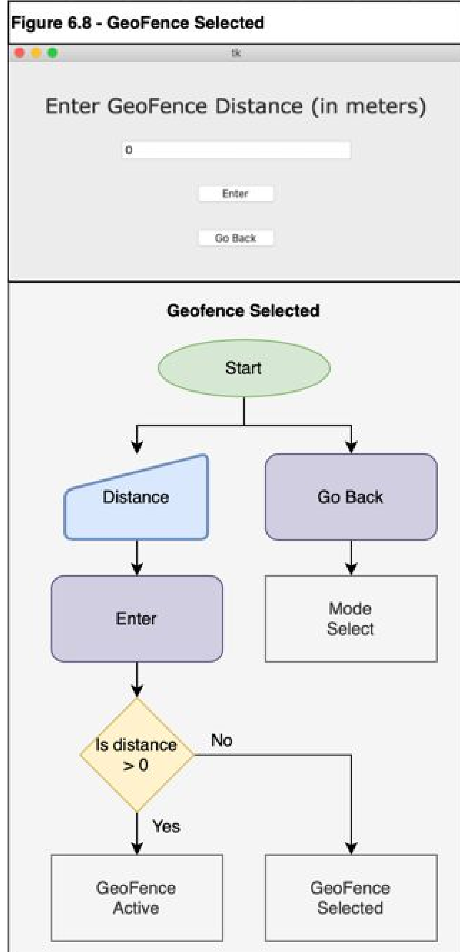
Figure 6.7

General Description:

The **Tracking Selected** page follows a selection of Location Tracking on the **Mode Select** page. This page contains the last user input before the system is ready to execute the desired function, which in this case, is location tracking. The user must input the time delay between location update texts. The minimum value for this is 30 (seconds). Upon entering a value into the text box, the user will hit the Enter button. This value will be sent to the check_delay() function to ensure it is greater or equal to 30. If the value meets this criteria, it is saved globally and the user is sent to the **Tracking Active** page outlined in 6.2.8. If the value is less than 30, they must enter a new value before continuing. They also have the choice to return to the **Mode Select** page using the Go Back button.

Embedded Modules: None

Global Functions Used: None



6.2.7 GeoFence Selected

Figure 6.8

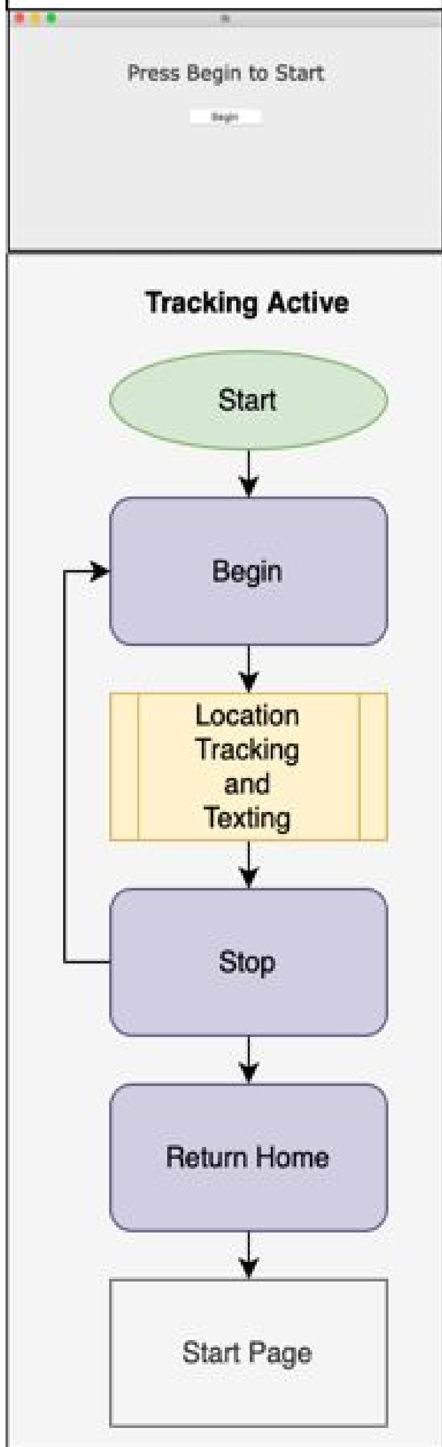
General Description:

The **GeoFence Selected** page follows a selection of Geofence on the **Mode Select** page. This page contains the last user input before the system is ready to execute the desired function, which in this case, is geofencing. The user must enter a geofence radius, specified in meters. The minimum value for this input is 1 (meter). Upon entering a value into the textbox, the value will be converted to an integer and sent to the set_geodistance() function, where it is checked to be greater than zero. If the value satisfies this criteria, it is saved globally and the user is brought to the **GeoFence Active** page (6.2.9)

Embedded Modules: None

Global Functions Used: None

Figure 6.9 - Tracking Active



6.2.8 Tracking Active

Figure 6.9

General Description:

The **Tracking Active** page allows the Location Tracking and Texting module to run in the background while sitting idle. The functions sit within a while loop that runs continuously as long as the global variable *textkill* is False. Upon the start of the program, this variable is initialized to False and can only be changed by calling the textingKillSwitch() function. This function is called by the Stop button present on the page. Once the Stop button is pressed, the while loop is exited and the Return Home button is added to the page. The user can then restart the loop by hitting the Begin button again, which reinitializes the *textkill* variable to False.

Embedded Modules: Location Tracking and Texting

As long as *textkill* is False, the function block represented by the Location Tracking and Texting process in Figure 6.9 will:

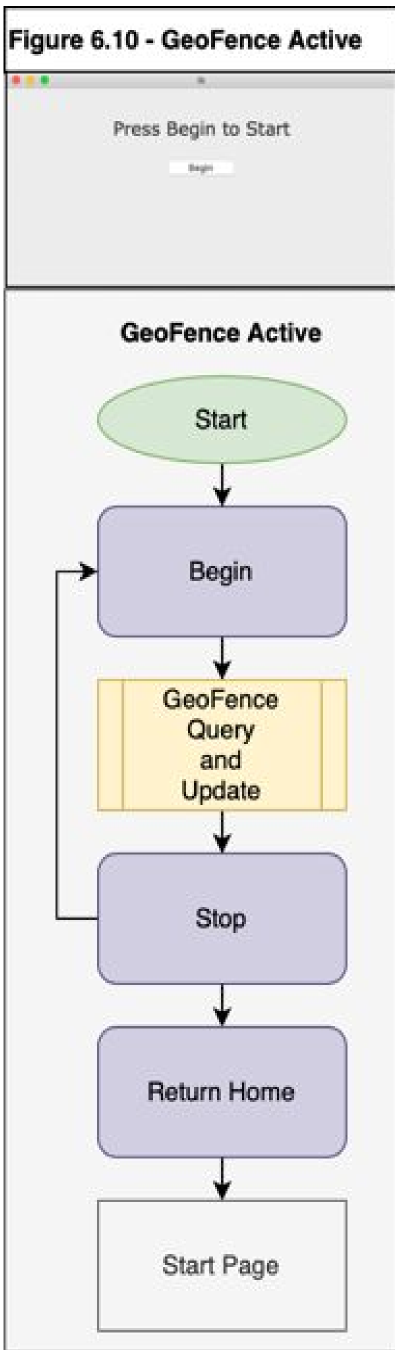
Request current GPS data from the BG96 through the serial port and convert the returned GPS data into latitude and longitude values. It will then send a text to the phone number saved in the **Phone Number Request** page that contains those latitude and longitude values. After this is sent, it will send another text to that same phone number that integrates those values with the Google Maps API, which, on a smartphone, will allow the message recipient to quickly open the current location on a map. Upon sending both messages, the program will delay for the amount of time specified on the **Texting Selected** page and then loop back.

