UPPSALA
UNIVERSITET

TVE-F 18 014
Examensarbete 15 hp
Juni 2018

Analytics tool for radar data

Hampus Naumanen
Torsten Malmgård
Eystein Waade

Populärvetenskaplig sammanfattning

Analytics tool for radar data var ett projekt som inleddes på grund av brist på smart programvara som kunde analysera komprimerad radardata hos Saab. Tidigare användes program som var utvecklade för andra syften och analysen blev då försämrad och ineffektiv för användaren. Dessa program innebar också en begränsning på vilka typer av radar som kan analyseras, vilket gjorde att de blev mindre tillämpbara som analysverktyg. Lösningen blev att konstruera ett helt nytt program som importerar, tolkar samt visualiserar radardatan oberoende av det radarsystem som inhämtat informationen.

Projektet var i huvudsak uppdelat i två stora delar som kunde arbetas på parallellt: En kod som tolkar och extraherar den komprimerade datan och ett gränssnitt som kan visualisera datan på ett användarvänligt sätt. Båda delarna var programmeringstekniska problem och Java valdes som utvecklingsspråk för applikationen då det innehåller ett stort bibliotek av inbyggda funktioner. Java var också ett språk som var bekant sedan tidigare och inlärningskurvan skulle därför inte bli lika brant jämfört med ett helt nytt språk.

Programmet utvecklades i flera etapper och med flera delar som behandlar olika steg i inläsningen av datan. Ett meddelande från en radar är uppbyggt av flera bitar med olika information, men som sitter ihop i ett långt band, och programmet ska kunna separera detta band så att informationen i bitarna ska kunna läsas. Det kan jämföras som en mening med flera ord, men där programmet ska identifiera och ta ut de relevanta orden som innehåller information om de mål som upptäcks. När programmet tolkat och extraherat den relevanta informationen, som i detta fall är positionen, öppnar den ett fönster där alla objekt som upptäcks under ett radarvarv visualiseras. Programvaran kan då användas för att spela upp flera radarvarv efter varandra som en film där användaren kan spola båda framåt och bakåt. Det finns även en funktion som placerar flera radarvarv på varandra så att det enklare går att identifiera och analysera målspår på skärmen.

Slutligen är programet även uppbyggt på ett sätt som möjliggör att flera funktioner kan läggas till i framtiden och vidareutvecklas för att kunna tillgodose flera behov. Det kan vara allt från att lägga till fler analysfunktioner eller att förbättra prestandan vid stora datamängder.

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Analytics tool for radar data

Hampus Naumanen, Torsten Malmgård, Eystein Waade

Analytics tool for radar data was a project that started when radar specialists at Saab needed to modernize their tools that analyzes binary encoded radar data. Today, the analysis is accomplished using inadequate and ineffective applications not designed for that purpose, and consequently this makes the analysis tedious and more difficult compared to using an appropriate interface. The applications had limitations regarding different radar systems too, which restricted their usage significantly. The solution was to design a new software that imports, translates and visualizes the data independent of the radar system.

The software was developed with several parts that communicates with each other to translate a binary file. A binary file consists of a series of bytes containing the information of the targets and markers separating the revolutions of the radar. The byte stream is split according to the ASTERIX protocol that defines the length of each Data Item and the extracted positional values are stored in arrays. The code is then designed to convert the positional values to cartesian coordinates and plot them on the screen. The software has implemented features such as play, pause, reverse and a plotting history that allows the user to analyze the data in a simple and user-friendly manner.

There are also numerous ways the software could be extended. The code is constructed in such a way that new features can be implemented for additional analytical abilities without affecting the components already designed.

Handledare: Åke Krantz
Ämnesgranskare: Martin Sjödin
Examinator: Maria Strömme
ISSN: 1401-5757, TVE-F 18 014

Contents

1 Introduction **5**
- 1.1 Background . . . 5
- 1.2 Objective . . . 5
- 1.3 Acronyms and Abbreviations . . . 5

2 Theory **6**
- 2.1 Programming languages . . . 6
  - 2.1.1 Java and JavaFX . . . 6
  - 2.1.2 CSS . . . 7
- 2.2 Git . . . 7
- 2.3 Binary data . . . 7
- 2.4 ASTERIX protocol . . . 8
- 2.5 The radar systems . . . 9

3 Implementation **9**
- 3.1 ASTERIX data format . . . 9
  - 3.1.1 Target report message . . . 9
  - 3.1.2 North marker message . . . 11
- 3.2 Programming . . . 12
  - 3.2.1 Reading the binary file . . . 12
  - 3.2.2 Graphical user interface . . . 14

4 Result **14**
- 4.1 The final program . . . 14
- 4.2 Import function . . . 16
- 4.3 Plot function . . . 18
- 4.4 Plot history function . . . 19
- 4.5 Media buttons . . . 19
- 4.6 Time-slider . . . 20

5 Discussion **20**
- 5.1 Evaluation of the software . . . 20
- 5.2 Future development . . . 22

6 Conclusions **23**

Appendices **24**

1 Introduction

1.1 Background

Radars are important tools for air surveillance in civilian and military applications. They operate by sending out a signal and then receiving the same signal as it bounces off a target. The position of the target can then be determined by measuring the time of travel for the signal.

Effective and accountable radars are important as the surveillance of a country's border is of high-priority. To accomplish this all radars used for civilian or military applications needs testing and maintenance to avoid collecting incorrect data. This is done with real-time analysis, but also by analyzing collected data at a later instant. When the data is collected it is saved as a binary file. Saab has struggled to find the proper tool for analyzing the stored data and have used unorthodox methods such as capturing screenshots and browsing through the data point by point. The method used for capturing the screenshots was only available for one of the radar systems which made the analyzis even more restricted. For these reasons an analytical tool needed to be developed.

1.2 Objective

The purpose of this project was to construct a user-friendly software that reads and visualizes binary data from radar systems and simplifies the analysis of the data with functional play-, pause-, rewind- and fast-forward buttons. The software should also implement history plotting and be compatible with three different radar systems as well as being able to import several hours of radar data without loss of performance.

