Wirelessly Integrating Topographic Teleoperation Instrument (WITTI)

Brian Smith (bdsmith1@email.arizona.edu)
Brianna Heersink (bmheersink@gmail.com)
Alex Warren (amwarren@email.arizona.edu)

03/26/2014

Team Witty

ECE 573 - Software Engineering Concepts

Professor Sprinkle

The University of Arizona

Tucson, Arizona

Executive Summary

The Wirelessly Integrating Topographic Teleoperation Instrument presented in this report is an application used to visualize LiDAR data on a mobile device. This visualization will allow the user to see the environment that an autonomous vehicle is sensing and using to make decisions. The application has the functionality to visualize prerecorded data that is saved on the mobile device or prerecorded data saved on a computer. As one option, the application may be able to collect and display data live from the Velodyne LiDAR via the computer and server. The application may also provide the functionality for the user to select a vehicle trajectory by touching the visualized LiDAR data on the screen.

Contents

List of Figures

List of Tables

1 Project Overview

The Light Detection and Ranging (LiDAR) visualization application is designed to display information collected from the LiDAR on a mobile device. The app may also allow users to draw a path on top of the visualized point cloud and determine if the path is reachable.

LiDAR devices use an array of lasers to collect data that give the distance and height values of an array of points stemming out of the LiDAR that can be used to determine the physical characteristics of the surrounding landscape. Typically, LiDAR is used to help autonomous vehicles get information on their surroundings.

This app will give researchers the ability to see a subset of what the vehicle sees by playing back previously recorded data and optionally displaying live data from the LiDAR. This could be useful in debugging unexpected vehicle behavior or determining the limitations of the LiDAR when driving. This app will also help researchers visualize how and why the vehicle makes certain decisions when determining if a chosen path is reachable.

This report provides a description of the implementation details and analysis for our application. The analysis section includes a description of how the app will function, class diagrams, a description of the supported use cases, several example sequence diagrams, and a discussion of the important algorithms the application will utilize. Also included in the analysis are sample screen shots of the various user interfaces that will be part of the application. After the application analysis section there is a description of how the application will be completed, including a timeline and a separation of duties. The report concludes that the application will be completed on time and will accomplish the application goals.

Part I
Analysis and Models

2 Requirements

2.1 'B' Requirements

B.1 Display Data: The phone application software shall load and display Velodyne LiDAR data on the phone.

Prerequisite(s): None

Difficulty: 4

B.2 Manual Data Refresh: The phone application software shall be capable of refreshing the displayed Velodyne LiDAR data manually based on user input.

Prerequisite(s): B.1

Difficulty: 2

B.3 Server Download: The phone application software shall be capable of downloading Velodyne LiDAR data in the form of an XYZ-point binary file from a remote server.

Prerequisite(s): None

Difficulty: 3

B.4 Server Availability: The host computer software shall be capable of uploading or directly serving Velodyne LiDAR data in the form of an XYZ-point binary file.

Prerequisite(s): None

Difficulty: 3

B.5 Demo Playback: The phone application shall be able to use XYZ-point text files stored locally for Manual Data Refresh mode.

Prerequisite(s): B.2

Difficulty: 2

2.2 'A' Requirements

A.1 Automatic Data Refresh: The phone application software shall be capable of refreshing the displayed Velodyne LiDAR data automatically through a set refresh rate.

Prerequisite(s): B.1

Difficulty: 4

A.2 Draw Trajectory: The phone application software shall read a trajectory selected by the user through touch events on the phone.

Prerequisite(s): B.1

Difficulty: 4

A.3 Trajectory Clearance: The phone application software shall indicate if the user selected trajectories would be too close to obstacles.

Prerequisite(s): A.2

Difficulty: 3

A.4 Trajectory Drivability: The phone application software shall reject trajectories that are not possible to follow due to vehicle limitations.

Prerequisite(s): A.2

Difficulty: 3

A.5 Perspective Change: The phone application software shall be capable of changing the perspective of the displayed data on the phone.

Prerequisite(s): B.1

Difficulty: 3

A.6 Dynamic LiDAR Conversion: The host computer shall be capable of processing and converting the PCAP files to phone readable data as they are received within a bounded delay that will be established.