Global Functions Used: GPSacquire(), SendTxt(), SendCmd()

6.2.9 GeoFence Active

Figure 6.10

General Description:



The **GeoFence Active** page allows the GeoFence Query and Update module to run in the background while sitting idle. The functions sit within a while loop that runs continuously as long as the global variable *geokill* is False. Upon the start of the program, this variable is initialized to False and can only be changed by calling the geoKillSwitch() function. This function is called by the Stop button present on the page. Once the Stop button is pressed, the while loop is exited and the Return Home button is added to the page. The user can then restart the loop by hitting the Begin button again, which reinitializes the *geokill* variable to False.

Embedded Modules: GeoFence Query and Update

On the first iteration of the module, the current position of the IoT Asset tracker will be requested and saved. The remaining GeoFence checks will then be made against this starting GPS location.

As long as *textkill* is False, the function block represented by the GeoFence Query and Update process in Figure 6.10 will:

Request status of the IoT device by measuring the distance between the current location and the location saved on the first iteration of the program. The distance will be checked against the radius set by the user on the **GeoFence Active** page and either: (a) the distance is less than the distance set by the user and nothing will happen or (b) the distance is greater and a text will be sent to the phone number provided by the user on the **Phone Number Request** page, alerting them that their tracker is outside of the radius.

Global Functions Used: queryGeo(), SendCmd(), SendTxt(), addGeoFence()

Note: There is hard-coded software delay within the GeoFence Query and Update module, giving a fifty second delay between geofence queries. An asset leaving the geofence radius will thus not give an alert for a maximum of fifty seconds.

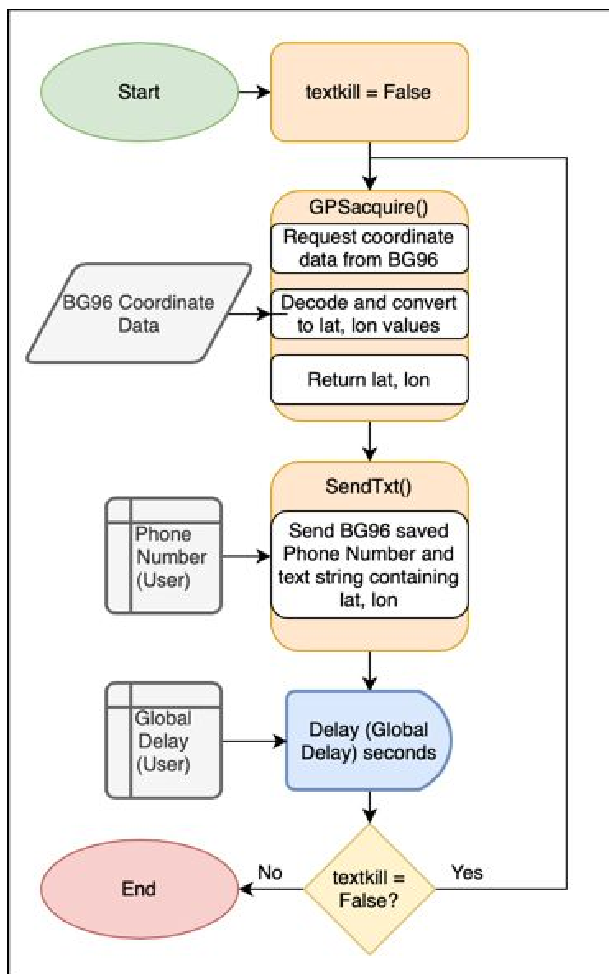
6.3 Embedded Modules

Note: The initialization module from 6.2.2 is described in section 5.1 in Table VI

In the pages outlined by sections 6.2.8 and 6.2.9, the majority of the functionality is actually embedded in code modules making use of both local and global variables and functions. These modules take information given in the overall application software and use it when communicating with the BG96 chipset. The first module, named Location Tracking and Texting, is outlined in 6.3.1, and the second module, GeoFence Query and Update, in section 6.3.2.

6.3.1 Location Tracking and Texting

The Location Tracking and Texting module handles the bulk of Location Tracker functionality on the IoT Asset Tracker. It is contained within a loop contingent on the global variable named *textkill* being Boolean False. As long as this is the case, the loop will run freely in the background of the application. The module consists of two global functions, namely, GPSacquire() and SendTxt().



With *textkill* set to True, the module will not run. If *textkill* is set to False, the module will first call the GPScquire() function. Note: *textkill* is set False by the Begin button on the **Texting Active** page.

GPSacquire() sends AT-commands to the BG96 requesting the coordinate data of the module. This data is returned via the serial port in a raw form. This data is then decoded and converted into standard latitude and longitude coordinates, and returned.
Example:

```
lat, lon = GPSaquire()
```