1.3 Acronyms and Abbreviations

ASTERIX	All Purpose Structured EUROCONTROL Surveillance Information Exchange
CAT-number	Category number, what category the message belongs to
GUI	Graphical user interface
JavaFX API	JavaFX Application Programming Interface
RPM	Rotations per minute
TOD	Time of detection

2 Theory

2.1 Programming languages

2.1.1 Java and JavaFX

The software was developed using the Java programming language. Java is a class-based and object-oriented programming language and has a rich variety of functions and libraries, and its simplicity makes it ideal for smaller projects like this. One extensive library is the JavaFX API which consists of several graphics and media packages that can be used to create versatile softwares across different platforms [1]. The ScatterChart class in the JavaFX package is a tool for plotting series of data in a chart[11]. The data series includes a fixed amount of Cartesian coordinates that describes the position of each data point. Figure 1 shows the use of the scatter chart in our project.

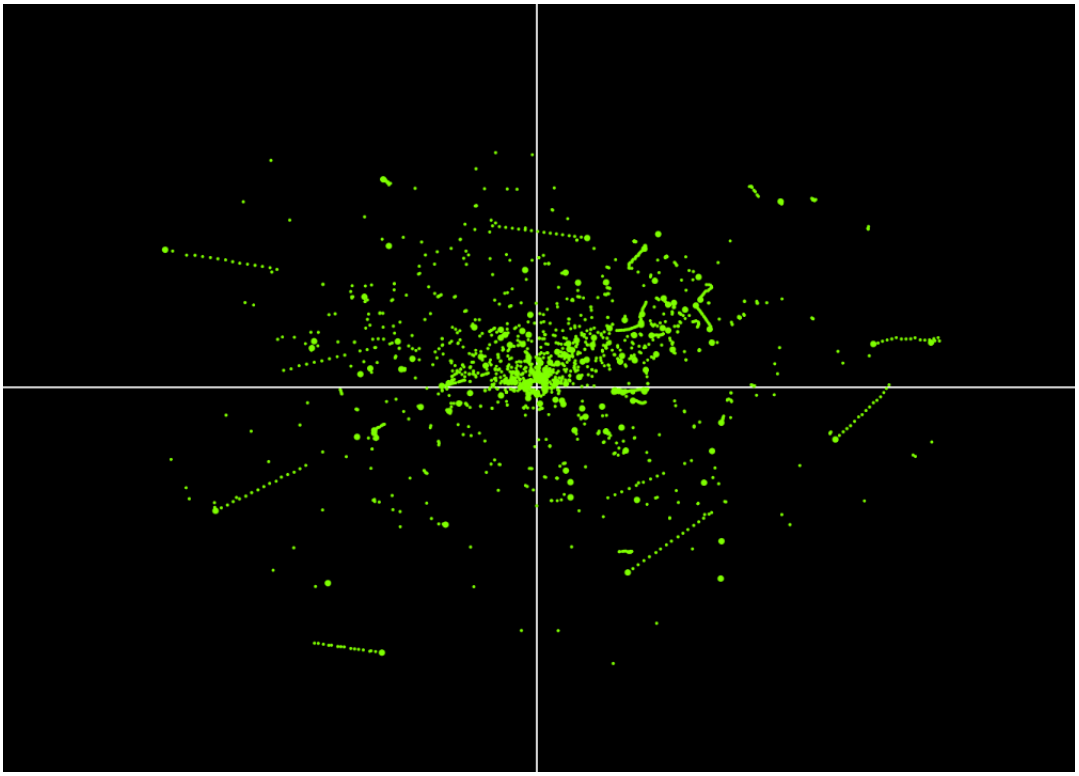


Figure 1: A snapshot of the radar data visualized with a scatter chart.

2.1.2 CSS

CSS is commonly used for graphical representation and visualization in both web pages and software. It allows skinning of programmed components, meaning that a developer can change the appearance of a feature without affecting its functions in any way [2]. CSS is also simple to integrate with JavaFX, and the layout of the software, as well as the integrated functions - such as the scatter chart, could easily be styled to represent data in a user-friendly way. Figure 2 is an example of CSS-styling and the code specifies the color and size of the lead-point, i.e. the largest points in figure 1. The color of the plot is described with a 6 digit hexadecimal code and the size is set to 10 pixels.

```
1   .chart-symbol { /* series (leadplot) */
2       -fx-background-color: #7FFF00;
3       -fx-background-radius: 10px;
4       -fx-padding: 2px;
5   }
```

Figure 2: CSS code example for the lead-points.

2.2 Git

To state the Git homepage: "Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency" [10]. A version control system allows the user to access earlier versions of a certain file or the whole project, and to compare changes over time. Git also stores a log of all the modifications made by users, so the participants are able to see which files have changed and by whom. A version control system is distributed when the files are copied to multiple hosts. A user can commit their local changes to the local repository, and then push those changes to the remote repository. Then the collaborating users sync their local repositories with the remote repository and so forth. GitHub was used as one of the hosts in the distributed system, with Git being the version control system used in this project.

2.3 Binary data

A bit is the smallest unit of information that is processed by computers. It has one of two values, usually represented by a 1 or a 0. The bit can also

be represented by a physical object such as a lever, button or light. The common idea is two states, where one state represents TRUE, and the other representing FALSE. In the early days of computers, data was transfered through hole punch cards, where one bit was represented by a spot on the card having a hole or not[4]. Several bits together can form binary numbers. The standard for transmitting bits are in octets, called bytes, where one byte consists of 8 bits. Figure 3 shows an octet structure from the ASTERIX protocol. Every bit in the octet, or a combination of them, could be used to store information, and by using several bytes, more complex data can be represented.

There are two ways to read binary data, either from left to right, or from right to left. This is commonly known as Big Endian and Little Endian. Big Endian specifies that the least significant bit is to the right, which is the standard way of reading numbers in the western world. Little Endian is where the least significant bit is to the left.

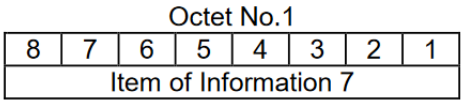


Figure 3: Example byte from the ASTERIX protocol.

2.4 ASTERIX protocol

ASTERIX is a standardized protocol for transmitting and translating surveillance data. The protocol is made and maintained by EUROCONTROL, a collaboration between 41 countries in Europe whose goal is to maintain a secure and effective air surveillance. The protocol is designed for systems with limited bandwidth meaning that the information is stored with as few bits as possible. This allows transitions of large quantities of information in small files.