Prerequisite(s): None

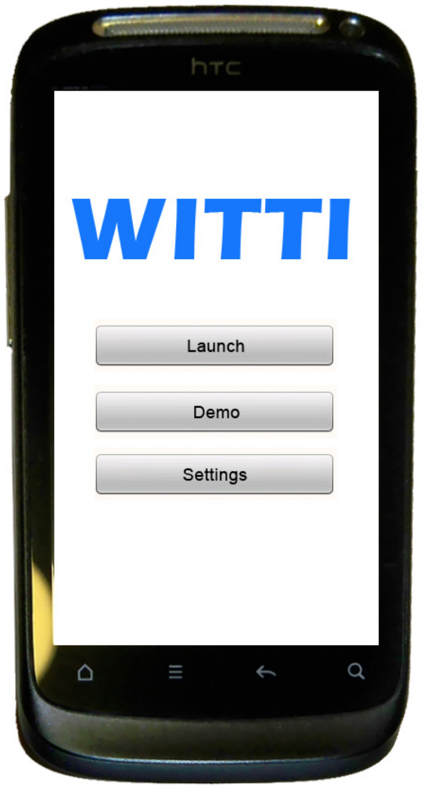
Difficulty: 4

A.7 JAUS Compatible: The host computer software shall be capable of sending JAUS messages containing converted LiDAR data.

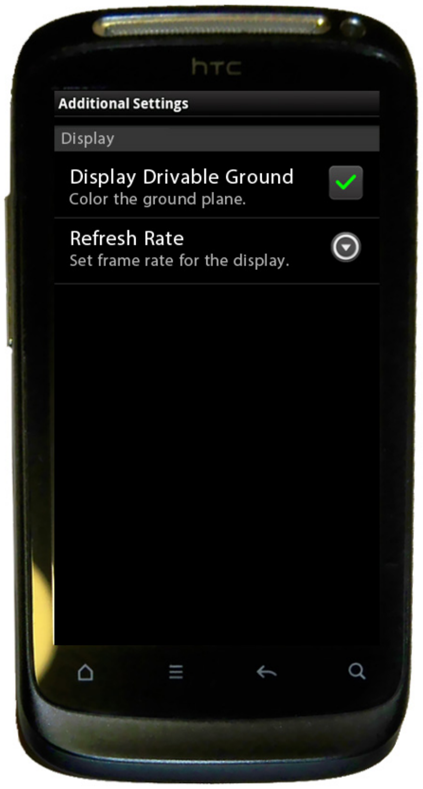
Prerequisite(s): None

Difficulty: 4

3 Application Analysis



(a) Home Screen



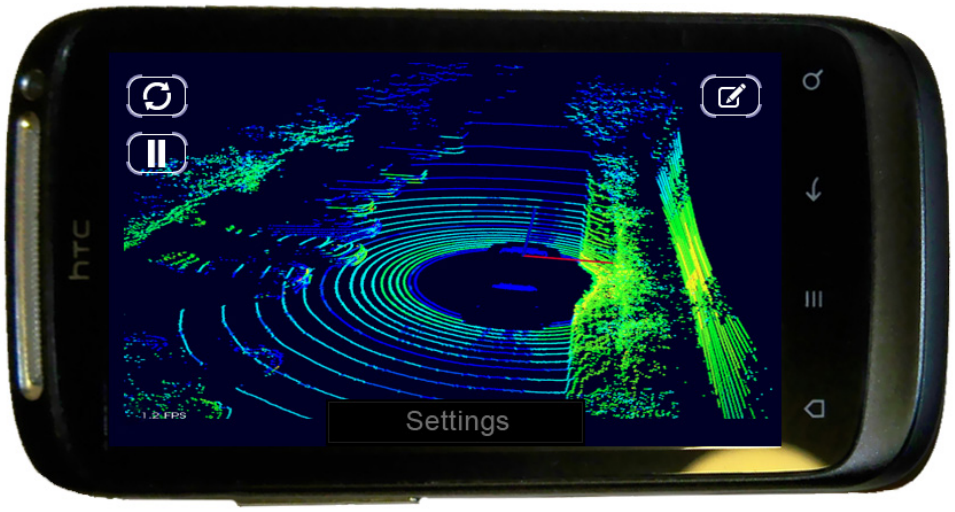
Additional Settings

Display

Display Drivable Ground ✓
Color the ground plane.

Refresh Rate
Set frame rate for the display.

(b) Settings Screen



(c) Display Screen

Figure 1: Mock Screen Shots

3.1 Overview

When the app initializes, the user is brought to the title screen. Here, the user is presented with three buttons which will choose the next action: Demo, Launch, and Settings. The user can navigate to the settings menu by clicking on the Settings button. The user can navigate to the display interface by clicking on the Demo button or the Launch button.

The Demo button is used initialize Demo mode, which will read point cloud data directly from a file stored on the device. The Launch button is used to initialize wireless mode, where the phone will connect with a host computer in order to receive point cloud data.