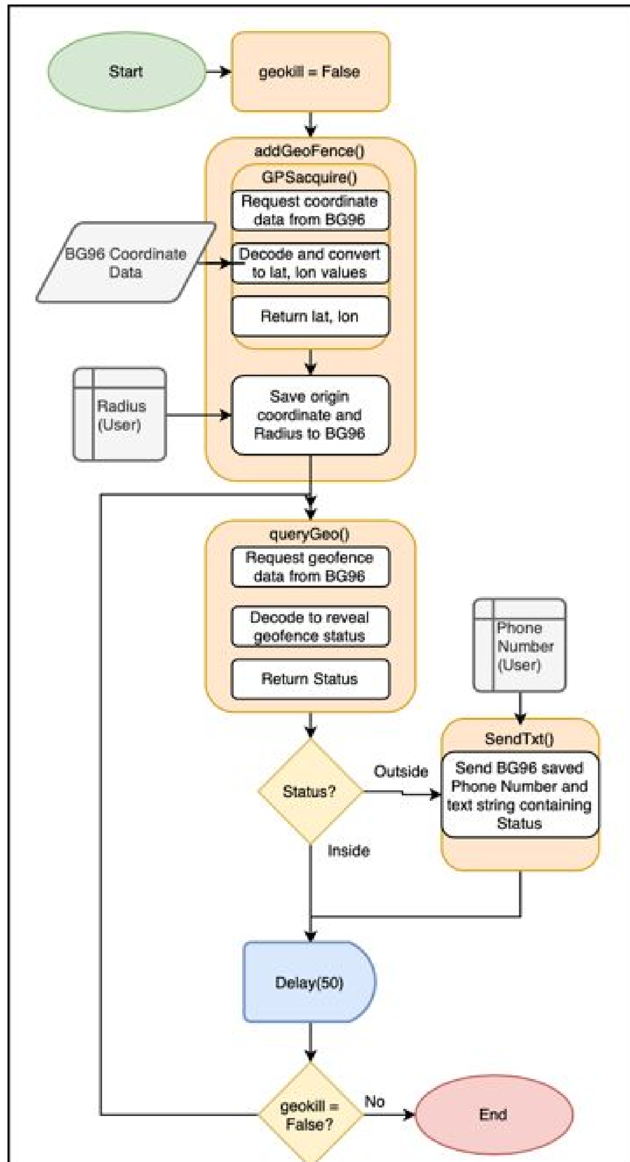
SendTxt() is fed the user-set phone number variable, as well as the latitude and longitude values returned in GPSacquire(). The function then sends the AT-commands to the BG96 requesting a text to that phone number containing the latitude and longitude values.
Example:

```
SendTxt(PhoneNumber, lat, lon)
```

This is followed by a software delay, specified by the user, and this iteration of the loop is completed. If *textkill* is True, the loop is terminated.

6.3.2 GeoFence Query and Update

The GeoFence Query and Update module handles the bulk of the geofence functionality on the IoT Asset Tracker. It is contained within a loop contingent on the global variable named *geokill* being Boolean False. As long as this is the case, the loop will run freely in the background of the application. The module consists of three global functions, namely, addGeoFence(), queryGeo(), and SendTxt().



With *geokill* set to True, the module will not run. If *geokill* is set to False, the module will call the addGeoFence() function. Note: *geokill* is set False by the Begin button on **GeoFence Active** page.

The **addGeoFence()** function is only called on the first iteration of the loop. This function first calls the **GPSacquire()** function discussed in 6.3.1, and returns the latitude and longitude values from the BG96. These values are then set as the origin coordinates for the geofence to compare new coordinates to, along with the user-defined Radius variable..

The true loop then begins, calling the **queryGeo()** function. The actual geofence comparison is handled mainly on board the BG96. **queryGeo()** requests a geofence status from the BG96 and is fed back a Status value, stored numerically. A '1' returned by the BG96 means that the IoT Asset tracker has not left the geofence radius. A '2' means that it has. This status is then converted to a "Inside" or "Outside" string by **queryGeo()** and returned.

If the status is "Inside", then the module will move to a software delay of 50 seconds and then rerun the loop (assuming *geokill* is still False). If the status is "Outside", then the **SendTxt()** function is called, sending an alert to the user-set phone number that the tracker has left the geofence radius. Again, there is then a software delay of 50 seconds and the loop starts over.

6.3 Globally Declared Functions

The application makes use of several globally declared functions that handle communication between the GUI and the BG96.

SendCmd(msg, rw)
Parameters:
msg: formatted AT Command
rw: ‘r’ or ‘w’ for read or write
Returns:
None if rw = ‘w’, Result if rw = ‘r’

```
def SendCmd(msg, rw):
   ser.write(msg)
   result = ser.read until('OK')
   print(msg)
   print(result)
   if rw == 'r':
       return(result)
```

SendCmd() is the primary avenue for communicating between the RaspberryPi and then BG96 chipset. Commands are written in the form of an AT command followed by a carriage return and new line character (\r \n).

SendTxt(number, lat, lon)
Parameters:
number: 10-digit phone number
lat: latitude coordinates
lon: longitude coordinates
Returns:
None

```
def SendTxt(number, lat, lon):

   SendCmd(b'ATE1\r\n', 'w')
   SendCmd(b'AT+CMGF=1\r\n', 'w')
   SendCmd(b'AT+CSMP=17,167,0,0\r\n', 'w')
   SendCmd(b'AT+CSCS="GSM"\r\n', 'w')
   SendCmd(str.encode(f'AT+CMGS="+1{number}"\r\n'), 'w')
   ser.write(str.encode(f'Current Location: \n {lat},  {lon}\x1A\r\n'))
   response = ser.read until('OK')
```

The SendTxt() function combines its input parameters with AT command strings and sends them to the BG96 using the SendCmd() function. It requires a phone number and latitude, longitude coordinates to properly execute.

addGeoFence(radius)

Parameters:

radius: Integer value greater than zero

Returns:

None

```
def addGeoFence(radius):
   lat, lon = GPSaquire()
   SendCmd(str.encode(f'AT+QCFGEXT="addgeo",0,3,0,{lat},{lon},{radius}\r\n'),
'w')
```

The addGeoFence() function first calls the GPSacquire() function to receive the current latitude and longitude values from the BG96. These values are then fed into an AT command string and sent via the SendCmd() function.

queryGeo()

Parameters:

None

Returns:

Status ("Inside", "Outside", or "State Unknown") depending on BG96 return message

```
def queryGeo():
   result = SendCmd(b'AT+QCFGEXT="querygeo",0\r\n', 'r')
   result = result.decode()
   new = result.replace('AT+QCFGEXT="querygeo",0\r\r\n+QCFGEXT:
"querygeo",0,', '')
   locate = new.replace('\r\n\r\nOK\r\n', '')
   print(locate)
   if locate == '1':
       return("Inside")
   elif locate == '2':
       return("Outside")
   else:
       return("State Unknown")
```

queryGeo() requests geofence status data from the BG96 via the SendCmd() function. This is handled in a single AT command in the first line of the function, followed by decoding the return message to the serial port. If the BG96 returns a '1' value, the module resides inside of the geofence defined by the initial GPS coordinate and the user-fed Radius value from the addGeoFence() function. Likewise, a '2' means that the module is outside of this radius. A '3' value is returned for an internal error on the BG96, where a calculation could not be properly made (such as a loss of signal).

