6-12-2018

vPlot

Dante Dalla Gasperina
Santa Clara University, ddallagasperina@scu.edu

Kush Mahajani
Santa Clara University, kmahajani@scu.edu

Alex Martin
Santa Clara University, atmartin@scu.edu

Collin Walther
Santa Clara University, cwalther@scu.edu

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: June 11, 2018

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Dante Dalla Gasperina
Kush Mahajani
Alex Martin
Collin Walther

ENTITLED

vPlot

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Ahmed Amer

Christopher Kitts

Nam Ling

vPlot

by

Dante Dalla Gasperina
Kush Mahajani
Alex Martin
Collin Walther

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 12, 2018

vPlot

Dante Dalla Gasperina
Kush Mahajani
Alex Martin
Collin Walther

Department of Computer Science and Engineering
Santa Clara University
June 12, 2018

ABSTRACT

Robotic systems under development in the Santa Clara University Robotic Systems Laboratory (RSL) generate large amounts of data that must be interpreted in real-time. Many dimensions of this data must be visualized at once, such as temperature, location, and certainty of the measurement. Current data visualization softwares (such as Mathematica and Simulink) are ill-suited to visualize this much data, due to lack of customization. To solve this, we propose a system that allows users to view real-time streaming data in a virtual reality environment. This allows the user to easily interpret large, detailed datasets through an intuitive interface.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Robotic systems currently under development in the Santa Clara University Robotics Systems Laboratory (RSL) generate large quantities of data that need to be visualized in real time. The RSL's Multi-Robot Cluster System consists of multiple robots working together in a *cluster*. Each cluster has a set of variables describing the overall cluster, such as its size and shape [1]. Clusters may have up to tens of robots and robots may independently or cooperatively gather data. Each data point exists as a coordinate in 3D space, and may have additional features such as temperature or flow direction, as well as metadata like variance and recency. To properly interpret the data, it is necessary to simultaneously view these features for each point. It is preferable to represent all of the features within a single plot. However, this is difficult to achieve while preserving visual clarity.

1.2 Existing Solutions

Existing solutions for viewing multidimensional dataare insufficient because they lack flexibility in the way individual data points are represented. They are unable to display more than just a few features per point, usually limited to their spatial coordinates and color. This is inadequate for modern datasets, many of which include far more than four features. Current solutions are also hampered by unintuitive user interfaces; they often require the use of programmatic statements to visualize data. For example, Wolfram Mathematica, a popular information visualization tool, requires users to learn the Wolfram programming language in order to process and display data [2]. *pandas*, a data analysis library, requires familiarity with Python. Both of these solutions have a significant learning curve. In addition, neither solution is able to plot more than three dimensions of data at once. Mathematica is supports a maximum of three dimensions [3], and *pandas* has no out-of-the-box support for interactive 3D plots [4].

1.3 Solution

In order to overcome these limitations, we have designed and implemented vPlot. vPlot is a general-purpose data visualization software that can concurrently display multiple data features with an intuitive interface to easily manipulate their presentation. Extra features are represented as visual characteristics such as color and size, enabling users to view a greater amount of meaningful data at once. Users may choose how features are mapped to visual characteristics. Additionally, we leverage virtual reality hardware to display the data in true 3D. This lends additional clarity to the plotted data that would be unattainable via a 2D image on a monitor.

This report describes the design and implementation of vPlot. Chapter 2 identifies the requirements for our system, while Chapter 3 elaborates the use cases of the system. Chapter 4 presents the architecture of the system. Chapter 5 discusses the technologies required to implement this system. Chapter 6 provides mockups of the proposed user interface. Appendix A contains the timeline for designing and developing the system while Appendix B identifies and analyzes the risks we encountered. Chapter 7 lays out our testing plan. In Chapter 8, we discuss the ethical impacts of vPlot. We describe the obstacles we encountered in Chapter 9. A user manual is included in Appendix C. Finally, Chapter 10 concludes with the lessons we learned over the course of the project and potential future extensions to vPlot.

Chapter 2

System Requirements

Listed below are the requirements that define the successful completion of our system. They are classified either as functional — those that involve capabilities that the system either does or does not have — or non-functional — qualities of the system that exist on a spectrum of completion. Within these classifications, each requirement is further categorized as critical, recommended, or suggested in decreasing order of importance.

2.1 Functional Requirements

Critical

- The system must be able to plot data points in three spatial dimensions. *Satisfied*
- The system must be able to represent two more data features in addition to three spatial dimensions via the color and size of each point. *Satisfied*
- The system must allow the user to choose which data features correspond to which of the spatial dimensions or additional representations in the plot. *Satisfied*
- The system must support fetching plottable data from files of a standardized, human-readable format. *Satisfied*
- The system must support receiving plottable data from a remote source streamed over a network connection. *Satisfied*

Recommended

- The system should support additional data features via representing data points as vector arrows, whose direction and magnitude correspond to data features. *Satisfied*
- The system should be able to save the contents and settings of a plot to a file. *Not satisfied*
- The system should be able to display multiple series of data in the same plot. *Not satisfied*

- The system should be able to display multiple different plots at once. *Not satisfied*
- The system should allow the user to rebind the controls used for actions in the virtual environment, such as positioning the camera and interacting with plots. *Partially satisfied*

Suggested

- The system may support multiple file types for reading and writing plottable data. *Not satisfied*
- The system may integrate custom voice commands in addition or as an alternative to controller-based input. *Not satisfied*

2.2 Non-Functional Requirements

Critical

- The system must respond to streamed input quickly enough for updates to a plot to be perceived by the user as taking place in approximately real-time. *Satisfied*
- The system must offer a user interface that is easy to use by being as simple and intuitive as possible while not compromising on functionality. *Satisfied*

Recommended

- The system should maintain a frame rate fast enough to avoid inducing nausea in the user. *Satisfied*

2.3 Design Constraints

Critical

- Use of the system involving the VR headset must not require using the keyboard for input simultaneously. *Satisfied*
- The system must be compatible with the Oculus Rift VR headset. *Satisfied*
- The system must be compatible with the Microsoft Windows operating system. *Satisfied*

Recommended

- The system should be compatible with the HTC Vive VR headset. *Satisfied*
- The system should be compatible with additional operating systems (e.g. Apple macOS, Linux). *Partially satisfied - Linux not tested*

Suggested

- The system may be compatible with additional VR solutions. *Partially satisfied - not tested*

Chapter 3

Use Cases

This section describes the independent usage scenarios for the system. The three major use cases are plotting data streamed from a network server, plotting data streamed from a network client, and plotting data read from a file. The network streaming use cases are of particular importance, as they are the most applicable to the needs of the RSL. See Fig. 3.1.