The user is taken to the display interface where point cloud data immediately starts playing at the set refresh rate and from the set location. While the user is viewing the display interface, there will be several on-screen options. If the user is in tap to refresh mode, a draw path option and a refresh option will be available. If the user is streaming video, the only option will be to pause the playback. After a successful pause, the user may have the option to draw a path. After a valid path is inputted, the app may immediately check for reachability and inform the user of the outcome. The user is then taken back to the display interface in the paused state.

The user is able to navigate back to the title screen by clicking the back button. The user can quit the application by clicking the back button at the title screen. The home button will pause the app and return control to the device. These actions can be reviewed in the State Diagram below.

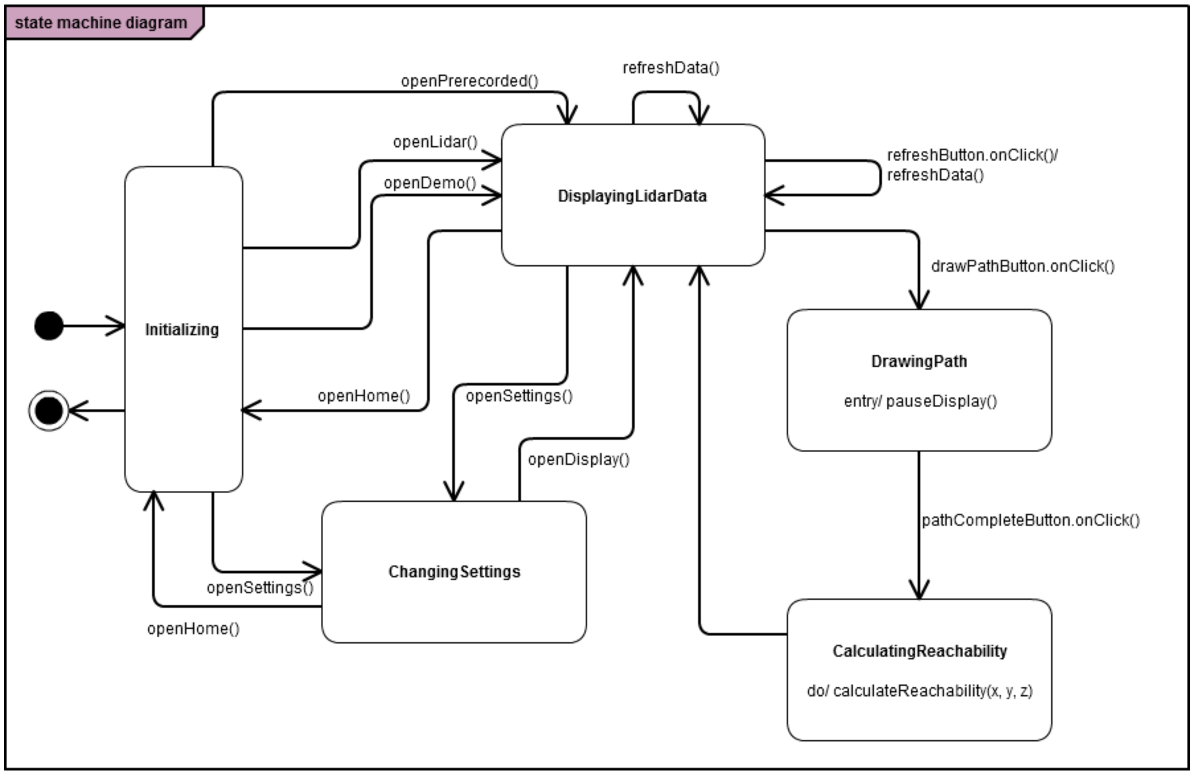


Figure 2: Phone Application State Diagram

3.2 Use Cases

The phone application has several use cases, as can be seen in the figure below. The actor for all use cases is the user of the phone application who wishes to view the LiDAR data and optionally draw vehicle trajectories on a smartphone.

3.2.1 Determine reachability of a drawn path.

Summary: The app determines if the path drawn by the user on the screen is reachable, i.e. no obstacles in the path.

Actors: User

Preconditions: The app is displaying point cloud data.

Description: The user first clicks the pause button if the current mode is set to stream data. Then the user draws a path on the screen of the app. The app first checks if the path is valid, i.e. does not go out of bounds of the image. If the path is valid, the app will calculate reachability of the selected path. The app will then alert the user whether the selected path is reachable. At this point, the app will exit the path drawing state and return to the image viewing with the image paused.

Exceptions: The following exceptions are expected:

Invalid Path The user draws a path outside the boundary of the image. The app will give the use a warning popup and clear the screen.

Postconditions: The app returns to the image view screen with the current view paused.