GPSacquire()

Parameters:

None

Returns:

latitude, longitude

```
def GPSaquire():
   nmeaStr = SendCmd(b'AT+QGPSLOC=0\r\n', 'r')
   nmeaStr = nmeaStr.decode()
   new = nmeaStr.replace('AT+QGPSLOC=0\r\r\n+QGPSLOC: ', '')
   locArr = (new.replace('\r\n\r\nOK\r\n', '')).split(',')

   if 'N' in locArr[1]:
       lat dir = 1
   else:
       lat dir = -1
   if 'E' in locArr[2]:
       lon dir = 1
   else:
       lon dir = -1

   lat = float(locArr[1].replace('N', '').replace('S', ''))
   lon = float(locArr[2].replace('E', '').replace('W', ''))

   latitude = round(lat dir * (((lat % 100) / 60) + math.floor(lat / 100)), 6)
   longitude = round(lon dir * (((lon % 100) / 60) + math.floor(lon / 100)),
6)

   return latitude, longitude
```

GPSacquire() is the main method for receiving current GPS coordinates from the BG96. The initial AT command, sent by SendCmd(), queries the BG96 for a current location. The return message contains these coordinates embedded in other information, which is decoded within the function and the coordinates are extracted. These coordinates are still in a raw form so the values are also converted into a standard latitude and longitude coordinate before being returned.

Conclusions:

The IoT Asset Tracker was a very dynamic and challenging project. In the end we were able to design a system that met all the requirements needed to accomplish the functionality that we intended. Throughout our multiple design iterations and hardware configurations we learned to be thoughtful in our module selection and quick to adapt to new challenges. There was a lot to learn with this project such as Cellular/GNSS technology and Communication protocols. These were not easy hurdles to overcome but once we did we were able to connect all of our functional blocks and build our device. One of the largest takeaways from this project is the physical device. Even though this project is an IoT Asset Tracker, it contains all of the necessary parts to expand into a larger project if needed.

Hardware Challenges:
One of the main challenges we found with the hardware was the selection of a microprocessor. This issue arose when we began work with the BG96. We found that a MSP432 was unable to communicate with the device. By switching to a raspberry Pi we were able to solve this and add more functionality to the project. Furthermore, we had challenges with the touch screen. But once we found the correct initialization we were able to integrate the feature with ease.

Software Challenges:
There were a few major challenges with the software. The first was concerning the GUI design, and how we could built it to be both straight forward to the user and handle all of the necessary information for proper execution. We started with a very simple GUI program and ended up switching to a full scale GUI library to accomplish this. The other main struggle was when we decided to use a GUI, we then had to refit all of our functioning code within the GUI for execution. Prior to the GUI, the program ran with no user control (and only the user inputs for phone number, delay, and radius. There was no ability to skip around, change modes, or return back to the beginning of the program like there is now. This meant embedding blocks of code within pages of the GUI, and reworking much of it so that it would work in this compartmentalized manner.

Future Improvements:
One of the best features of this device is the use of the BG96. By using this module we were able to easily achieve the features we needed. However, we did not fully utilize this module. In future revisions of the project one can use the cellular chip to update locations on the web and use software to connect location pings so that you can see the route that the device has taken.

References:

---

[1] “BG96 AT-Commands Manual.” *Quectel BG96*, 2018, www.quectel.com/product/bg96.htm.

[2] “Quectel BG96 GNSS AT-Commands Manual.” *Quectel BG96*, www.quectel.com/product/bg96.htm.

[3] “HDMI Interface 5 Inch 800x480 TFT Display.” *Elecrow*, www.elecrow.com/wiki/index.php?title=HDMI_Interface_5_Inch_800x480_TFT_Display.

[4] “LTE BG96 Cat M1/NB1/EGPRS Module.” *Quectel LTE BG96 Cat.M1/NB1 & EGPRS Module*, www.quectel.com/product/bg96.htm.

[5] Kolokowsky, Steve. “Touchscreens 101: Understanding Touchscreen Technology and Design .” *Cypress Perform*, www.cypress.com/file/95156/download.

[6] “Raspberry Pi 3B+ User Guide.” *Projects.raspberrypi.org*, projects.raspberrypi.org/en/projects/raspberry-pi-setting-up.

[7]Roseman, Mark. “Modern Tk Best Practices.” *TkDocs Home*, tkdocs.com/.

[8]“Raspberry Pi Boot Modes.” *Raspberry Pi Boot Modes - Raspberry Pi Documentation*, www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/README.md.

[9]Liecheti, Chris. *PySerial*, pyserial.readthedocs.io/en/latest/pyserial.html.

[10] “Python 3.9.0 Documentation.” *3.9.0 Documentation*, docs.python.org/3/.

Appendix A: Analysis of Senior Project Design

Project Title: IoT Asset Tracker
Student's Name: Jonny Erickson & Matt Murray
Advisor's Name: Advisor's Initials: Joseph Callenes-Sloan
Date: 2/18/2020

• 1. Summary of Functional Requirements I
The IoT Asset Tracker has three main functions. The first is to provide accurate location tracking data based on user inputs. This is a goal of the project because as with modern day tracking solutions the accuracy is very high and the project needs to be competitive with these products already on the market. The second functional requirement is stability. The device needs to be stable enough to run for the entirety of its battery life without interruption, This is accomplished by using a more complex microcontroller(Raspberry Pi) running our project and by using connections that are semi permanent and resistant to shock and dislodgement. The final requirement is an easy to understand GUI to lead the user through initialization, feature selection and user inputs.

With these two goals we have compiled a list of required components. The most important is a high accuracy low power integrated Cellular GNSS chipset. This device can easily handle all Cellular related tasks such as initialization to the network, AT-Commands handling and GNSS control. The BG96 is also relatively low cost and low power and this assists in meeting the customer needs. Furthermore, the IoT Asset tracker is free of any leads or exposed wires. The device is wired up using DB9 cables and USB cables. These cables are less susceptible to coming undone and add to the stability of the device. The final large component that allows the device to meet its functional requirements is the 5in touch screen. This component meets multiple customer needs. The screen allows the user to easily understand what the device is doing, while also acting as a user defined state machine that keeps the device in bounded modes of operation. The screen also allows for a simplified use interface that eliminates the need for start/stop buttons.

• 2. Primary Constraints

The two primary constraints of this project are the formation of the GUI and the communication bus between the BG96 and the microcontroller. The GUI is a primary constraint because it acts as a state machine for the entire project. If the GUI hangs or is disabled the project will be unresponsive and need a full power cycle. To help mitigate this constraint we will have to keep the GUI as simple as possible while still being easy to set up and understand. This will assist in keeping the operational modes bounded and complete. The next constraint is the formation of the data link between the Raspberry Pi and the Microcontroller. We had found this portion of the design the hardest to overcome. The BG96 has very specific parameters for its UART connection and we needed to select a microcontroller to fit those requirements. By selecting a Raspberry Pi we were able to achieve a highly stable data link that allows us to easily write and read At-Commands and their responses.