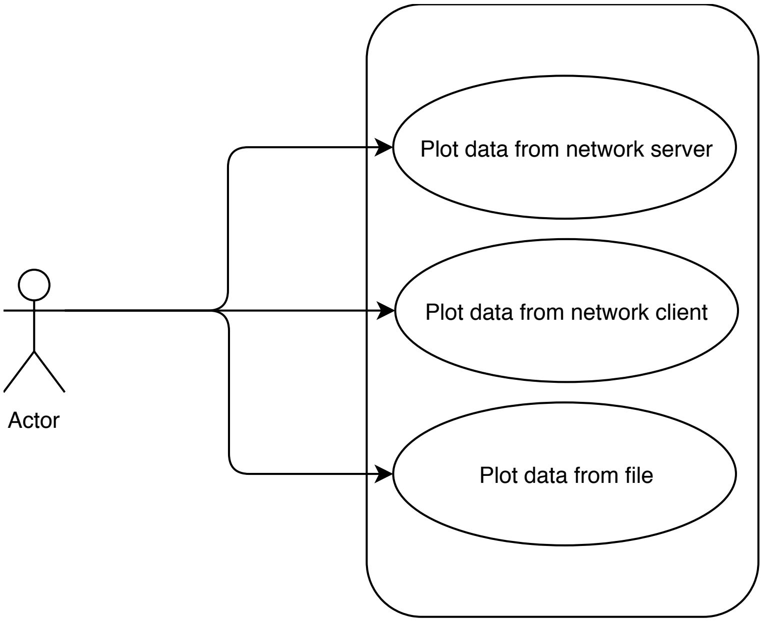


Figure 3.1: Use case diagram

3.1 Case 1: Plot Data From Network Server

Goals

- Accept data from a network stream

- Plot data

Actors

- User

Preconditions

- Data source is connected to the other end of the network stream

Postconditions

- Data can be viewed and manipulated in the user interface

Exceptions

- Network connection is unexpectedly terminated

- Network stream payload is malformed

3.2 Case 2: Plot Data From Network Client

Goals

- Accept data from a network stream

- Plot data

Actors

- User

Preconditions

- Data source is connected to the other end of the network stream

Postconditions

- Data can be viewed and manipulated in the user interface

Exceptions

- Network connection is unexpectedly terminated

- Network stream payload is malformed

3.3 Case 3: Plot Data From File

Goals

- Read data from a file
- Plot data

Actors

- User

Preconditions

- The file is accessible

Postconditions

- Data can be viewed and manipulated in the user interface

Exceptions

- The file or its contents are malformed

Chapter 4

Architecture

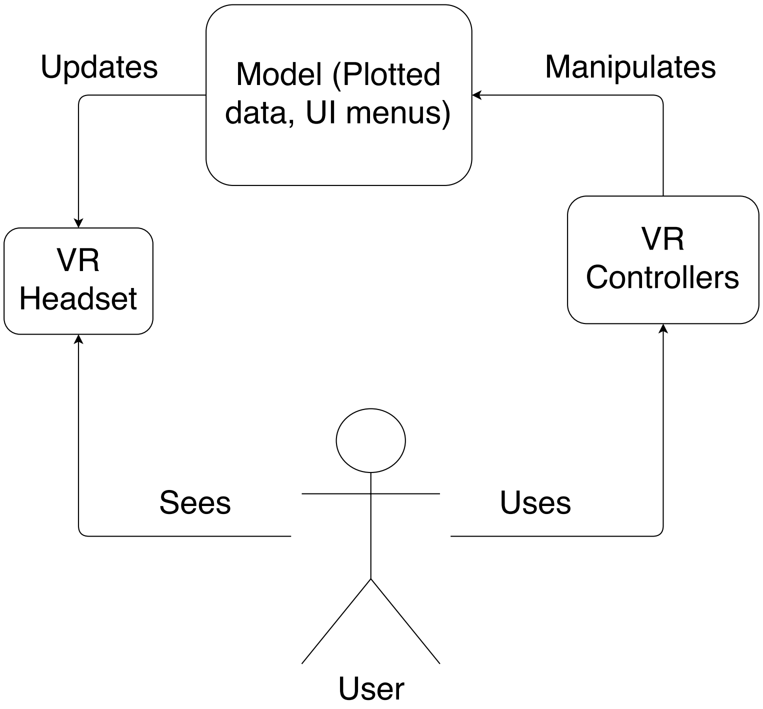


Figure 4.1: Model/view/controller architecture diagram

Since the aim of this project is to present already-produced data to the user in an intuitive way, we have designed the system using a model/view/controller architecture, the standard architecture for implementing a GUI. The user manipulates the VR controllers in his hands, which in turn modifies how the data is presented to the user. This data architecture is represented in Fig. 4.1.

Chapter 5

Technologies Used

In designing any virtual reality software product, the specific technology that is used is dependent on a few design choices. These choices are:

- Which VR headset (e.g., Oculus Rift, HTC Vive, Gear VR)
- Which 3D development platform (e.g., Unity or Unreal Engine)
- Which development language

For this project, we decided to develop for the Oculus Rift using C# in Unity.

5.1 VR Headset

We settled on the Oculus Rift as the target VR headset for this project for one main reason: an Oculus Rift was already available in the RSL, so we wouldn't have to wait for our funding request to be approved before we began VR development. Additionally, development across multiple VR devices is very similar, so our choice largely depended on convenience.

5.2 Development Platform

For our development platform, the two main choices we had to choose between were Unity and Unreal Engine. Our research revealed that development time was shorter with Unity than with Unreal Engine, due to Unity's comprehensive documentation and the fact that Unreal Engine is targeted toward experienced game developers. Because there is a strict time frame for the completion of our project, we chose Unity.

5.3 Programming Language

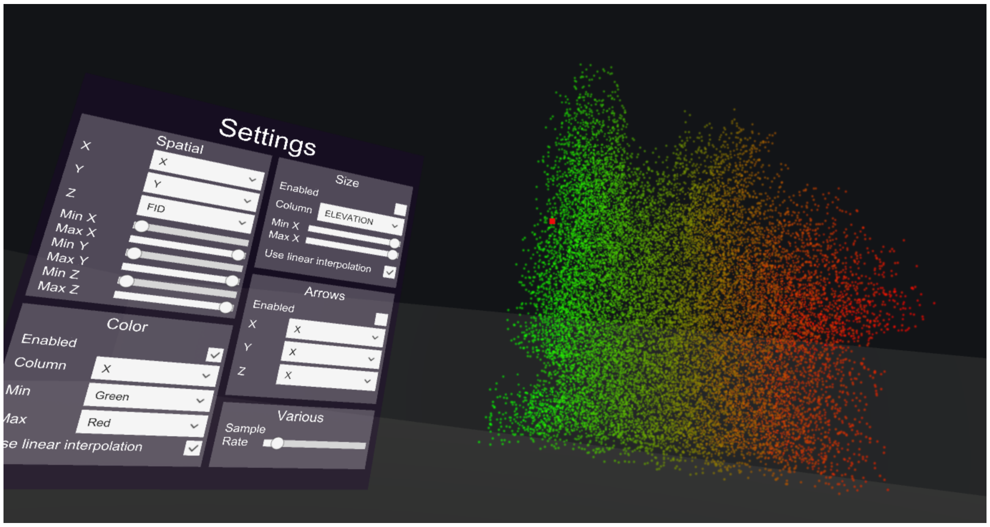
The programming language choices that we had were limited to C# and JavaScript, due to our choice of Unity as our development platform. C# offers a rich development environment as part of the .NET framework, object-orientation for scalable design of large projects, and slightly better performance than JavaScript. JavaScript offers a friendlier, less verbose syntax, which can reduce development time, and support for object-orientation. Of these two options, we chose C#, due to its advantages in performance and scalability. C#'s static typing also makes it much easier to debug, as opposed to the dynamically-typed JavaScript.

5.4 Other Technologies Used

Other technologies that are used in our project are TCP socket communication and usage of the CSV file format. We use TCP sockets to communicate directly with real-time streams of plottable data, and we use the CSV file format to read and store unchanging plottable data. To develop this project, we used Git for version control, the Unity development environment, and Visual Studio as our IDE (since it integrates very well with the Unity development environment).