A target report is a message with a fixed amount of Data Items. A Data Item is a combination of one or more bytes, with the purpose of giving a specific information in the target report. There are two ways the Data Items are translated: Either into a decimal number, e.g. the azimuth or the range from the radar, or a bit containing the information itself, e.g. 1=civilian and 0=military. The Data Items are always sorted in a specific order according to its reference number. The protocol has a a description for each reference number that explains the information included in every Data Item.[6].

The ASTERIX protocol consists of different categories specified in the

beginning of each target report. They specify the structure of the target report and how to interpret it. Category 048 indicates that the target report contains radar data and category 034 specifies the structure of the north marker service message.[3]

2.5 The radar systems

The three radar systems that required compatibility with the software was the Swedish Armed Forces Surveillance radar 640, Surveillance radar 861 and Radar system 871. The 640 system is a coastal radar used for detecting ships but can also detect low flying aircraft. It can detect targets up to a range of 50 km and has a angular velocity of 12-24 RPM adjusted by the operator [9]. This system also produces the shortest messages of 18 bytes. The 871 radar is a low height radar with a range of 100 km. It is mounted near the coast to detect threats both in the air and on water and produces messages of between 48 and 50 bytes[8]. Among the three radars, the 861 system has the longest range of up to 300 km and it also registers height of an encountered target. The target reports of the 861 has a length of 34 bytes [7]. Both the 861 and 871 system has an angular velocity of 6 RPM .

3 Implementation

3.1 ASTERIX data format

3.1.1 Target report message

The first task when implementing the software was decoding the binary raw files produced by the radar. When the radar detects a target it sends a target report of category 048 as described above. The message sent can be translated by following the standard documentation released by Eurocontrol. Messages sent by other radar systems could vary in length since some radar system gathers different information about their targets.

The first three bytes in category 048 are always the same, where the CAT-number is represented by the first two bytes, and the length of the message is represented by the third byte. Data Item 048/040, which represents the polar coordinates of the target, are stored in byte 11 through 14 for the radar system 640, and 13 through 16 for the systems 861 and 871. This is why information about the length of the message is important, it does not only convey what bytes need to be read, but differentiates between messages from different radar systems. Figure 4 is an example of a category 048 message

produced by the 871 radar system after translation. On row 13 the 048/040 Data Item is displayed.

To convert the binary data to decimal numbers it was first established that every message is encoded using Big Endian. Then, since it was only relevant to translate the positional values in the project, a code was written to extract Data Item 048/040. The Data Item contains four bytes, where the first two describes the range, i.e. radius, and the last two describes the azimuth angle. The least significant bit for the range has a value of $\frac{1}{256}$ nautical miles and the least significant bit for the azimuth has a different value of $\frac{360^\circ}{2^{16}}$. This means that the translated decimal values of the range and azimuth must be multiplied by the above values respectively. The range also has to be converted from nautical miles to kilometers by multiplying it by 1.852.

The last step is to convert the polar coordinates to Cartesian coordinates. The angle α is defined clockwise from the geographical north and is the angle convention in the target report. The transformation from polar to Cartesian coordinates is shown in equation 1 and 2 below.

$$x = r * sin(\alpha) \tag{1}$$

$$y = r * cos(\alpha) \tag{2}$$

where α is the azimuth and r is the range. The code then carries out these operations and a method is defined as getCartesian which returns these specific x and y values.

```
------------------------------------------------------------
Frame        2;   Length: 48

Block        2
CAT        048
LEN         48

1. Record of Block 2 (2. Record absolute)

   I048/010[SAC: 099; SIC: 100]
   I048/020[TYP: 1 (Single PSR); ACT; RDP1; DEF; DEF]
   I048/040[RangeCentroid: 13.405 km;
            AzimuthCentroid: 84.858 deg]
   I048/130[PRI Azimuth extent: 1.934 deg;
            PRI Max Magnitude: 39.0 dBm]
   I048/SP [Range extent: 0.05 meter;
            Signal to Noise ratio: 11.50 dB;
            Blip to Scan ratio: 0;
            Average Magnitude: 35.50 dB;
            RadarCrossSection: -2.409 dBsm;
            No. of CPIs with Async.Interf.: 0;
            No. of hits with Async.Interf.: 0;
            Scan mode: 1 (Normal);
            Number of Hits in High Beam: 9;
            Number of Hits in Low Beam : 0;
            Best estimated Radial Velocity: -0.10 m/s;
            Second best estimated Radial Velocity: -0.10 m/s;
            Plot in Notch Filter (1-8)   : 0 0 0 0 0 0 0 0;
            Plot in Notch Filter (9-16)  : 0 0 0 0 0 0 0 0;
            Plot in Notch Filter (17-24) : 0 0 0 0 0 0 0 0;
            Plot in Notch Filter (25-30) : 0 0 0 0 0 0]

   f3 01 01 04 00 00 00 00 00 20 07 3d 3c 58 18 2c
   27 1c ff f8 00 02 17 00 00 47 f6 97 00 00 00 00
   01 00 09 00 00 ff ff ff ff 00 00 00 00
------------------------------------------------------------
```

Figure 4: Category 048 target report example.

3.1.2 North marker message

A north marker message, category 034, is different to the 048 message since it is not an actual target report, but instead a service message generated at every north crossing. It is differentiated from the 048 message at the CAT-value in the first byte and hence the code was built to sort and store it

separately in an array when such a message is discovered. The North marker, practically understood as when the radar has made one full revolution around its own axis, is used as a reference so that only targets between two north markers are plotted at a time in the scatter chart. Figure 5 shows that the north marker message contains less information than a category 048 message.

```
Frame       51;    Length: 14

Block       51
CAT        034
LEN         14

1. Record of Block 51 (51. Record absolute)

   I034/010[SAC: 099; SIC: 100]
   I034/000[Message Type: 1 (North Marker)]
   I034/070[REP: 1
            TYP: 2 (Single SSR): 89]

   e1 80 00 00 00 00 00 a9 01 10 59
-------------------------------------------------------------
```

Figure 5: North marker target report.

3.2 Programming

3.2.1 Reading the binary file

Figure 6 and 7 shows that the binary data is split into messages. It was possible to extract the specific length of the message by marking the position in the byte stream and saving the information in a vector with three bytes. The class then reads the first three bytes in each message, containing both the CAT-number and the length of the message, and then sorts the data properly. The split and labeled data is stored as objects of type DataPoint which in turn is stored in an array.