• 3. Economic

• *What economic impacts result?*

The IoT asset tracker has a few economic impacts. The first beneficial impact is the mitigation of stolen assets. By decreasing the threat of theft the project will benefit the consumer. This will benefit the consumer and the insurance provider if the asset is insured. By mitigating the risks for both of these groups we are able to benefit both.

• *When and where do costs and benefits accrue throughout the project's lifecycle?*

The costs associated with the IoT Asset Tracker are primarily up front with the purchase of the device. After purchase and initialization the device should only cost the user electric costs of charging the device. The benefits that will occur throughout the lifecycle of the device are heavily dependent on the application of the device. If the device is used to track a child's car, then the benefits are peace of mind. But if the device is used to track an expensive asset that is susceptible to theft then the benefits arise if the device is stolen and able to be retrieved due to the device.

• *What inputs does the project require? How much does the project cost? Who pays?*
The project requirements and costs can be seen below in the cost evaluation.

Materials Costs

Manufacturer	Part Number	Description	Quantity	Cost per Unit	Total Cost
Quectel	BG96-TE-A+UMTS-LTE-EVB-KIT	LTE/GIS Evaluation Board	1	$ 116.00	$ 116.00
Element	Raspberry Pi 3 B+	Raspberry Pi Motherboard	1	$ 38.91	$ 38.91
Elecrow	RPA05010R	5 Inch RPI Touchscreen	1	$ 38.99	$ 38.99
Jelly Comb	B07CQGWNP2	Mini Number Pad - USB	1	$ 10.99	$ 10.99
RAVPower	RP-PB19	16750 mAh Power Bank	1	$ 23.99	$ 23.99
Benfei	000151BLACK	USB to Serial Cord (RS-232)	1	$ 8.59	$ 8.59
Optix	MC-17	20"x30"x0.093" Acrylic Sheet	1	$ 19.98	$ 19.98
GE	2708920	Clear Silicone Sealant Caulk	1	$ 6.57	$ 6.57
Total Materials Cost:					**$ 264.02**

Research and Development Costs

Job	Description	Hourly Rate	Number of Hours	Total Cost
Front-End GUI Design		$ 45.00	20	$ 900.00
Backend Software Design		$ 50.00	25	$ 1,250.00
Hardware Interfacing		$ 40.00	45	$ 1,800.00
Mechanical Design		$ 40.00	5	$ 200.00
Total Research and Development Cost:				**$ 4,150.00**

Manufacturing Labor Costs

Job	Description	Hourly Rate	Number of Hours	Total Cost
Enclosure Construction	Cutting and drilling of acrylic sheet, assembly of enclosure	$ 15.00	3	$ 45.00
Touch Screen Calibration	Downloading necessary drivers to Raspberry Pi and calibrating	$ 15.00	0.5	$ 7.50
Raspberry Pi Startup Initialization	Adding initialization code to the startup sequence on Rpi	$ 20.00	0.25	$ 5.00
Electrical Assembly	Wiring between RPi, BG96, Power Bank, Touch Screen, Keypad	$ 20.00	0.5	$ 10.00
Full Assembly	Attaching electrical assembly to the enclosure	$ 15.00	0.5	$ 7.50
Total Manufacturing Labor Cost:				**$ 75.00**

Total Costs

Type	Total Cost
Materials	$ 264.02
Manufacturing Labor	$ 75.00
Total Unit Cost:	**$ 339.02**
Research & Development	$ 4,150.00
Project Total Cost (Single Unit):	**$ 4,489.02**
Breakeven at 100 Units:	$ 380.52
Breakeven at 1,000 Units:	$ 343.17
Breakeven at 10,000 Units:	$ 339.44
Units sold to breakeven at $350 per Unit	**378 Units**

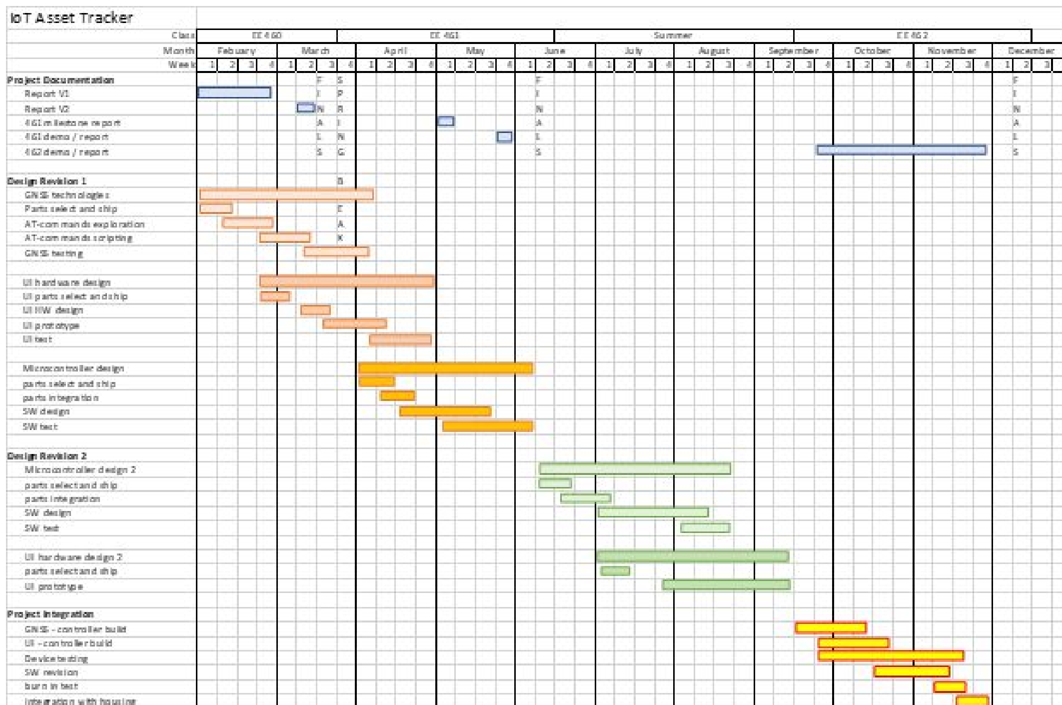
As for who pays, the developmental costs associated with the project are the responsibility of the developers. But the cost of the device lands on the consumer due to the fact that this

project is a consumer electrictronics product. Added maintenance costs are also minimal because the hardware configuration is static. Upon further development the cellular chip can be used for FOTA(Firmware Over The Air) where software revisions can be updated out of band.

- *How much does the project earn? Who profits?*

This project does not directly profit anybody but the seller directly. The seller profits by the margin of production. But the consumer benefits by the decrease in risk of a stolen asset. For example you have a .22% chance of having your car stolen[?]. Over lets say 60 years you can then say that you have a 13.5% chance of having your car stolen. By using the IoT Asset tracker you are able to highly increase the chance of recovering said vehicle and then profiting by saved time and opportunity cost.