Chapter 6

Application Overview



Settings

Spatial

X: X

Y: Y

Z: FID

Min X

Max X

Min Y

Max Y

Min Z

Max Z

Size

Enabled □

Column: ELEVATION

Min X

Max X

Use linear interpolation ☑

Arrows

Enabled □

X: X

Y: X

Z: X

Color

Enabled ☑

Column: X

Min: Green

ax: Red

se linear interpolation ☑

Various

Sample Rate

Figure 6.1: Conceptual model of the operational system

Fig. 6.1 shows a general example of what the final product looks like. In this example, the user is analyzing a dataset consisting of multiple elevation measurements across the city of Chesapeake, VA. There are two main elements that the user will be able to see and manipulate when they use the system:

1. The plotted data

2. The UI menu

6.1 Plotted Data

Most of the user's field of view will be occupied by the data they are trying to analyze. The user will have the ability to move around the data in order to get different angles on it, zoom in and out, and rotate the data. When the user makes changes to how the data is displayed via the UI, those changes will be instantly reflected in the plotted data.

Each point in the dataset is represented by a single visual point (or an arrow, if the user wants to assign direction to each point). These points are highly customizable, and the user can manipulate the menu in order to change what features of the data correspond to different visual characteristics. For example, in Fig. 6.1, you can clearly see how color correlates to the x-axis of the plotted data. But it could also be correlated to the y-axis if the user so desired.

When being used with a static dataset (i.e., one that is contained within a .csv file), the data points will all be loaded when the application launches, and no more will load after that. When the application is being used with real-time streaming data, points will render as soon as they are received by the application; no intervention from the user is necessary to make newly-received points render.

6.2 User Interface

While using the application, the user can press the Y button on the Oculus controller to open or close the menu (see Fig. C.1 and Fig. C.2 for a complete overview of the controls). When the menu is closed, the user is free to rotate around the data, zoom in and out, and move their physical head to get a different viewing angle.

When the user opens the menu, they are promped with a number of dropdown menus, sliders, and checkboxes. A complete list of these UI elements are as follows:

- Spatial section: controls where points are displayed.
  - X: The spatial x-axis.
  - Y: The spatial y-axis.
  - Z: The spatial z-axis.
  - Min X: Prevents points from being plotted if the feature corresponding to the x-axis for a given point is less than a certain value, determined by this slider. By default, it allows all points to be plotted.
  - Max X: Like Min X; prevents points from being plotted if the feature corresponding to the x-axis for a given point is greater than a certain value, determined by this slider. By default, it allows all points to be plotted.
  - Min Y: See Min X.
  - Max Y: See Max X.
  - Min Z: See Min X.
  - Max Z: See Max X.
- Color section: controls how points are colored.

  - Enabled: Enables/disables color. When disabled, all points are white. When enabled, points will be colored according to which feature is assigned to color in the dropdown below. Points where the given feature is small will be assigned a color close to the minimum color, whereas points where the given feature is large will be assigned a color close to the maximum color.
  - Column: Which feature/column of the data corresponds to color.
  - Min: The minimum color. There are 8 options for the minimum/maximum color: black, white, magenta, red, green, yellow, orange, and cyan.
  - Max: The maximum color.
  - Use linear interpolation: When a point is assigned a color, this determines whether a point's color is linearly interpolated between the minimum and maximum colors based on its value, or its rank in relation to the other points. This is useful if many points are clustered around a particular value.

- Size section: controls the size of the points.
  - Enabled: Enables/disables size. When disabled, all points are the same size. When enabled, points are sized according to which feature is assigned to size. Points where the given feature is small will be assigned a size close to the minimum size (given by the Min slider), whereas points where the given feature is large will be assigned a size close to the maximum size (given by the Max slider).
  - Column: Which feature/column of the data corresponds to size.
  - Min: The minimum size.
  - Max: The maximum size.
  - Use linear interpolation: When a point is assigned a size, this determines whether a point's size is linearly interpolated between the minimum and maximum colors based on its value, or its rank in relation to the other points. This is useful if many points are clustered around a particular value.

- Arrows section: controls how points are represented as arrows. In datasets where each point has a sense of direction (i.e., the dataset has three features: one containing magnitude of x direction, one containing magnitude of y direction, and one containing magnitude of z direction), this feature is useful for visualizing the direction of each point.
  - Enabled: Enables/disables arrows. When disabled, points are rendered as points. When enabled, points are rendered as arrows.
  - X: The feature corresponding to magnitude of the x component of the direction.

  - Y: See X.
  - Z: See Z.
- Various
  - Sample Rate: Allows the user to randomly sample the data. If the dataset has too many points to display at once, the user can turn down the slider to limit the amount of points that are simultaneously displayed.

Interacting with UI elements is fairly simple:

- For dropdowns, move the reticle over the dropdown and press A to open it. Then select the desired option from the dropdown menu.
- For checkboxes, move the reticle over the checkbox and press A to toggle it.
- For sliders, move the reticle over the circle on the slider and press A to select it. Once it is selected, move the left control stick left or right to increase or decrease the slider.

Chapter 7

Testing Procedure

Our testing came down to three main categories: usability testing of the user experience, unit testing of the network functionality, and real world testing with the robots in the RSL.

7.1 Usability Testing

Usability testing was an important factor in the development of vPlot as VR is an emerging field and at this point is not an intuitive platform. We began our testing by implementing our basic features on a desktop and monitor with no VR capabilities. This ensured that everything was working correctly and further issues on subsequent tests could be isolated to the VR platform. Once everything was working as anticipated on the desktop and monitor, vPlot was moved into VR. All existing capabilities of vPlot were then tested in VR to make sure all functionality was maintained during the switch. This was not an easy feat; the menu transition will be discussed in detail during the obstacles encountered section. Once all features were tested and functioning in VR, different control schemes were tested. All of the control schemes were based upon the Oculus Touch controllers which are the default controllers for the Oculus Rift.

After the controls were as smooth and intuitive as possible, we moved onto testing different interaction schemes. Basically, we tested the way users could interact with the data and the menu. Once we believed interaction was good, we transitioned into user testing. During this time, we had users who were not directly involved in the implementation of vPlot try vPlot and give us feedback. We then worked this feedback into vPlot.

7.2 Unit Testing

The capabilities of the network functionality of vPlot were tested with unit tests. Basically, we set up an even amount of tests for vPlot to act as client or server for a known set of data. If vPlot was acting as a client for a particular test, we would have it send data to a server test module and compare the data sent to that which was received. If vPlot was acting as a server for a particular test, we would send it data from a client test module and compare the data received to that which was sent. Once all test cases completely successful, we were confident with our network functionality.

7.3 Real World Testing

Real-world testing was done to test vPlot's ability to integrate with a robotic system in the RSL. We tested with the RSL's ClusterControl project which uses sensor-equipped robots and the OptiTrak tracking system. The robots are communicating with a base station to which they are sending their positional and rotational data. The base station connects to the vPlot server, aggregates sensor data, forms CSV file, and sends it to vPlot. To confirm the success of these tests, we compared the plotted data to the base station. By testing with this robotic system, we are confident vPlot's functionality can be generalized to different robotic systems and plot data from these robots that may be different than positional and rotational data.

Chapter 8

Ethical Considerations

We have concluded that vPlot itself possesses no innate ethical dimension. Being a tool, our project assumes the ethical context of whatever purpose it is used for. As vPlot is primarily intended to represent measured data for scientific and engineering applications, there is the concern that a user might handle a plot in such a way as to willfully misrepresent the data within, thereby involving our software in an unethical process. However, while our project was always intended to suit a general use case, it was developed based on requests and feedback from the RSL. As such, our project may be considered ethical provided we are in agreement with the RSL and its applications for our system. Regarding other potential users of the system, we will simply have to accept that we do not have control over their actions.