3.2.2 Watching live LiDAR data

Summary: The user is streaming live LiDAR data.

Actors: User

Preconditions: The server is accessible to both the host computer and the mobile device. The app is open to the title screen, HomeActivity.

Description: The user selects Launch Mode. The user selects either a constant refresh rate or tapping the screen to get the current LiDAR view. Once the user acknowledges all settings, the device will open to the DisplayActivity screen and begin streaming live LiDAR data.

Exceptions: The following exceptions are expected:

Connection Signal Lost The connection between the host computer or mobile device and the server is lost. The app will do some memory clean up and then return to the Introduction screen.

Postconditions: The app is open to the DisplayActivity.

3.2.3 Demo mode viewing

Summary: The user is viewing prerecorded LiDAR data stored on the mobile device.

Actors: User

Preconditions: The app is open to the title screen.

Description: The user selects Demo Mode. The user selects either a constant refresh rate or tapping the screen to update the LiDAR view. Once the user acknowledges all settings, the app opens to the DisplayActivity screen and begins displaying the prerecorded LiDAR data.

Exceptions: None.

Postconditions: The app is open to the DisplayActivity.

4 Domain Analysis

4.1 Overview

The application will involve many processes transparent to the user. In addition to the software on the mobile device, hardware and software outside of the mobile device are involved in providing the full functionality on the Android app.

4.2 Refresh

The main functionality of the app is to provide the user a visualization of the data previously collected and saved from the Velodyne LiDAR. The objects and methods involved in refreshing the image displayed on the mobile device can be seen in the sequence model below. The preconditions to this sequence model are as follows:

1 The app is in the Launch mode rather than the Demo mode.

2 The app settings are configured to have a refresh rate of 0, which requires the user to manually select when the screen should be updated.

3 The app settings are configured to have the LiDAR data downloaded from the server.

4 The LiDAR data is accessible on the server.

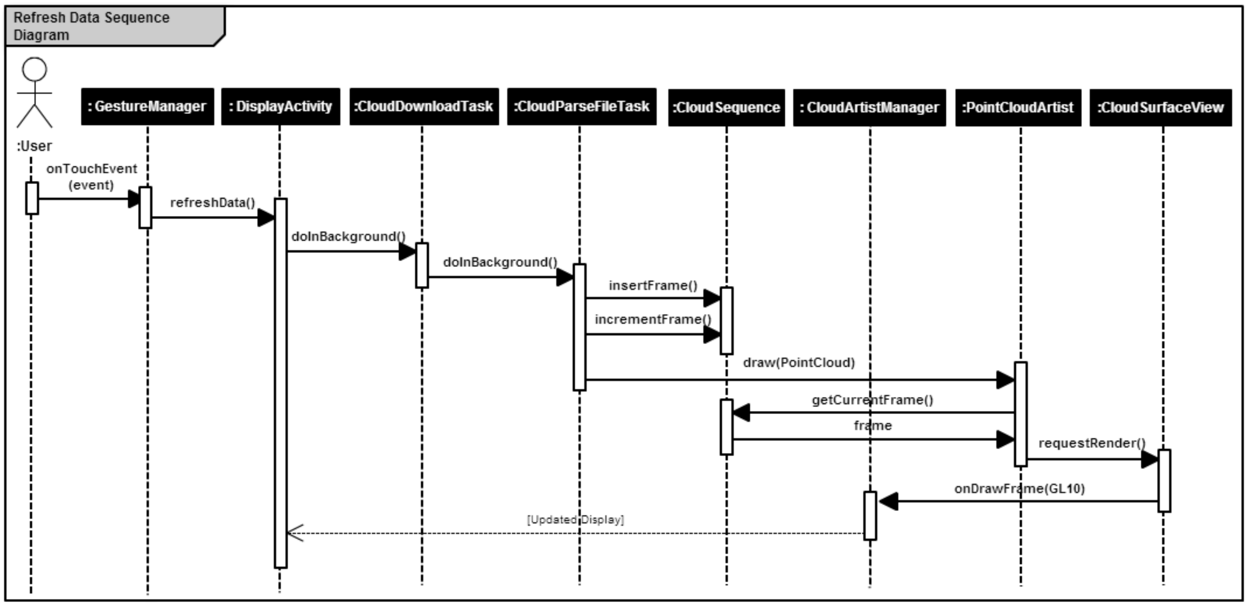


Figure 3: Refresh Data Sequence

This sequence model shows the objects involved in getting new LiDAR data from the server and rendering the new image on the app display. This involves downloading and parsing the next data file on the server and displaying the point cloud using OpenGL.