When the GUI class is set to plot a frame it reads the set of DataPoints and picks out each DataPoint from the array in the ReadBinary class. It starts from the designated north marker and moves on to the next, and calls on a method named getCartecian in each DataPoint. This method takes the specified bytes and converts them from binary to decimal cartesian coordinates. These are stored in a series object that can be plotted using ScatterChart.

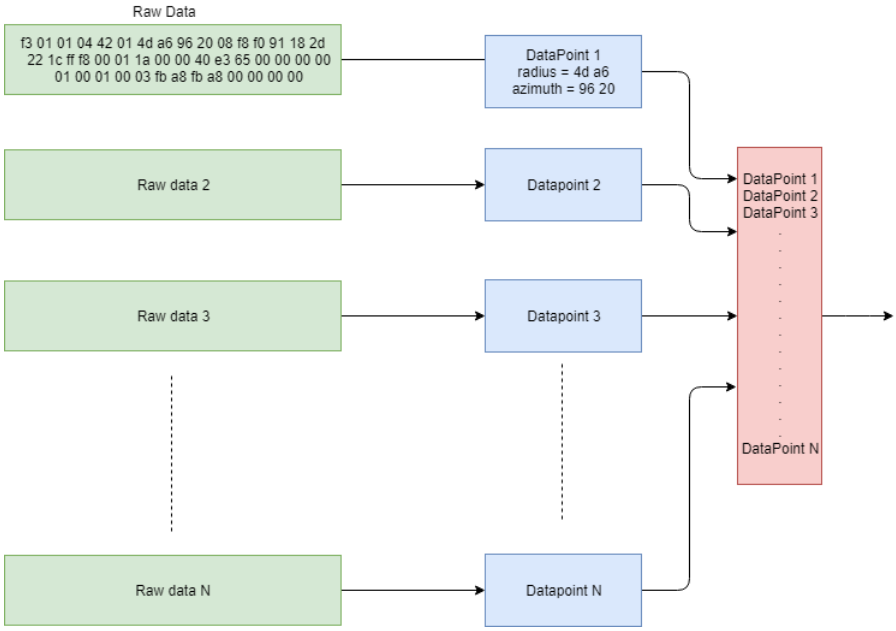


Figure 6: Schematic: Raw data to DataPoints

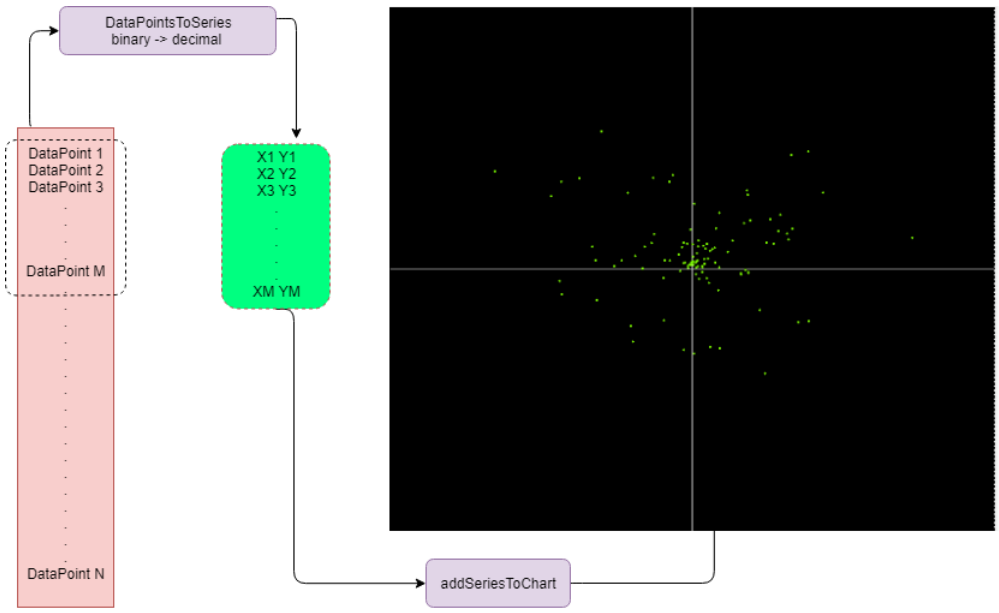


Figure 7: Schematic: DataPoints to plot

3.2.2 Graphical user interface

The GUI is based upon the built-in class in JavaFX called BorderPane. It seperates the scene in five different parts as shown in figure 8. The menu bar was first constructed with a File and Edit item placed at the top of the BorderPane, and then a drop-down menu was constructed for both items. The import function was then implemented with an ActionEvent, which is user dependent, combined with the built-in class FileChooser which prompts the software to display a file open dialog. In the center pane the scatter chart was added with a StackPane for more functionality, should the software need a zoom function for example. At the bottom pane, a VBox and HBox was constructed to lay out the buttons and sliders in a vertical and horizontal row.

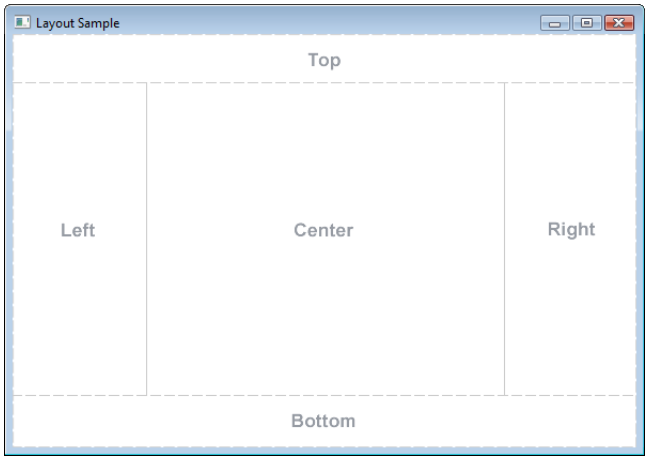


Figure 8: Schematic view of the a scene with a BorderPane layout [13].

4 Result

4.1 The final program

When the software is started the interface is white with a menu bar visible at the top, see figure 9. The import button allows the user to select a binary raw file, and when a file is chosen, the GUI-class creates an object of type ReadBinary with it. Figure 10 shows the class architecture of the software. The ReadBinary object takes the byte stream in the binary raw file and splits it in separate target reports and stores the messages in objects of type DataPoint in an array. ReadBinary also detects north marker messages and stores the necessary information. The GUI class then creates an extended

interface containing the scatter chart, playback buttons, timeline- and history slider. The GUI then takes the data from the ReadBinary class and transfers them to a new array. This stops when encountering a north marker. The data is then plotted and the program sleeps for a set amount of time before it clears the chart and plots the data between the previous and the next north marker. This is visualized in figure 11.