Separate from the purpose of the project itself is the nature of the work ethic involved in any software engineering undertaking. It has been our intention from the beginning not just to treat our senior design project as a learning experience, but to create something that the RSL will find useful even after we have moved on from the university. A large part of achieving this goal is ensuring that students and staff from outside our team can continue development on our project as necessary, so that it can be adapted to the constantly evolving needs of the RSL. Documentation plays an important role in project development and maintenance for this reason. Here, documentation refers not only to the user manual but to comments within the source code that help direct the efforts of future development. Neglecting to create such documentation for the project could have saved time and effort in the short term, but ultimately would have rendered the project unmaintainable.

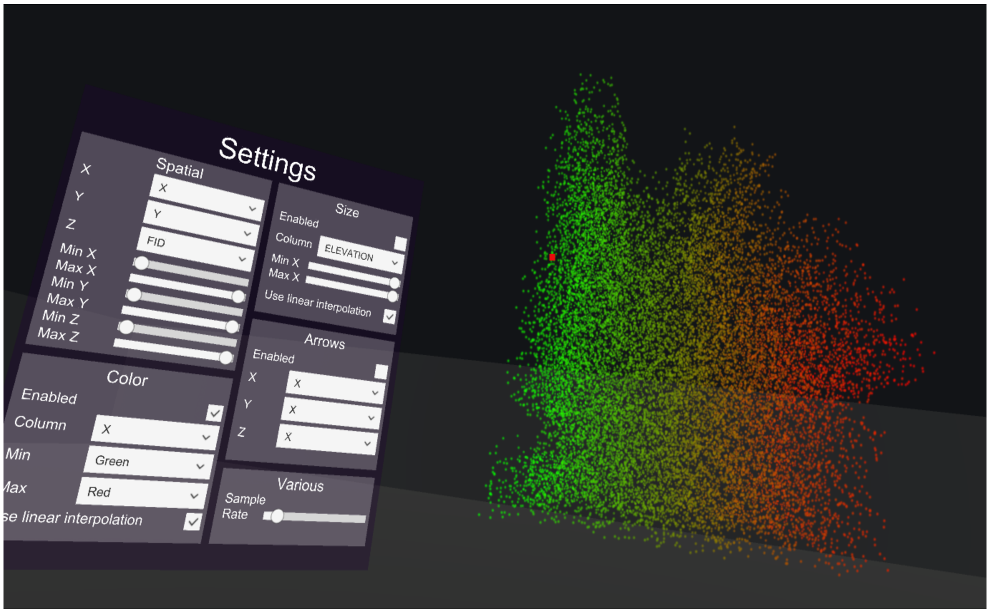
Also of great importance are the design decisions that form the foundation of our project; even the most thorough documentation cannot save a project built atop a bad initial design. Often during the completion of our project we were faced with decisions between what is most robust, and what is easiest to implement. We discovered that there were cases that called for either approach. Software components that were unrelated to the purpose of our project but were necessary to construct a working demo, such as the launcher user interface, could safely be done the quick and easy way, freeing up valuable time for more pressing issues. On the other hand, critical components, such as the data plot itself, had to be refactored multiple times over the course of development to ensure they were both performant and well-designed. If such components were written poorly, then in the future the RSL may find it more expedient to throw away our efforts and begin again from scratch, rather than build upon them as we intend.

Chapter 9

Obstacles Encountered

During the implementation of vPlot, we ran into many obstacles that caused us to re-evaluate our design and processes. In this section, three critical obstacles will be discussed as well as how each obstacle was dealt with. The first obstacle appeared quite early in development and dealt with versioning auto-generated files. When a Unity project is created, Unity generates a large amount of project files that are not supposed to be directly edited by the programmer, but edited by Unity as the project progresses. On multiple occasions, pushing these files to git caused merge errors that were hard to manually address. When looking at two versions of the same Unity project file side by side, it was difficult for us to choose which portions of each file were most up-to-date. We partially alleviated this issue by using one of Unity's built in merge tools, but it did not completely handle the problem. On occasion, we still had to manually sift through select files and resolve merge conflicts ourselves.

The next obstacle came from the processing power requirements of vPlot for large data sets. If a large data set is loaded all at once, vPlot gets very slow and the framerate drops significantly. In order to fix this problem, the Sample Rate slider was implemented. The Sample Rate slider samples a given percentage of a data set randomly based upon the value of the slider. By implementing this, the processing power of an individual computer will never be a handicap for viewing a data set. Even the lowest powered computers that can handle our VR application will be able to sample some percentage of any data set given. Shown in Fig. 9.1 and Fig. 9.2 is the difference in viewing the data with lower and higher sample rates.



Settings

Spatial
X: X
Y: Y
Z: FID
Min X
Max X
Min Y
Max Y
Min Z
Max Z

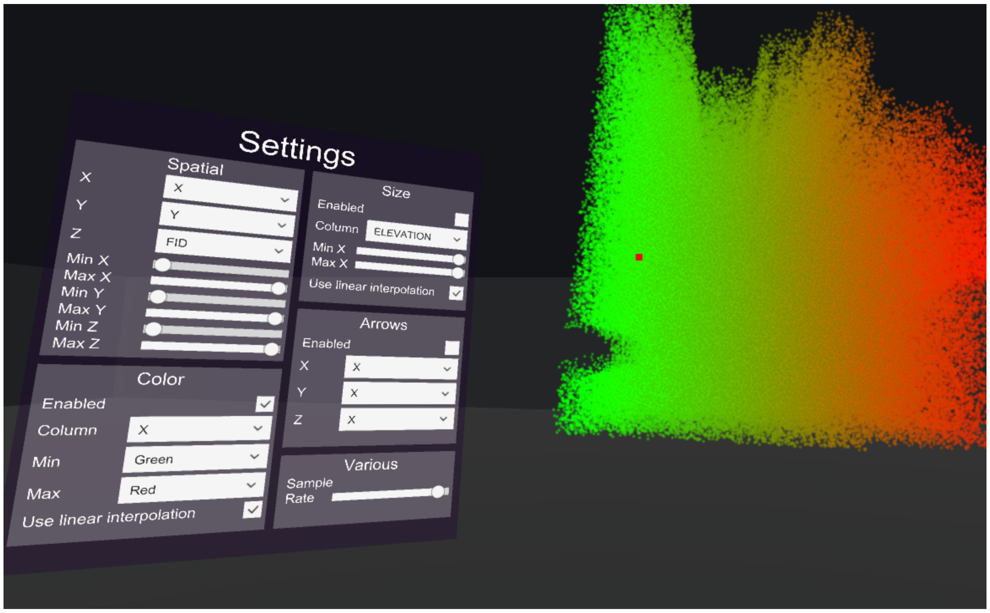
Size
Enabled ☐
Column: ELEVATION
Min X
Max X
Use linear interpolation ☑