On the computer, the data ahs already been sufficiently reduced and converted to XYZ coordinates for the Witti app to process. This file is available on the server, where the app's CloudDownloadTask object identifies and downloads the file in the background. Upon completion, the CloudParseFileTask parses the data file for use by OpenGL. The CloudSequence object is used to save the new frame from the server as well as track the frames already displayed on the app. The PointCloudArtist is used to draw the point cloud frame, which requires rendering and drawing the frame using the CloudSurfaceView and CloudArtistManager objects.

Another optional functionality of the app not shown in the sequence diagram above is sensing the user's touch trajectory for the vehicle. This occurs after the user selects to draw a new path trajectory for the vehicle. This will involve calculating the coordinates on the Ground and determining whether this trajectory is clear of obstacles for the vehicle. If the path selected by the useris not valid, the the app will display an error message.

4.3 Settings

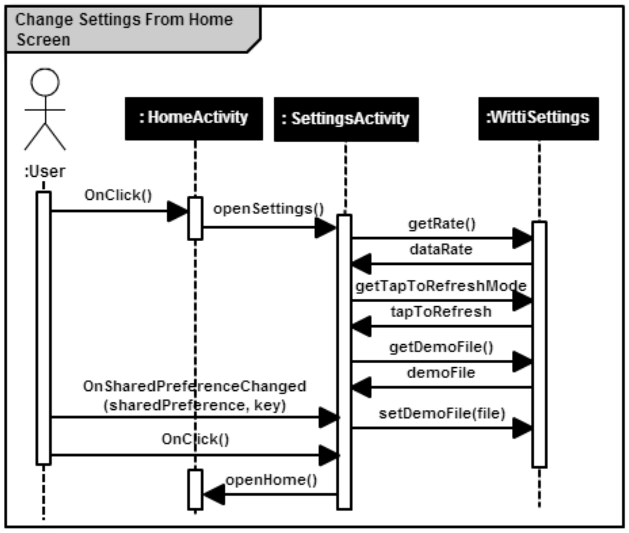


Figure 4: Change Settings Sequence

The user is currently viewing the HomeActivity user interface. The user then taps the Settings button which will call the HomeActivity's openSettings() method. This will initialize and create the Settings-Activity menu. The stored settings will be loaded from the WittiSettings onject upon opening the SettingsActivity. In the SettingsActivity menu, the user can make changes to the different settings, such as setting the demo file to be used in Demo mode. The SettingsActivity will implement the SharedPreferences.OnSharedPreferenceChangeListener to listen for changes made to the settings via the user's touch events. Upon changing a setting, the setting value will be set in the WittiSettings object. After the user has finished with the SettingsActivity, the back button will be used to initialize the HomeActivity and return the user interface to the title screen.

4.4 System Algorithms

4.4.1 Network exchange

The phone and host computer will communicate using using a local network and http protocol. On the phone this will be accomplished using its libraries. On the host computer, a simple server will be running that will respond to http requests with an index of the available sequences. For a live feed the server will respond to requests that include the last frame obtained, if the server has access to a newer frame it will send it ortherwise respond with an error.

4.4.2 Ground Plane

The ground plane will be determined by first selecting the points nearest the vehicle that are not on the vehicle itself. The RANSAC algorithm will be used to test possible planes by calculating the number of matching points and determining an estimate for the local terrain. It can be assumed that the four tiresof the vehicle are on a level plane, so any far deviation from this standard will be considered an error.

4.5 Phone Application Algorithms

4.5.1 3d Display

The LiDAR visualization will be accomplished using the OpenGL es2 libraries for Android. There are several different options depending on what we finally display. The ground plane can be displayed as a simple mesh with coloring based on the occupancy grid, which can be done using vertex coloring and smooth interpolation. The points can be displayed as sprites, and if we choose to create a mesh, this will need to be converted from the representation in point cloud library to one that can be loaded in OpenGL.

4.5.2 Path Drawing

The path will be drawn onto the ground plane from a 3d perspective. Because the camera is represented as a matrix that transforms from 3d position to a 2d pixel coordination, it is possible to use linear algebra to find a ray cast from the center of each pixel into the 3d space. After finding this line, it is possible to find its intersection with the ground plane. As the user drags a path accross the map, it may be desirable to smooth the path to prevent noise from making the path undrivable. The path can be displayed as a series of points or as a mesh.

4.5.3 Trajectory Clearance

Clearance of a trajectory will be determined by checking if there is space of a predetermined amount on both sides of the path. To facilitate this, the host computer will send an occupancy grid that it calculates based on a check of whether points exist above each sector with the ground plane as a reference. This will be a grid with a predetermined size in meters and indexing scheme. To save space in transfer, the open sectors will be sent as an array to the vehicle. A simple check would be to see if each point in that path is clear in a radius of sectors.