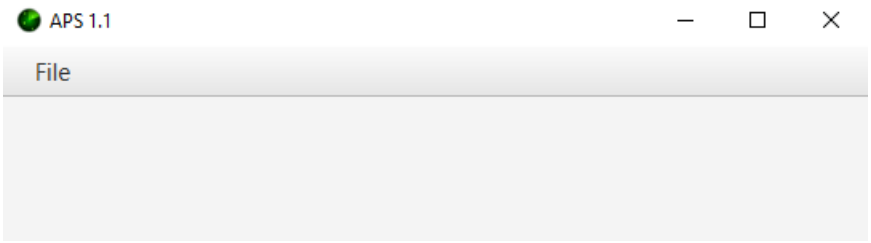


Figure 9: Interface of the software when launched.

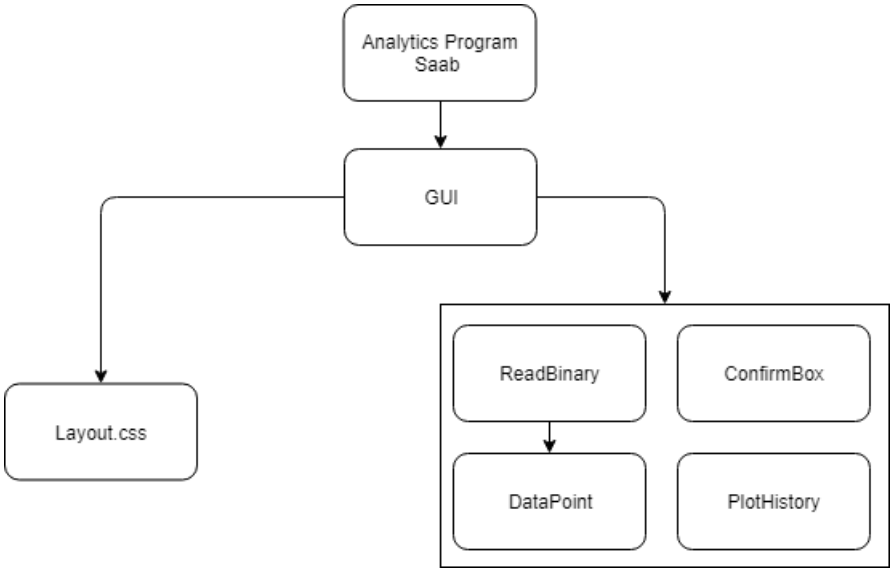


Figure 10: Class Architecture of the APS

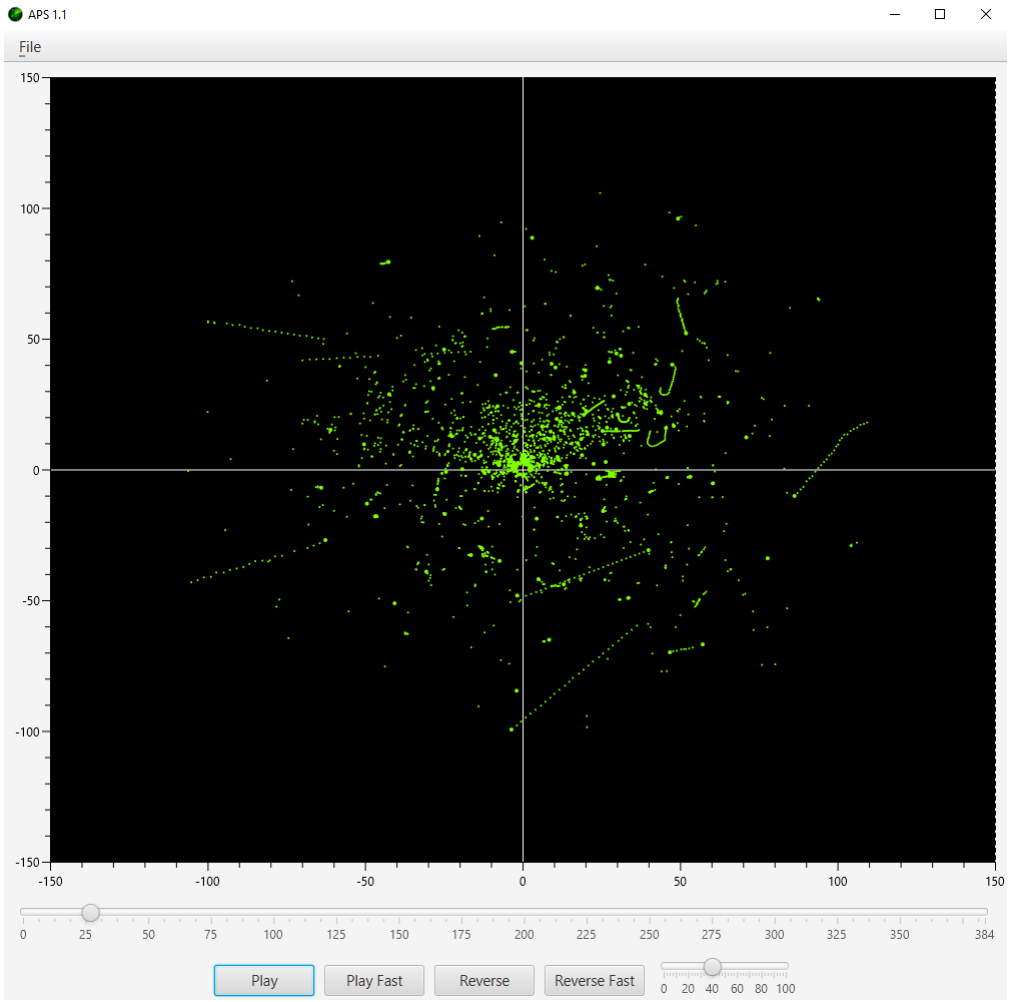


Figure 11: Snapshot of the final application while running.