Arrows
Enabled ☐
X: X
Y: X
Z: X

Color
Enabled ☑
Column: X
Min: Green
lax: Red
se linear interpolation ☑

Various
Sample Rate

Figure 9.1: Samping of 5% of the data points



Settings

Spatial
X: X
Y: Y
Z: FID
Min X
Max X
Min Y
Max Y
Min Z
Max Z

Size
Enabled ☐
Column: ELEVATION
Min X
Max X
Use linear interpolation ☑

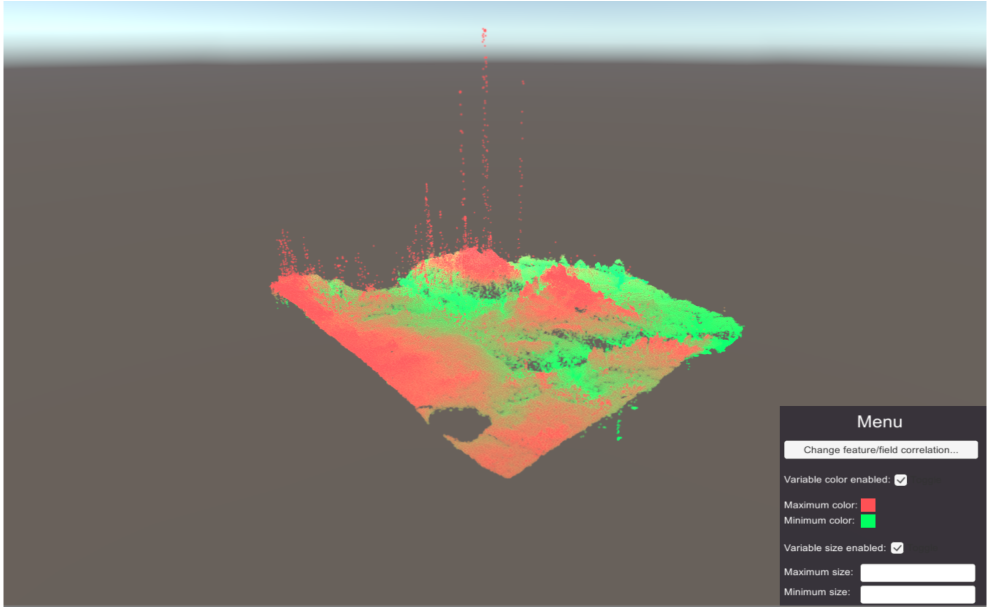
Arrows
Enabled ☐
X: X
Y: X
Z: X

Color
Enabled ☑
Column: X
Min: Green
Max: Red
Use linear interpolation ☑

Various
Sample Rate

Figure 9.2: Sampling of 95% of the data points

Another obstacle was the menu design, regarding both its implementation and interactivity. The menu went through 3 major phases. The first phase originated from the classic implementation of a menu for a desktop application. We originally thought that the best place for our menu to be was the bottom right corner as it could be easily accessed whenever the user needed it.



Variable color enabled: ☑

Maximum color: 🟥

Minimum color: 🟩

Variable size enabled: ☑

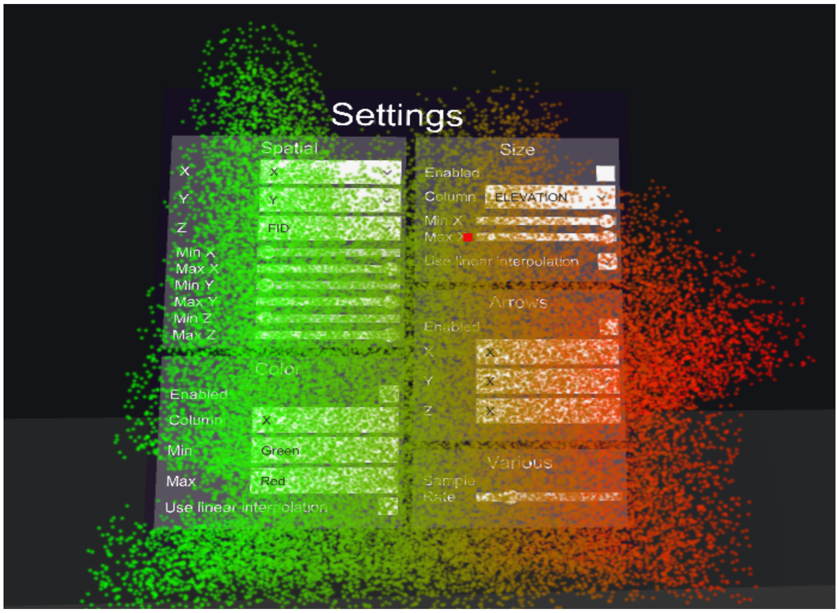
Maximum size:

Minimum size:

Figure 9.3: The menu locked to the bottom right corner (as seen in conceptual model)

This worked well with the initial testing of vPlot on a desktop and monitor. When testing transitioned into VR, this menu implementation ended up not working well. During the use of a VR application, the user will move their head towards an element on screen in order to get the item centered in the their field of view. With the menu being locked to the right corner, the user could never center the menu in their field of view; our design went against an expected design decision for an application like ours which made it very unintuitive. In addition to it being unintuitive, it was painful to use. In order to interact with the menu, the user needed to strain their eyes to the corner.

To fix this issue, we decided to put the menu into the worldspace of vPlot. What this means is that the menu is no longer locked onto the screen, but exists as a 3D object.



Settings

Spatial

X: X

Y: Y

Z: FID

Min X

Max X

Min Y

Max Y

Min Z

Max Z

Color

Enabled ☑

Column: X

Min: Green

Max: Red

Use linear interpolation ☑

Size

Enabled □

Column: ELEVATION

Min X

Max X

Use linear interpolation ☑

Arrows

Enabled

X: X

Y: X

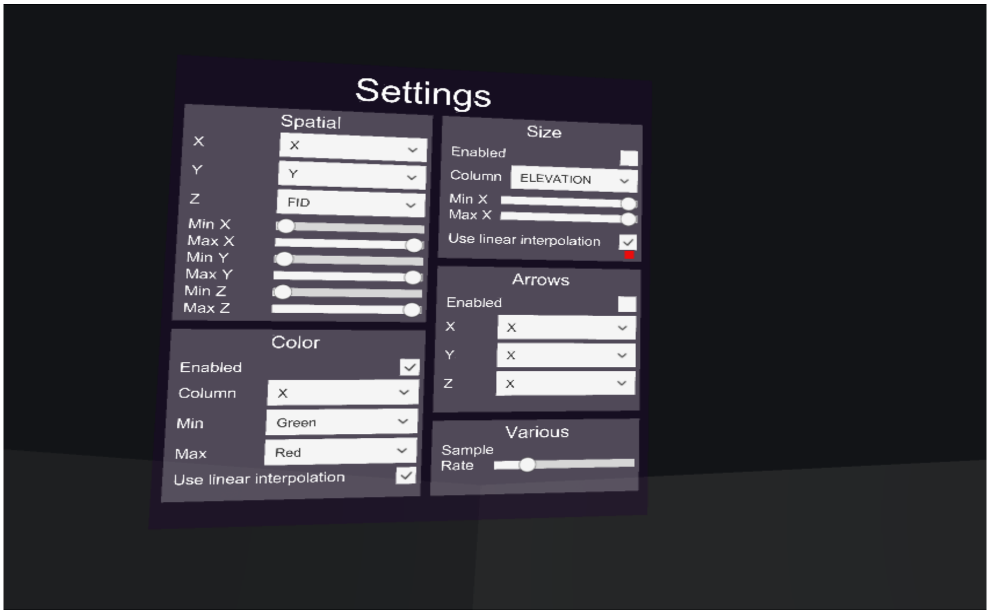
Z: X

Various

Sample Rate

Figure 9.4: The menu in worldspace

With this change, a user is able to intuitively find the menu in worldspace and focus their gaze upon it when they need to use it. Though this was a great step forward, putting the menu into the worldspace caused new issues. First, the menu interfered with the view of the data and vice versa. Because neither of these elements should be obstructed, we moved our menu out of the way of the data. Second, by rotating around the data, the orientation of the menu rotated as well. This led to the user trying to interact with an upside down menu or ending up behind the menu. To fix this problem, we implemented a feature to summon the menu into the users center of vision with the click of the button, while also allowing the closing of this menu with the same button click. While the menu is active, all rotation and circulation controls are disabled, so the user does not reach a state where they can no longer use the menu properly.