4.5.4 Trajectory Drivability

To determine if a path is reachable from a control point of view, we will assume that the vehicle is traveling below a safety speed. From this it is possible to assume that a minimum turn radius will be reachable. As long as the path, when suitable interpolated and smoothed, does not have tighter turns that this, the path is determined to be reachable. This problem could easily be expanded to include issues of under and over shoot.

4.6 Host Computer

4.6.1 Point Reduction

Point reduction can be accomplished in many ways. The first method is to limit the output of the Velodyne by setting speed, field of view, and range. Next it is possible to programmatically ignore points that are deemed unnecessary. One option for reducing the number of points is to use a voxel grid to reduce the resoultion of the 3d space. This can be done using the Point Cloud Library (PCL). Another possibility is to randomly discard data points.

Part II
Design and Test

5 Class Design

5.1 Class Diagrams

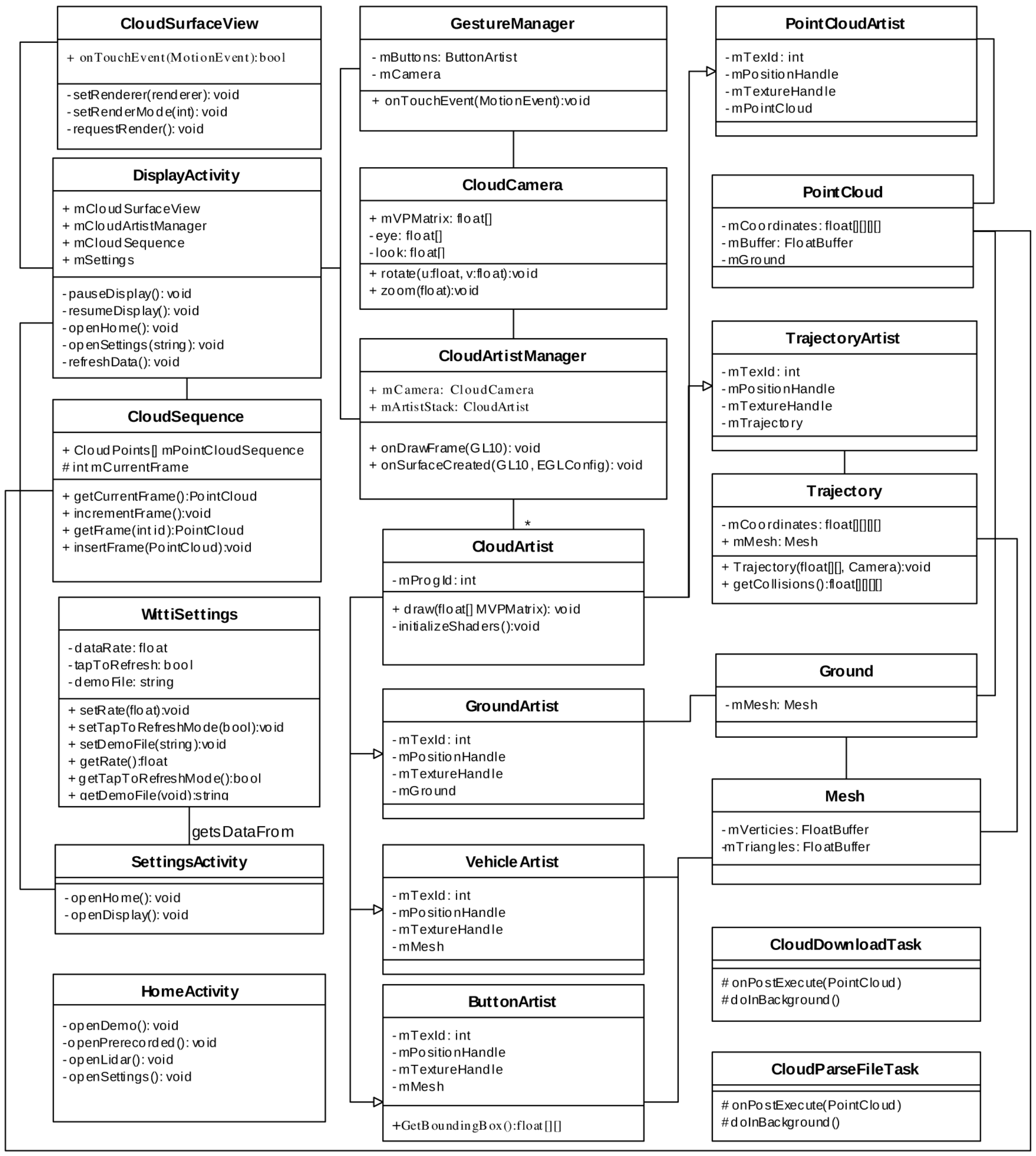


Figure 5: App Classes

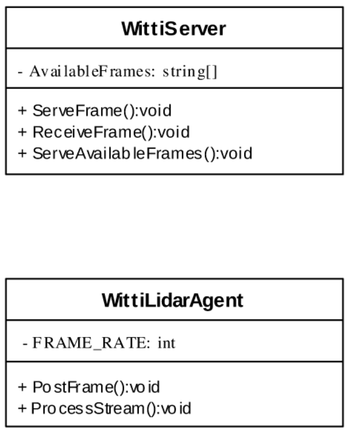


Figure 6: Host Classes

6 Testing Strategy

2.1 'B' Verification

B.1 **Display Data**: An already prepared point cloud file is stored on the phone and can be loaded into OpenGL as a buffer.

B.2 **Manual Data Refresh**: The phone will also make a new request and get the latest file from the server, this can be simulated with a dummy point cloud where all the values are 1s and then 2s, loaded based on the newest time.

B.3 **Server Download**: A prepared point cloud file is loaded both on to the phone and the server, the phone will download and load the file and compare it to results from loading the local file, to ensure that they match.

B.4 **Server Upload**: As in B.3 except the host computer will be the server, or it will upload the file onto a remote server.

B.5 **Demo Playback**: A prepared point cloud sequence file is stored on the phone and when loaded, the user refresh button presses change the point cloud that is loaded into an openGL draw function.

2.2 'A' Verification

A.1 **Automatic Data Refresh**: Similar to B.1, but the phone will request at a given request rate, for the newest frames, and dummy value will be suplied with timestamps to show that they are correctly updating.

A.2 **Draw Trajectory**: The phone will generate reasonable paths and determine the user input required to create them. It will simulate this user input and determine if the created path is similar to the initial one.