4.2 Import function

When the File menu item is pressed a drop-down menu appears where the user can import data into the software, see figure 12. When the import button is pressed a file-chooser window opens where binary raw files are listed and the user is able to choose which file to import, see figure 13. When the user chooses a file it is imported into the software and a scatter chart is constructed. The scatter chart can be interacted with by pushing the media buttons located below, see figure 14.

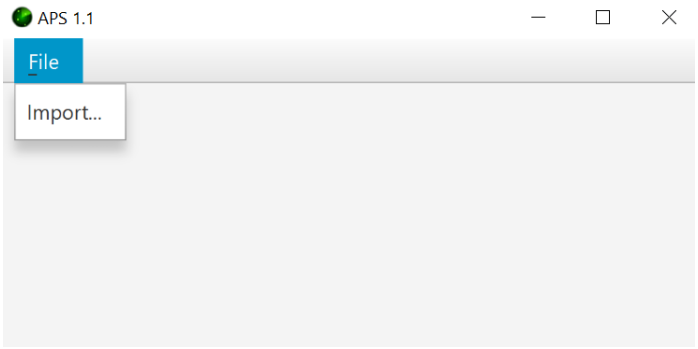


Figure 12: File menu interface.

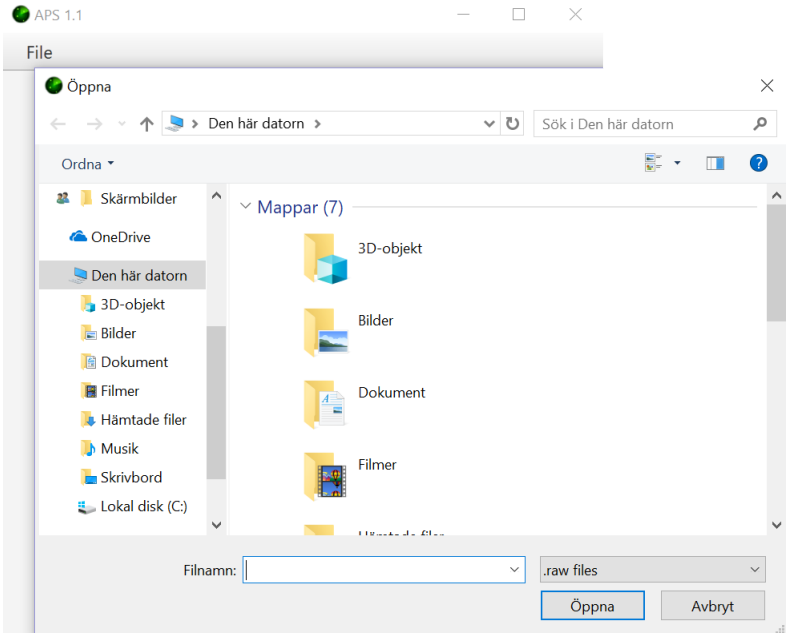


Figure 13: File import menu interface.

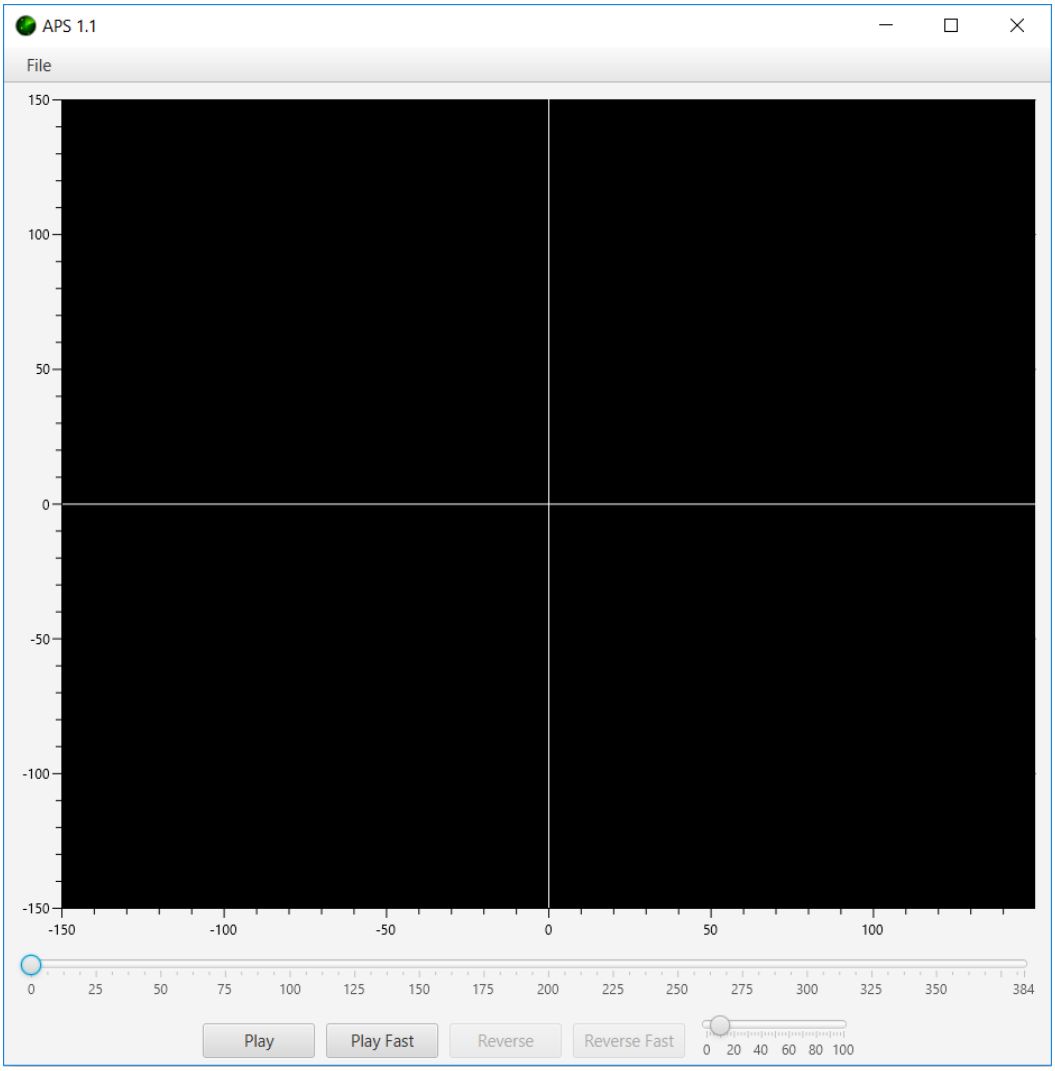


Figure 14: Final screen when a file is loaded and ready for analysis.