- *Timing*



The total number of man hours to complete the project was estimated at roughly 100 man hours. The timeline for this project can be seen above in the given Gantt and Pert chart. Products emerged at roughly one half of the production cycle.

• 4. If manufactured on a commercial basis:

- Estimated number of devices sold per year: 1000
- Estimated manufacturing cost for each device: $339
- Estimated purchase price for each device: $400
- Estimated profit per year: $61,000
- Estimated cost for user to operate device, per unit time: 50/year

• 5. Environmental
The largest environmental impact will be the electrical component cost to the environment. The project requires a large lithium ion battery and complicated electrical materials to operate. Furthermore, the project requires a cellular network to operate and the environmental cost of setting up a cellular network is immense.

• 6. Manufacturability
The manufacturing cost on a large scale would require more engineering work to be done. A single PCB would be needed to keep manufacturing costs down. Doing so will require many more man hours to design and test, but will ultimately increase margins and decrease the size of the device. From then on manufacturing would become a supply chain problem of sourcing parts and hiring a contract manufacturer to assemble and load firmware.

• 7. Sustainability
The project is fairly sustainable due to the small size and complexity of the project. However the project can be more sustainable if recycled plastics were used for the housing, and if we choose to remove the battery or decrease it in size. Each of these changes may change the use case of the project. but overall the project is stable and even with a failure it would not cause a cascade in functionality.

• 8. Ethical
One ethical outcome of this project is the use of tracking someone without their consent. While this is something that is impossible for the producer to regulate, We can make the device larger and harder to hide. Also, since the device sends automated text messages over an interval, the device could be used as a harassment tool if the delay between reportings were to be too small. To mitigate this we set a minimum delay to 30s so that the recipient of the reportings is not over encumbered by the number of responses from the project

• 9. Health and Safety
This project is intended to increase the safety of the consumer. We are able to help protect loved ones from unintended circumstances. as mentioned before the usage of this project is heavily dependent on the use case of the device. If used as a vehicle tracker for a new driver, a parent can set a geofence with the device to monitor their child and keep them from putting themselves at risk.

• 10. Social and Political
This project is definitely best used by people who live in high risk areas where their assets may be at a higher risk. Such as in a city or out in farmland where theft can be high. Fortunately, people from these areas are at a higher economic risk than people in wealthy safe neighborhoods. Therefore, the project is able to protect people at higher risk if used correctly.

• **11. Development**
In this project we utilized two new technologies that were new to us. The first being the BG96, although we had used this device slightly before we had never dove into its larger functionality and use cases. We also started and built our GUI from scratch. We had no prior experience in this but it was gratifying to know that we had developed the full stack.