Settings

Spatial

X: X

Y: Y

Z: FID

Min X

Max X

Min Y

Max Y

Min Z

Max Z

Color

Enabled ☑

Column: X

Min: Green

Max: Red

Use linear interpolation ☑

Size

Enabled □

Column: ELEVATION

Min X

Max X

Use linear interpolation ☑

Arrows

Enabled □

X: X

Y: X

Z: X

Various

Sample Rate

Figure 9.5: The menu as it is now

Fig. 9.5 is the design that we chose for our final product. With this design, the user can summon the menu in a separate space where data is not visible, use the menu (and check back on the data while changing settings), and then close the menu and continue observing the data.

Chapter 10

Conclusion

In conclusion, we have identified the Robotics Systems Laboratory's need for a system to visualize data in real time. This need has not been met by existing solutions; they are either too complex to quickly pick up and use, or do not provide interactive and clear visualizations. To satisfy this need, we have designed and implemented vPlot, a software system which uses virtual reality to intuitively visualize data. Using vPlot, members of the RSL are able to interpret important data in real time, without needing to learn complicated or specialized tools. vPlot provides an easy-to-use interface that is simple to learn, yet flexible enough to meet the RSL's needs in examining data. vPlot also provides the ability to visualize data arriving over a network in real time.

10.1 Lessons Learned

Throughout the course of our project, our team learned a lot of lessons. First, there are no best practices for VR UI systems. VR is an emerging field and there is not a set way of doing UI. Our UI was built from our team manually converting Unity's built-in 2D menu components into components that functioned in 3D space. Second, version control systems are quite complex. Understanding the content of each and every file is important in avoiding hard-to-solve version control issues. Third, one should integrate early and often. Make sure to combine different portions of a project before implementing the next feature. If one of the components does not work correctly, fixing the issue before merging it with other components will minimize time and frustration spent on the issue. Fourth, it is important to consult the customer often. Checking in on the customer's thoughts throughout the design and implementation process will lead to a happier customer and an application that is better suiting the needs it was built for. Fifth, leave adequate time to coordinate with other. This is especially the case if another group's technology is being used within your project and there are expectations that this group will be assisting with integration. Respect the time of the other group and give them flexibility on integrating their technology instead of waiting until the last possible minute.

10.2 Future Work

We have identified a number of improvements that could be made to vPlot in the future. First, we would like to improve our representation of vector fields. Our current implementation of vector field visualizations takes a heavy toll on application performance; we cannot display nearly as many points in a vector field as we can for scalar datasets. In addition, vPlot requires that vector data sets take the form of two inputs: a “base point” and an “external point”. vPlot then draws an arrow pointing from the base point to the external point. While functional, this representation is uncommon in real-world data sets; in the future, we would like to support more standard representations of vectors, like Euler angles and quaternions.

We would also like to modify vPlot to allow for more customizability in the control scheme. We currently offer the control scheme that we found to be most intuitive during development and testing; however, there is no user-facing method to modify the bindings of actions to controller buttons. Providing an easy method of rebinding the controls would improve the accessibility of vPlot—users could remap the controls to the scheme that is easiest to use for them. In addition, it would allow users to use different VR (other than the Oculus Rift and HTC Vive) that come with thier own controllers.

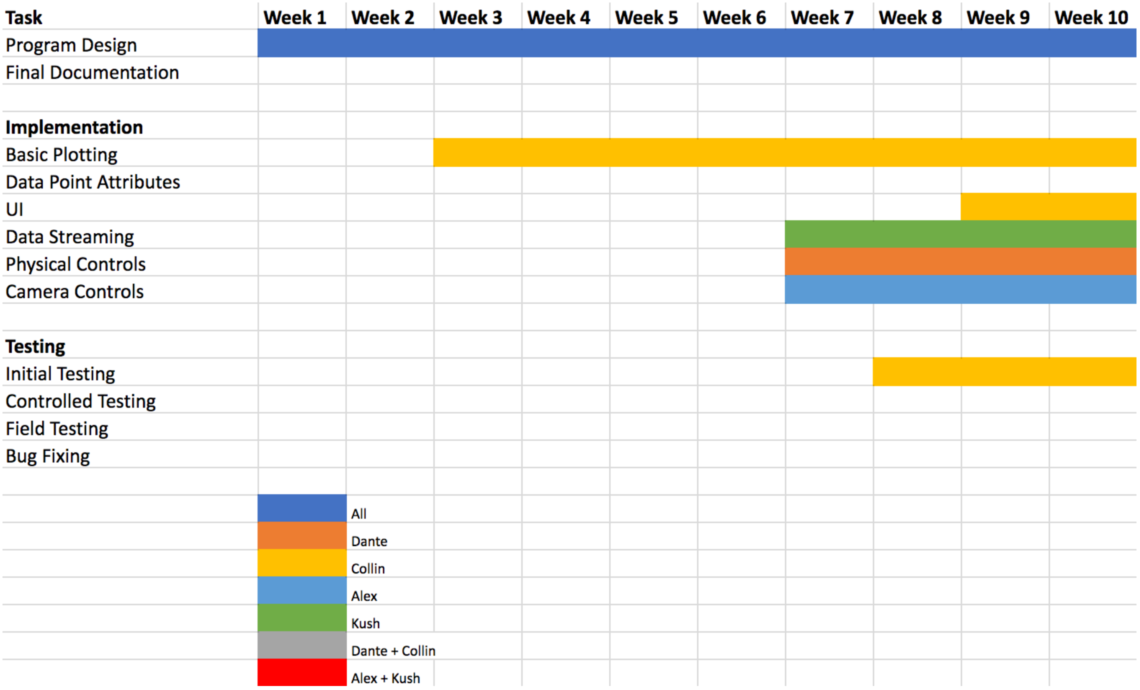
Finally, we would also like to provide greater support for representation of data staleness in vPlot. vPlot does currently support representing time in the plot—in fact, when recieving data from a network source, it will automatically add a timestamp indicating when the data point was recieved. However, time is treated as if it were just any other data feature. We would like to give users greater control over the specific representation of the time feature; for example, we could add an option to automatically shrink or desaturate data points that arrived before a specified point in time.

Appendix A

Timeline

The figures in this section give a timeline of when various major functionalities of the system were developed. We spent most of Fall 2017 on creating preliminary documentation and prototyping a very basic implementation of the system. In Winter 2018, we completed a majority of the implementation of our system. In Spring 2018, we finished the implementation, verified the functionality of our system by running various tests, and wrote our final thesis.

See Fig. A.1, Fig. A.2, and Fig. A.3 for details about the timeline. These figures include information about when major milestones were reached, as well as which team members were responsible for each milestone.



Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
Program Design	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Final Documentation										
Implementation										
Basic Plotting			🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
Data Point Attributes										
UI									🟨	🟨
Data Streaming							🟩	🟩	🟩	🟩
Physical Controls							🟧	🟧	🟧	🟧
Camera Controls							🔷	🔷	🔷	🔷
Testing										
Initial Testing								🟨	🟨	🟨
Controlled Testing										
Field Testing										
Bug Fixing										

🟦	All
🟧	Dante
🟨	Collin
🔷	Alex
🟩	Kush
▨	Dante + Collin
🟥	Alex + Kush

Figure A.1: Fall timeline

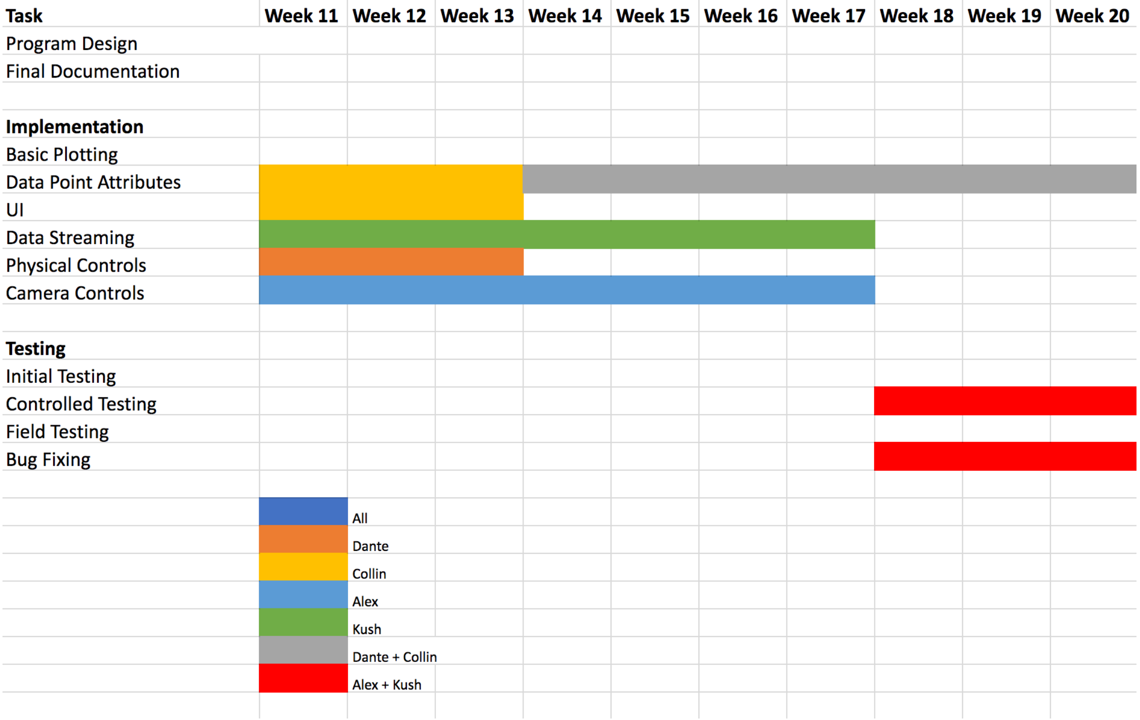


Figure A.2: Winter timeline

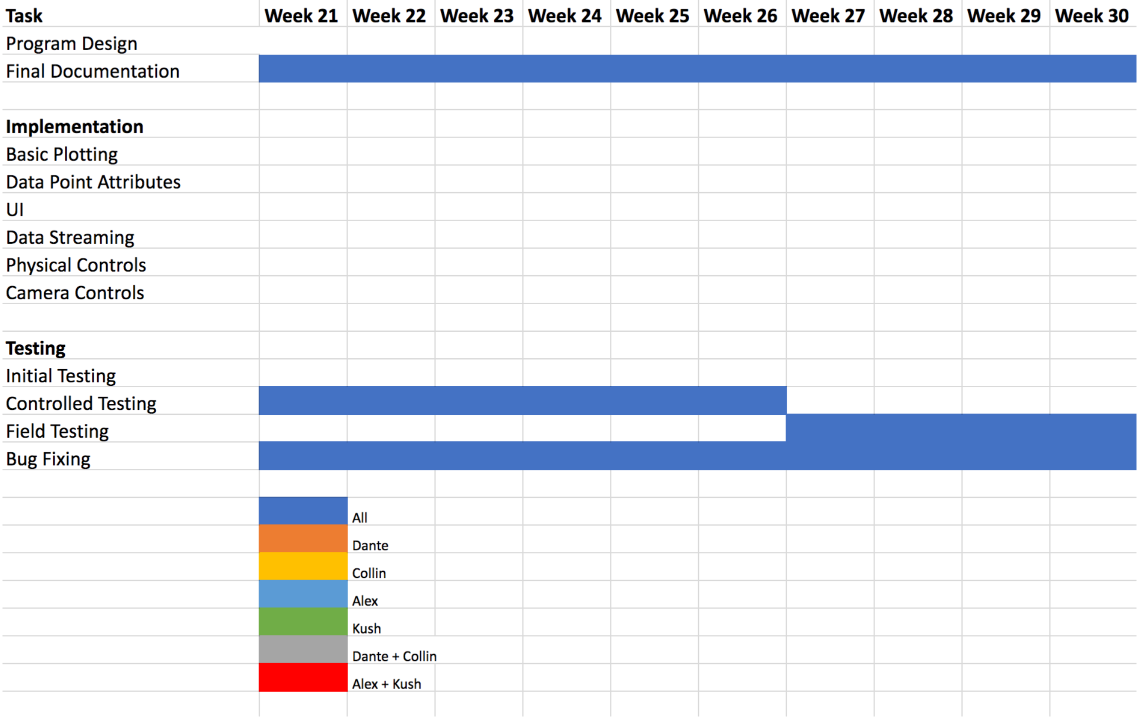


Figure A.3: Spring timeline

Appendix B

Risk Analysis

Knowing the risks associated with completing a functional product is an important part of the design process. Highlighted in Table B.1 are potential risks that could have either hindered completion or affected quality. For each risk, we have formulated consequences associated with the risk, the total impact of the risk, and mitigation strategies to use if the risk was encountered.

Risk Name	Consequence	Probability	Severity	Impact	Mitigation Strategy
Time	Running out of time to implement all that we have planned would cause us to sacrifice crucial features or deliver an unfinished/broken project.	0.8	5	4	1. Find a good place to wrap up. 2. Make a progress report for another team to pick up in the future.
Software bugs	Software bugs could cause our application to behave improperly.	1	3	3	1. Spend time debugging existing code before adding more features.
Loss of work	Loss of work would require us to re-do portions of the project, using time we could have used to implement new features.	0.3	8	2.4	1. Keep a revision history / backup using GitHub. 2. Have redundant configuration management through the RSL SVN Server.
Scope creep	Making our scope too large could overwhelm us and potentially lead to none of our intended functionalities being properly implemented.	0.6	4	2.4	1. Consult with client to re-evaluate our requirements. 2. Make sure critical requirements are met before addressing other features.
Sickness/ member out	Portions of the project will have to be done by less people until member is able to return.	0.9	2	1.8	1. Have each team member work ahead of time so any delays can be made up for later. 2. Document code well so that any module can be completed by another team member with minimal ramp-up time.
Failure of external libraries	Failure of external libraries would lead to a similar outcome had we not been able to afford them: we will have to spend a lot more time reinventing what is contained in those libraries.	0.3	6	1.8	1. Use free/open source libraries as replacements if needed.
Recommended computer cannot run application in VR	We would need to either spend money on upgrading the computer or forfeit features to allow the present computer to handle it.	0.2	6	1.2	1. Prioritize use of funding to provide adequate computer setup. 2. Allow the user to disable non-critical modules that harm performance.
Robots are not available for testing	Receiving data from the robots in the RSL is the main form of testing for the streamed-data portion of our application. If they are unavailable, we will have to find another way to test our application's ability to plot streamed data in real-time.	0.1	7	0.7	1. Simulate data streams with prototype TCP applet.
Insufficient budget	If we do not receive enough money for our project, we may not be able to purchase important Unity libraries. This will result in us spending a lot of time reinventing what is contained in those libraries.	0.2	3	0.6	1. Use free/open source libraries as replacements if needed.