A.3 **Trajectory Clearance**: In combination with part A.2, paths can be generated that are both good and bad. The phone will respond to the simulated inputs with behavior appropriate to the path. The original paths will be created independently by hand using another method to cross verify the results.

A.4 **Trajectory Drivability**: Same method as in A.3 but checking for impossible paths, given our definition.

A.5 **Perspective Change**: The phone will simulate user input to change the perspective, and the OpenGL representation of a camera will change to expected values or in the expected way.

A.6 **Dynamic LiDAR Conversion**: Given the tests B.3 and B.4, the data uploaded will be created by a program based on current code to capture LiDAR frames, a time stamp will be used to verify that the data is transferred within an acceptable delay.

A.7 **JAUS Compatible**: The JAUS compatible messages will be read by another JAUS component and checked that they match a preprocessed file.

7 Integration with Platform

The application will be interacting with the platform's server in order to communicate with the host computer. When the application starts the Launch mode, an initialization process between the server and the device is launched. The prerequisite for this is that a WittiServer will already be running and the phone readable LiDAR data is available on the server.

After a connection is made, the app will download and parse LiDAR data from the server. After this point, new data from the server will only be downloaded and parsed when the user taps the screen, indicating a refresh action. Optionally, the application may automatically download new data from the host computer based on a set refresh rate. The downloaded data will be managed by the CloudSequence, which will be used to provide the current PointCloud frame to render on the app display.

Part III
Implementation Plan

8 Task Allocation and Breakdown

8.1 Breakdown

Broadly, the tasks are broken down into managing the host computer and server activities, downloading and processing files onto the app, displaying the point cloud data using OpenGL, managing the app settings, and handling touch events on the app. Although task allocation may vary throughout the project, the initial allocations for these "B requirement" tasks are listed below. The "A requirements" task allocation will be determined upon finalization of which requirements will be implemented.