Appendix B: Project Code:

```
import tkinter as tk
from tkinter import ttk
import serial
import time
import math
import threading

# Initialize Global Values
geoKill = False
textKill = False

ser = serial.Serial('/dev/usbTTY0', baudrate=115200, timeout=1.0)

# Initialize fonts within TKinter
LARGEFONT = ("Verdana", 35)
MEDFONT = ("Verdana", 25)
SMALLFONT = ("Verdana", 15)
# Define GeoFence initialization
def addGeoFence(radius):
  lat, lon = GPSaquire()
  SendCmd(str.encode(f'AT+QCFGEXT="addgeo",0,3,0,{lat},{lon},{radius}\r\n'), 'w')
# Define GeoFence removal
def delGeoFence():
  SendCmd(b'AT+QCFGEXT="deletegeo",0\r\n', 'w')
# Define GeoFence Query loop
def queryGeo():
  ser.flushInput()
  result = SendCmd(b'AT+QCFGEXT="querygeo",0\r\n', 'r')
  print('RESULT:')
  print(result)
  result = result.decode()
  new = result.replace('AT+QCFGEXT="querygeo",0\r\r\n+QCFGEXT: "querygeo",0,', '')
  print("locate:")
  locate = new.replace('\r\n\r\nOK\r\n', '')
  print(locate)
  if locate == '1':
```

```
        return("Inside")
    elif locate == '2':
        return("Outside")
    else:
        return("State Unknown")
# Define GPS Acquisition
def GPSaquire():
    nmeaStr = SendCmd(b'AT+QGPSLOC=0\r\n', 'r')
    nmeaStr = nmeaStr.decode()
    new = nmeaStr.replace('AT+QGPSLOC=0\r\r\n+QGPSLOC: ', '')
    locArr = (new.replace('\r\n\r\nOK\r\n', '')).split(',')

    if 'N' in locArr[1]:
        lat_dir = 1
    else:
        lat_dir = -1
    if 'E' in locArr[2]:
        lon_dir = 1
    else:
        lon_dir = -1

    lat = float(locArr[1].replace('N', '').replace('S', ''))
    lon = float(locArr[2].replace('E', '').replace('W', ''))

    latitude = round(lat_dir * (((lat % 100) / 60) + math.floor(lat / 100)), 6)
    longitude = round(lon_dir * (((lon % 100) / 60) + math.floor(lon / 100)), 6)
    print(latitude, longitude)
    return latitude, longitude
# Serial Send Command through RasberryPi
def SendCmd(msg, rw):
    ser.write(msg)
    result = ser.read_until('OK')
    print(msg)
    print(result)
    if rw == 'r':
        return(result)
# Send Text
def SendTxt(number, lat, lon):

    SendCmd(b'ATE1\r\n', 'w')
    SendCmd(b'AT+CMGF=1\r\n', 'w')
    SendCmd(b'AT+CSMP=17,167,0,0\r\n', 'w')
    SendCmd(b'AT+CSCS="GSM"\r\n', 'w')
    SendCmd(str.encode(f'AT+CMGS="+1{number}"\r\n'), 'w')

    ser.write(str.encode(f'Current Location: \n {lat},  {lon}\x1A\r\n'))
    response = ser.read_until('OK')
```

```
    SendCmd(b'ATE1\r\n', 'w')
    SendCmd(b'AT+CMGF=1\r\n', 'w')
    SendCmd(b'AT+CSMP=17,167,0,0\r\n', 'w')
    SendCmd(b'AT+CSCS="GSM"\r\n', 'w')
    SendCmd(str.encode(f'AT+CMGS="+1{number}"\r\n'), 'w')

    googleMapsURL = f'https://google.com/maps/search/?api=1&query={lat},{lon}'
    ser.write(str.encode(f'{googleMapsURL} \x1A\r\n'))
    response = ser.read_until('OK')

def SendGeoTxt(number, status):
    SendCmd(b'ATE1\r\n', 'w')
    SendCmd(b'AT+CMGF=1\r\n', 'w')
    SendCmd(b'AT+CSMP=17,167,0,0\r\n', 'w')
    SendCmd(b'AT+CSCS="GSM"\r\n', 'w')
    SendCmd(str.encode(f'AT+CMGS="+1{number}"\r\n'), 'w')

    ser.write(str.encode(f'Current Status: {status} \x1A\r\n'))

def saveNumber(num):
    global PhoneNumber
    PhoneNumber = num

def saveDelay(delay):
    global Delay
    Delay = delay

def saveGeoDist(geo):
    global GeoDistance
    GeoDistance = geo

# Create application
class locationTracker(tk.Tk):

    # __init__ function for class tkinterApp
    def __init__(self, *args, **kwargs):
        # __init__ function for class Tk
        tk.Tk.__init__(self, *args, **kwargs)

        # creating a container
        container = tk.Frame(self)


        #self.attributes('-fullscreen', True)
        container.pack(side="top", fill="both", expand=True)

        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)
```

```
    # initializing frames to an empty array
    self.frames = {}

    # iterating through a tuple consisting
    # of the different page layouts
    for F in (StartPage, Initialization, PhoneNumberRequest,
            ModeSelect, InvalidNumber, SelectedText, Documentation,
            TextingActive, SelectedGeo, GeoActive):
        frame = F(container, self)

        # initializing frame of that object from
        # startpage, page1, page2 respectively with
        # for loop
        self.frames[F] = frame

        frame.grid(row=0, column=0, sticky="nsew")

    self.show_frame(StartPage)

    # to display the current frame passed as

# parameter
def show_frame(self, cont):
    frame = self.frames[cont]
    frame.tkraise()

# first window frame startpage

def check_num(self, num):
    if len(str(num)) == 10:
        saveNumber(num)
        print(num)
        return self.show_frame(ModeSelect)
    else:
        return self.show_frame(InvalidNumber)

def check_delay(self, delay):
    if delay >= 30:
        saveDelay(delay)
        print(delay)
        return self.show_frame(TextingActive)
    else:
        return self.show_frame(SelectedText)

def set_geodistance(self, num):
    if num > 0:
        saveGeoDist(num)
```

```
            return self.show_frame(GeoActive)
        else:
            return self.show_frame(SelectedGeo)

# Define opening page
class StartPage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        welcomeLabel = ttk.Label(self, text="Welcome", font=LARGEFONT)

        welcomeLabel.place(relx=0.5, rely=0.4, anchor=tk.CENTER)

        enterPhnButton = ttk.Button(self, text="Begin",
                    command=lambda: controller.show_frame(Initialization))

        enterPhnButton.place(relx=0.5, rely=0.45, anchor=tk.CENTER)

        docButton = ttk.Button(self, text="Show Documentation",
                    command=lambda: controller.show_frame(Documentation))
        docButton.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
# Define initialization page
class Initialization(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        self.welcomeLabel = ttk.Label(self, text="Press below to begin initialization", font=MEDFONT)

        self.welcomeLabel.place(relx=0.5, rely=0.4, anchor=tk.CENTER)

        self.enterPhnButton = ttk.Button(self, text="Initialize",
                    command= self.InitializationSequence)

        self.enterPhnButton.place(relx=0.5, rely=0.45, anchor=tk.CENTER)

        self.currentStatusLabel = ttk.Label(self, text='...', font=SMALLFONT)
        self.currentStatusLabel.place(relx=0.5, rely=0.55, anchor=tk.CENTER)


        self.continueButton = ttk.Button(self, text="Continue",
                        command=lambda: controller.show_frame(PhoneNumberRequest))

        self.skipinitialization = ttk.Button(self, text = "Skip Initialization",
                            command= self.areYouSure)
        self.skipinitialization.place(relx = 0.5, rely = 0.6, anchor = tk.CENTER)
        self.areyousurebutton = ttk.Button(self, text="Are you Sure?",
                            command=lambda: controller.show_frame(PhoneNumberRequest))
    def areYouSure(self):
```

```
        self.skipinitialization.destroy()
        self.areyousurebutton.place(relx = 0.5, rely = 0.6, anchor = tk.CENTER)
        return


    def InitializationSequence(self):
        InitATCommands = [[b'AT\r\n', 'w'], [b'ATV1\r\n', 'w'], [b'ATE1\r\n', 'w'],
                          [b'AT+CMEE=2\r\n', 'w'], [b'AT+IPR?\r\n', 'w'],
                          [b'ATI\r\n', 'w'], [b'AT+GSN\r\n', 'w'], [b'ATI\r\n', 'w'],
                          [b'AT+CPIN?\r\n', 'w'], [b'AT+CIMI\r\n', 'w'], [b'AT+CSQ\r\n', 'w'],
                          [b'AT+CREG?\r\n', 'w'], [b'AT+CGREG?\r\n', 'w'], [b'AT+COPS?\r\n', 'w'],
                          [b'AT+QGPSCFG="outport","none"\r\n', 'w'], [b'AT+QGPS?\r\n', 'w']]

        self.skipinitialization.destroy()
        self.progressBar = ttk.Progressbar(self, orient=tk.HORIZONTAL,
                                           length=112, mode='determinate')
        self.progressBar.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
        self.progressBar['value'] = 0
        self.progressBar.update()
        for cmd in InitATCommands:
            SendCmd(cmd[0], cmd[1])

            self.progressBar['value'] += 7
            self.progressBar.update()
            self.currentStatusLabel.destroy()
            self.currentStatusLabel = ttk.Label(self, text=str(cmd[0]), font=SMALLFONT)
            self.currentStatusLabel.place(relx=0.5, rely=0.55, anchor=tk.CENTER)