4.3 Plot function

When the data is processed and the scatter chart is created the software reads the coordinates from the Data Point object stored between each northmarker and plots them. The speed at which the chart is updated is decided by the buttons below the chart, see paragraph 4.5.

4.4 Plot history function

The plot history function makes the user able to see the previous locations of the plots. A slider enables the user to choose a history between 0 and 100 units. One unit is equal to $\frac{1}{RPM}$, which means that a plot history of 6 units is equal to one minute for a radar with 6 RPM. The plot history function makes it easier to see patterns in target paths and the radar data. Figure 15 shows the airspace picture with and without plot history at a specific time.

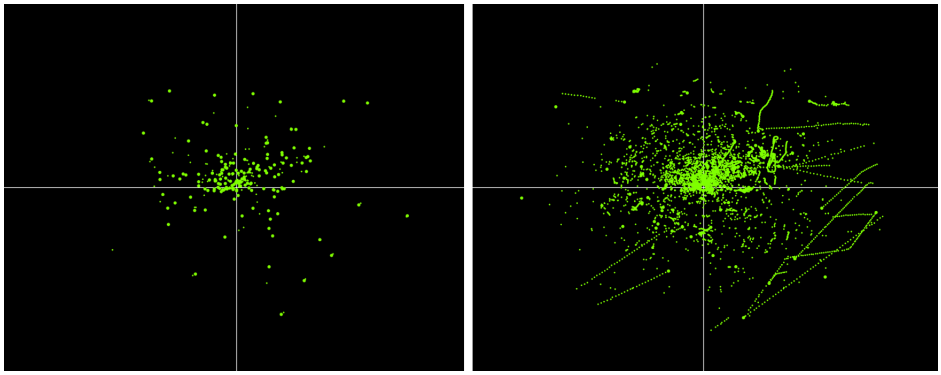
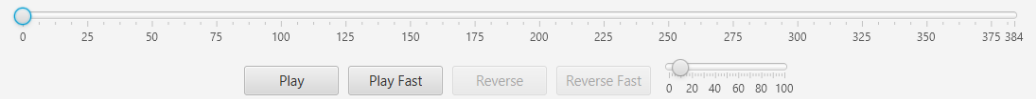


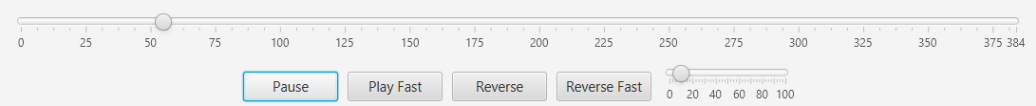
Figure 15: Comparison of the scatter chart with and without history.

4.5 Media buttons

The software features four buttons for analyzing the data. A play button, fast forward button, reverse button and fast reverse button. When the software starts and a file is loaded, both reverse buttons are disabled, since reverse is not possible at the start. This is seen in figure 16a. When the user clicks on either the Play or Play Fast button, the software starts displaying data at the desired frame speed. At the same time the Play button changes text to "Pause" and the reverse buttons is enabled since reversing now is possible. The program continues to plot the frames one by one and the user can, at anytime, pause the plotting, reverse it or make it go faster. This mode is visualized in figure 16b. When the last frame has been displayed the software stops and the two play buttons are disabled so only reverse is possible, this can be seen in figure 16c. Since the "Play Fast", "Reverse" and "Reverse Fast" buttons are togglebuttons, they are indented when active. The "Play/Pause" button is not since it still has a function while being active. The three togglebuttons are all part of a so called togglegroup. This is a built in function of in Java, that prevents two or more buttons to be active at the same time.



(a) Stage 1: at the start.



(b) Stage 2: while playing.

(c) Stage 3: at the end.

Figure 16: Close up on the controls layout in three stages

4.6 Time-slider

Below the chart is the time slider. This slider is labeled according to the position of the north marker messages in the bytestream. The slider updates for each frame and thus works as a slider on a media-playing device. The slider also has a method for updating the internal clock which gives the user the ability to drag the slider and examine any chosen part of the data.

5 Discussion

5.1 Evaluation of the software

The aim was to create a software that imports radar data in binary form, translates it and presents it in a user-friendly way. The software was supposed to implement fast-forward and playback features, as well as a history plot. In the final version, all of these functions were implemented and works as intended. The software also contains features that was not intended but was easy to build and gave the application more functionality and the user more analytical tools. For example, the history plot became a slider where the user can choose how much history they want, instead of only choosing between history and not. A time-slider was also added where the user is able to jump between points of time in the data. That gives the user a more flexible way to analyze data, and especially saves lots of time and effort when importing several hours of data. As seen in the examples presented in figure 17, the program can easily be used to analyze the airspace. One can quickly

see an area on the screen where several planes seemed to take-off and enter. Matching the coordinates of the radar with a map and measuring out where the area should be, gave the conclusion that it was a nearby airport.

One improvement that could be done is to open the scatter chart when the program is launched. That would give the application a proper look as soon as the program is launched instead of after the raw-file has been loaded.

The scatter chart also has performance issues when plot history and play speed is at a maximum. The CPU power required to run the program often causes it to crash or behave unexpectedly, especially at the endpoints of the data. The performance issue is probably a consequence of inefficient coding and could be solved with more troubleshooting. The chart also resizes with the softwares window. While this is desired in many applications, resizing the scatter chart and removing the symmetry of the original square could lead to a false interpretation of the data as targets along the x-axis would appear further apart than targets on the y-axis or vice versa.