Table B.1: Display of risks associated with vPlot

Appendix C

User Manual

C.1 Installing and Running

We have not compiled cross-platform binaries for our project, so you will need to build the project yourself using Unity. This requires 3 programs to be installed on your computer:

1. Unity 2017.2.0f3 Personal

2. Python 3

3. Git

Once you have installed the above prerequisites, the build process is as follows:

1. Downloading the source code

   (a) Using git, obtain the source code from the repository at https://github.com/collinwalther/vPlot. You may download it wherever is convenient.

2. Building the project

   (a) Navigate to the root directory of the repository on your local machine.

   (b) Use Unity to open Assets/Scenes/StartupScene.unity.

   (c) In Unity, click on File > Build Settings to open the build menu.

   (d) Click ”Build” without changing any other settings to build the project. Save the built project as ”vPlot” in the root directory of the repository.

Once the project is built, follow these steps to run the project:

1. To run the project, there is a startup script written in Python that allows you to easily select the data you would like to visualize. To Use this file, make sure your working directory is the root directory of the project, and then run

```
python3 ExternalUI/main.py
```

A small menu will open, prompting you to select the source of your data:

- If you are viewing data from a .csv file on your machine, select the ”File” radio button, and click on the ”Choose File” button to select the file you would like to view.
- If you are viewing data accessible at a remote server, then select the ”Socket: client” radio button, and enter the IPv4 address and port of the server that you want to connect to.
- If you are viewing data that will be sent to you buy a remote client, then select the ”Socket: server” option, and enter the port on which to listen for incoming connections. Note that you will also need to know your computer’s IP address, so the remote client can connect to you.

2. Click ”Start” to begin the plotting session. At this point, you may put on your VR headset and begin manipulating the data using your VR controllers.

C.2 Using the Application

Once you have your VR headset on and are using the application, the VR controllers allow you to manipulate and move around the data in many ways:

1. Rotate around the data using the left control stick.
2. Zoom in and out using the left and right triggers.
3. Open or close the menu by clicking the Y button.
4. Select and deselect menu items by moving your head to move the reticle over the desired menu item, and then pressing the A button.

Refer to Fig. C.1 and Fig. C.2 for a complete overview of what function each button serves. Once you are finished visualizing the data, exit the application by closing it from your system’s task bar.

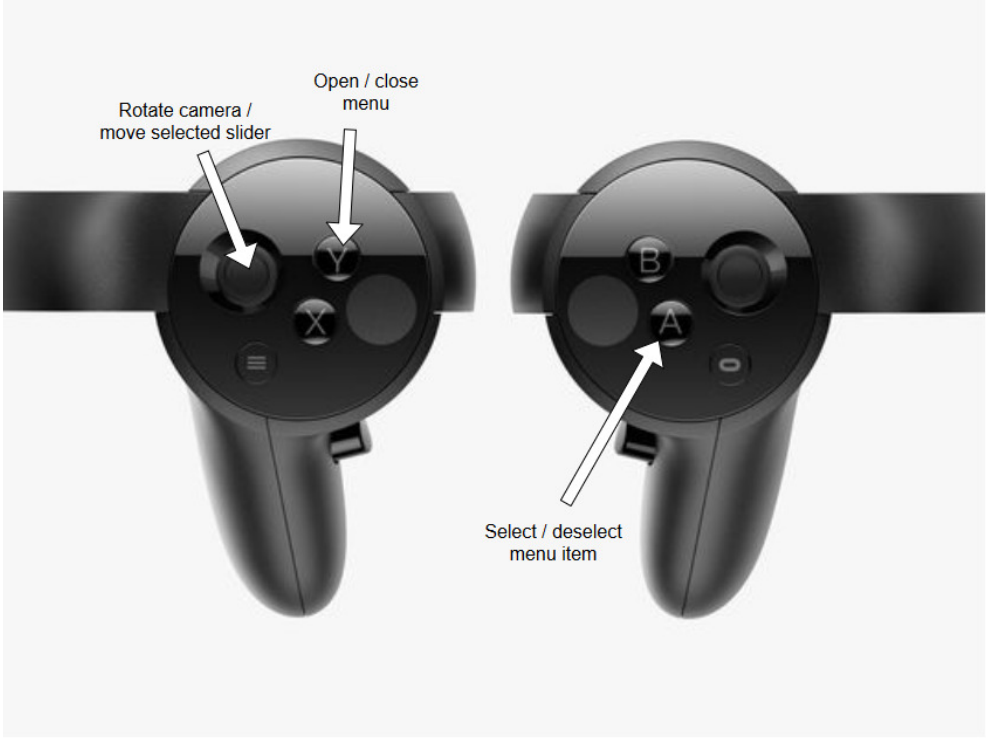


Figure C.1: Overhead view of the Oculus Controller

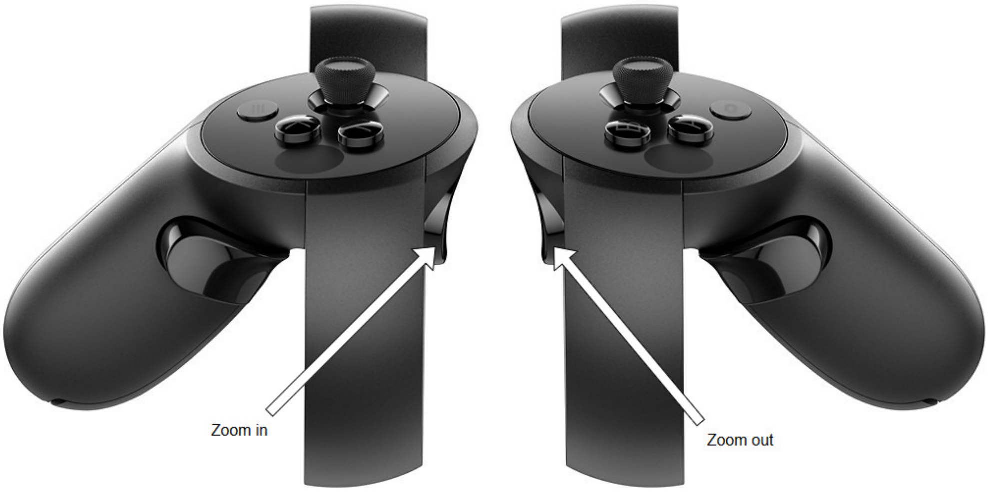


Figure C.2: Side view of the Oculus Controller

Bibliography

[1] C. A. Kitts and I. Mas, “Cluster Space Specification and Control of Mobile Multirobot Systems,” *IEEE/ASME Transactions on Mechatronics*, vol. 14, no. 2, pp. 207–218, April 2009.

[2] “Data Visualization-Wolfram Language Documentation.” [Online]. Available: http://reference.wolfram.com/language/guide/DataVisualization.html

[3] “ListPointPlot3D-Wolfram Language Documentation.” [Online]. Available: http://reference.wolfram.com/language/ref/ListPointPlot3D.html

[4] “Visualization.” [Online]. Available: http://pandas.pydata.org/pandas-docs/stable/visualization.html