        self.welcomeLabel = ttk.Label(self, text="Do you wish to Re-initialize or continue?", font=MEDFONT)

        self.welcomeLabel.place(relx=0.5, rely=0.4, anchor=tk.CENTER)

        self.enterPhnButton.destroy()
        self.enterPhnButton = ttk.Button(self, text="Re-Initialize",
                                         command=self.InitializationSequence)

        self.enterPhnButton.place(relx=0.5, rely=0.45, anchor=tk.CENTER)
        self.continueButton.place(relx=0.5, rely=0.6, anchor=tk.CENTER)
# Define phone number request page
class PhoneNumberRequest(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label1 = ttk.Label(self, text="Enter Phone Number", font=LARGEFONT)
        label1.place(relx=0.5, rely=0.4, anchor=tk.CENTER)
        label2 = ttk.Label(self,
                          text="Please enter a valid 10 digit phone number",
```

```
                 font=SMALLFONT)
        label2.place(relx=0.5, rely=0.45, anchor=tk.CENTER)

        label3 = ttk.Label(self,
                 text="A U.S. country code of +1 is automatically applied",
                 font=SMALLFONT)
        label3.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

        phoneNumber = tk.StringVar()
        numEntered = ttk.Entry(self, width = 30, textvariable = phoneNumber)
        numEntered.place(relx=0.5, rely=0.55, anchor=tk.CENTER)

        button1 = ttk.Button(self, text="Enter",
                  command= lambda: controller.check_num(phoneNumber.get()))

        button1.place(relx=0.5, rely=0.6, anchor=tk.CENTER)

        button2 = ttk.Button(self, text="Go Back",
                  command=lambda: controller.show_frame(StartPage))

        button2.place(relx=0.5, rely=0.65, anchor=tk.CENTER)


# Define operation mode select page
class ModeSelect(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = ttk.Label(self, text="Please Select Operation Mode", font=LARGEFONT)
        label.place(relx=0.5, rely=0.4, anchor=tk.CENTER)

        button1 = ttk.Button(self, text="Location Tracking",
                  command=lambda: controller.show_frame(SelectedText))

        button1.place(relx=0.5, rely=0.45, anchor=tk.CENTER)

        button2 = ttk.Button(self, text="Geofence",
                  command=lambda: controller.show_frame(SelectedGeo))

        button2.place(relx=0.5, rely=0.5, anchor=tk.CENTER)


# Define invalid number entry page
class InvalidNumber(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = ttk.Label(self, text="Invalid Entry", font=LARGEFONT)
        label.place(relx=0.5, rely=0.4, anchor=tk.CENTER)
```

```
        label1 = ttk.Label(self,
                    text = "Invalid Phone Number. Please try again.",
                    font=MEDFONT
                    )
        label1.place(relx=0.5, rely=0.45, anchor=tk.CENTER)
        # button to show frame 2 with text
        # layout2
        button1 = ttk.Button(self, text="Try Again",
                     command=lambda: controller.show_frame(PhoneNumberRequest))

        # putting the button in its place by
        # using grid
        button1.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
# Define documentation page
class SelectedText(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label1 = ttk.Label(self, text="Enter Delay between Updates (in seconds)", font=LARGEFONT)
        label1.place(relx=0.5, rely=0.4, anchor=tk.CENTER)
        label1_1 = ttk.Label(self, text="Minimum 30 Seconds", font=MEDFONT)
        label1_1.place(relx=0.5, rely=0.45, anchor=tk.CENTER)

        timeDelay = tk.IntVar()

        delayEntered = ttk.Entry(self, width = 30, textvariable = timeDelay)
        delayEntered.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

        button1 = ttk.Button(self, text="Enter",
                     command= lambda: controller.check_delay(timeDelay.get()))

        button1.place(relx=0.5, rely=0.55, anchor=tk.CENTER)

        button2 = ttk.Button(self, text="Go Back",
                     command=lambda: controller.show_frame(StartPage))

        button2.place(relx=0.5, rely=0.6, anchor=tk.CENTER)


class TextingActive(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label1 = ttk.Label(self, text="Press Begin to start", font=LARGEFONT)
        label1.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

        button = ttk.Button(self, text="Begin",
                    command= self.beginText)
```

```
        button.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

        self.button1 = ttk.Button(self, text="Stop",
                            command= self.textKillSwitch)


        self.button2 = ttk.Button(self, text="Return to Mode Selection",
                               command=lambda: controller.show_frame(ModeSelect))



    def beginText(self):
        global textKill
        textKill = False

        self.button1.place(relx=0.5, rely=0.25, anchor=tk.CENTER)
        return self.textingExecution()

    def textingExecution(self):
        if not textKill:
            latitude, longitude = GPSaquire()
            SendTxt(PhoneNumber, latitude, longitude)
            time.sleep(Delay)
            TextingActive.after(self, 25, self.textingExecution)

    def textKillSwitch(self):
        global textKill
        textKill = True

        self.button2.place(relx=0.5, rely=0.5, anchor=tk.CENTER)


class SelectedGeo(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label1 = ttk.Label(self, text="Enter GeoFence Distance (in meters)", font=LARGEFONT)
        label1.place(relx=0.5, rely=0.4, anchor=tk.CENTER)

        geoDist = tk.IntVar()
        geoDistEntered = ttk.Entry(self, width = 30, textvariable = geoDist)
        geoDistEntered.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

        button1 = ttk.Button(self, text="Enter",
                            command= lambda: controller.set_geodistance(geoDist.get()))

        button1.place(relx=0.5, rely=0.55, anchor=tk.CENTER)
```

```
        button2 = ttk.Button(self, text="Go Back",
                             command=lambda: controller.show_frame(StartPage))

        button2.place(relx=0.5, rely=0.6, anchor=tk.CENTER)

class GeoActive(tk.Frame):


    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label1 = ttk.Label(self, text="Press 'Begin' to Start", font=LARGEFONT)
        label1.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

        button = ttk.Button(self, text="Begin",
                            command =  self.beginGeo)
        button.place(relx = 0.5, rely = 0.2, anchor=tk.CENTER)
        self.button1 = ttk.Button(self, text="Stop",
                             command=  self.geoKillSwitch)

        self.button2 = ttk.Button(self, text="Return to Mode Selection",
                                  command=lambda: controller.show_frame(ModeSelect))



    def beginGeo(self):
        addGeoFence(GeoDistance)
        global geoKill
        geoKill = False
        self.button1.place(relx=0.5, rely=0.25, anchor=tk.CENTER)
        return self.geofenceExecution()

    def geofenceExecution(self):
        if not geoKill:
            state = queryGeo()
            statuslabel = ttk.Label(self, text="Status: {}".format(state), font=LARGEFONT)
            statuslabel.place(relx=0.5, rely=0.4, anchor=tk.CENTER)
            if state == 'Outside':
                SendGeoTxt(PhoneNumber, state)
            time.sleep(5)

            GeoActive.after(self, 25, self.geofenceExecution)

    def geoKillSwitch(self):
        global geoKill
        geoKill = True
        self.button2.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

class Documentation(tk.Frame):
```

```
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = ttk.Label(self, text="Documentation", font=LARGEFONT)
        label.place(relx=0.5, rely=0.1, anchor=tk.CENTER)

        label = ttk.Label(self,
                          text = "All the shit about our project",
                          font=SMALLFONT
                          )
        label.place(relx=0.5, rely=0.2, anchor=tk.CENTER)
        # button to show frame 2 with text
        # layout2
        button1 = ttk.Button(self, text="Return",
                             command=lambda: controller.show_frame(StartPage))

        # putting the button in its place by
        # using grid
        button1.place(relx=0.5, rely=0.8, anchor=tk.CENTER)


app = locationTracker()
app.mainloop()
```