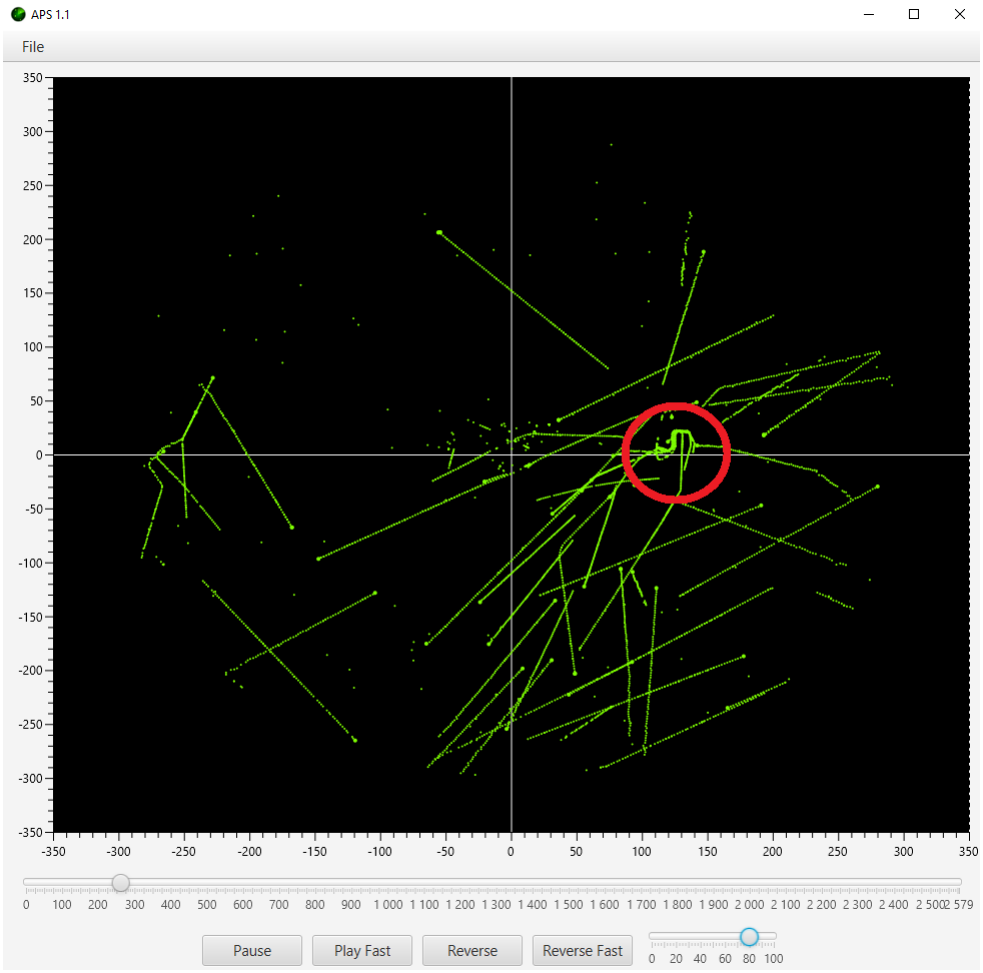


Figure 17: Snapshot of the program with a marking where a nearby airport is clearly visible.

5.2 Future development

The program has been developed so that it is easy to save more information in each DataPoint if needed. The current software saves the bytes representing the position of the actual track in cylindrical coordinates. Radial velocity, signal strength and time of detection are other examples of information stored in the file. This information can easily be added to each DataPoint if wanted. This information can then be extracted as decimal numbers by constructing a unique method for each variable.

The internal clock system uses the north markers as reference points. This means that one time step is equal to the rotation time. A feature that would improve the flexibility of the software would be to include the TOD, time of

detection, in DataPoint and use it as a reference for the internal clock. This would allow the user to update the picture several times during one rotation. Another advantage is that it would be possible to visualize the time stamp of the current data while the software is running.

New features could also be added. The history slider would be more user friendly if it had buttons on each side that increased or decreased the number of history points plotted. Another feature that would make analysis of specific frames easier is buttons that does not play the data but jump forward one frame at a time. This would enable the user to thoroughly examine every frame in the data.

6 Conclusions

The program has the required functions and visualizes the radar data as intended. Play-, pause-, rewind- and fast-forward buttons, time- and history sliders are logical to use, though the actual time is not visualized and the plot history bar could have had a higher resolution. Additional functions can be added with minor adjustments and development of the software.

Appendices

Eurocontrol standard document for surveillance data exchange part 4: category 048
https://www.eurocontrol.int/sites/default/files/content/documents/nm/asterix/archives/asterix-cat048-monoradar-target-reports-next-version-of-cat-001-part-4-v1.19-032011.pdf
Eurocontrol standard document for surveillance data exchange part 2b
https://www.eurocontrol.int/sites/default/files/content/documents/nm/asterix/archives/asterix-cat034-monoradar-service-messages\-next-version-of-cat-002part-2b-v1.26-112000.pdf

References

[1] Pawlan, Monica (2013). *What is JavaFX?*. [online] Oracle docs, Available at: https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm [Accessed 23 May 2018]

[2] W3C, (2016). HTML & CSS. [online] Available at: https://www.w3.org/standards/webdesign/htmlcss [Accessed 23 May 2018]

[3] European Organisation for the Safety of Air Navigation, (2011). *Surveillance Data Exchange - Part 4 Transmission of Monoradar Target Reports.*

[4] Nationalencyklopedin, hålkort. http://www.ne.se/uppslagsverk/encyklopedi/lång/hålkort (Accesed 25 May 2018)

[5] European Organisation for the Safety of Air Navigation, (2007). *Surveillance Data Exchange - Part 2b Transmission of Monoradar Service Messages.*

[6] EOUROCONTROL: All-purpose structured EUROCONTROL surveillance information exchange (ASTERIX). http://www.eurocontrol.int/services/asterix (Accesed 25 May 2018)

[7] Swedish Armed Forces. [online] Available at: https://www.forsvarsmakten.se/sv/information-och-fakta/materiel-och-teknik/ovrig-materiel-sensorer/spaningsradar-861/ [Accessed 24 May 2018]

[8] Swedish Armed Forces. [online] Available at: https://www.forsvarsmakten.se/sv/information-och-fakta/materiel-och-teknik/ovrig-materiel-sensorer/spaningsradar-870/ [Accessed 24 May 2018]

[9] Swedish Armed Forces. [online] Available at: https://www.forsvarsmakten.se/sv/information-och-fakta/materiel-och-teknik/ovrig-materiel-sensorer/spaningsradar-640/ [Accessed 24 May 2018]

[10] Git. [online] Available at: https://git-scm.com/ [Accessed 24 May 2018]

[11] Java ScatterChart documentation. [online] Available at: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/chart/ScatterChart.html [Accessed 24 May 2018]

[12] Java Scene documentation. [online] Available at: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Scene.html [Accessed 24 May 2018]

[13] Java BorderPane documention. [online] Available at: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/BorderPane.html [Accessed 25 May 2018]