8.2 Responsiblities

8.2.1 Brian Smith

Requirements: B.4 and B.5

Classes: WittiServer, WittiLidarAgent, DisplayActivity, and GestureManager

8.2.2 Brianna Heersink

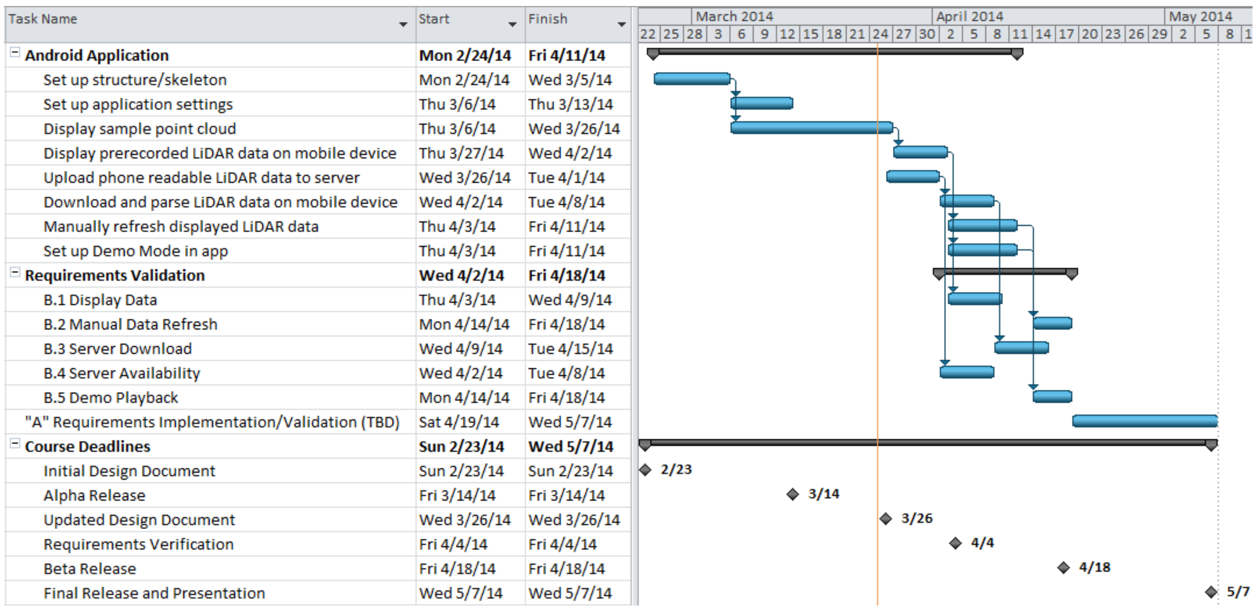
Requirements: B.2 and B.3

Classes: WittiSettings, SettingsActivity, HomeActivity, CloudDownloadTask, CloudParseFileTask, and CloudSequence

8.2.3 Alex Warren

Requirements: B.1

Classes: CloudSurfaceView, CloudArtistManager, CloudArtist, ButtonArtist, PointCloudArtist, and PointCloud



Task Name	Start	Finish
Android Application	**Mon 2/24/14**	**Fri 4/11/14**
Set up structure/skeleton	Mon 2/24/14	Wed 3/5/14
Set up application settings	Thu 3/6/14	Thu 3/13/14
Display sample point cloud	Thu 3/6/14	Wed 3/26/14
Display prerecorded LiDAR data on mobile device	Thu 3/27/14	Wed 4/2/14
Upload phone readable LiDAR data to server	Wed 3/26/14	Tue 4/1/14
Download and parse LiDAR data on mobile device	Wed 4/2/14	Tue 4/8/14
Manually refresh displayed LiDAR data	Thu 4/3/14	Fri 4/11/14
Set up Demo Mode in app	Thu 4/3/14	Fri 4/11/14
Requirements Validation	**Wed 4/2/14**	**Fri 4/18/14**
B.1 Display Data	Thu 4/3/14	Wed 4/9/14
B.2 Manual Data Refresh	Mon 4/14/14	Fri 4/18/14
B.3 Server Download	Wed 4/9/14	Tue 4/15/14
B.4 Server Availability	Wed 4/2/14	Tue 4/8/14
B.5 Demo Playback	Mon 4/14/14	Fri 4/18/14
"A" Requirements Implementation/Validation (TBD)	Sat 4/19/14	Wed 5/7/14
Course Deadlines	**Sun 2/23/14**	**Wed 5/7/14**
Initial Design Document	Sun 2/23/14	Sun 2/23/14
Alpha Release	Fri 3/14/14	Fri 3/14/14
Updated Design Document	Wed 3/26/14	Wed 3/26/14
Requirements Verification	Fri 4/4/14	Fri 4/4/14
Beta Release	Fri 4/18/14	Fri 4/18/14
Final Release and Presentation	Wed 5/7/14	Wed 5/7/14

Figure 7: Project Timeline

8.3 Global/Shared Tasks and Experience

The integration of all components will be a shared task between all group members. Additionally, all members will write tests, although the tests may cover more than just the classes they wrote themselves. Finally, all members will review each other's code and contribute to required documentation that is submitted throughout the project.

The task allocation was determined to allow the work to be divided and then and integrated easily. Alex was interested in gaining more experience with OpenGL, so his task allocation included working on the data visualization. The remaining tasks were split between Brian and Brianna.