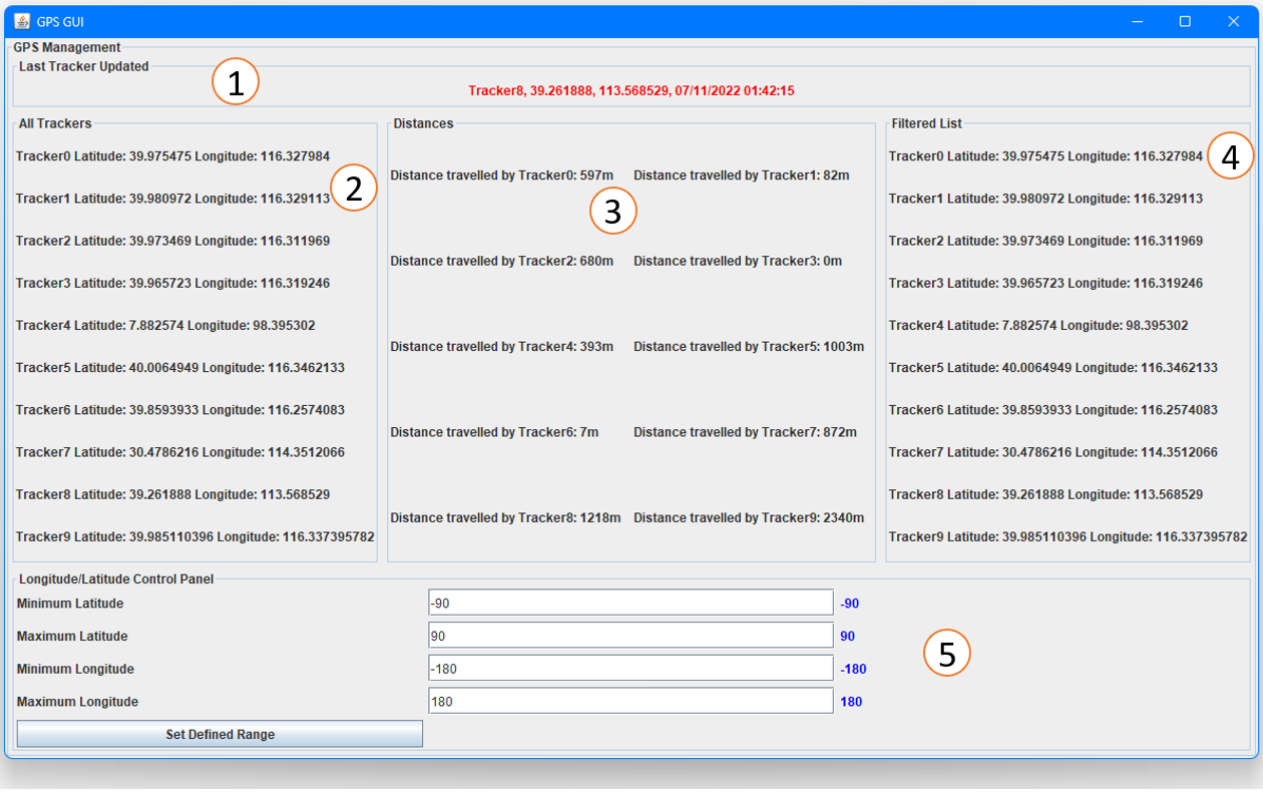
Testing Report for EDC Assignment 3:

How to start:

1. Run the command to compile
    a. javac -cp ".;lib/sodium.jar;lib/swidgets.jar" GpsGUI.java GpsService.java GpsEvent.java
2. Run the command to run the application
    a. java -cp ".;lib/sodium.jar;lib/swidgets.jar" GpsGUI

Testing Report:



The user interface of the GPS GUI is divided into 5 distinct sections. According to the Expected GUI Display/ Output Format, the following are the points and the corresponding sections:

1. [SECTION 2] Ten tracker simplified displays. Simplified tracking data has its altitude data removed and this should be carried out with Sodium FRP operations. From this stream, display, in separate cells:
    - Tracker number
    - latitude

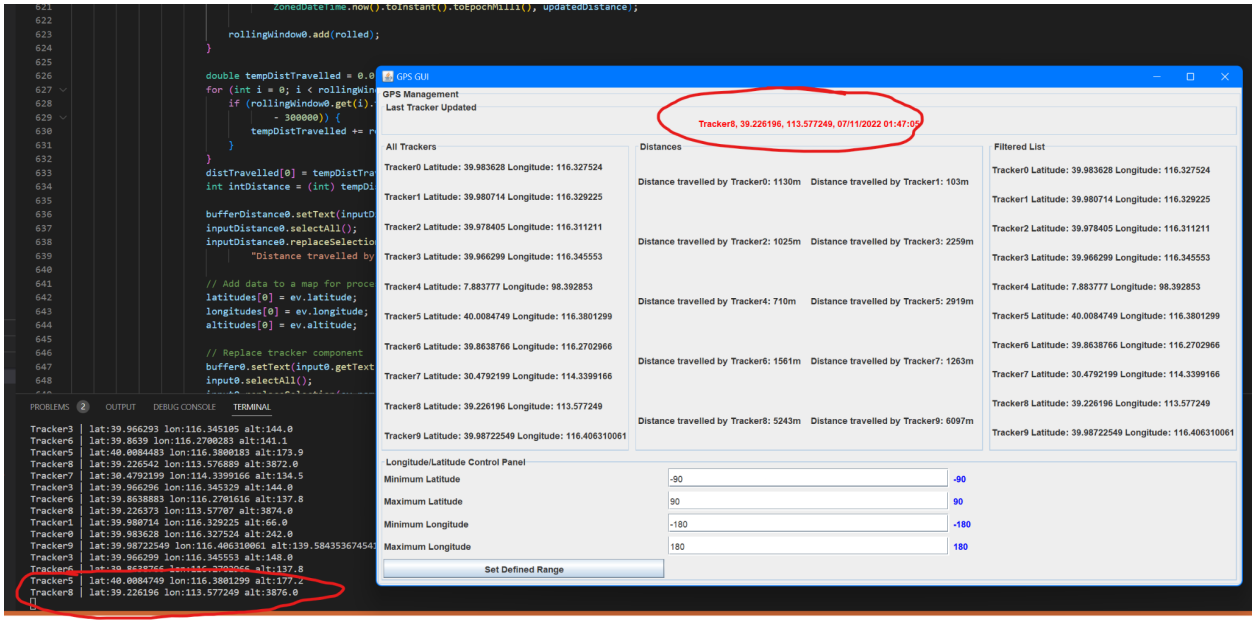
    - longitude
2. **[SECTION 1]** A display field shows each event as it is passed to the GUI, at the time it occurs.
    - This is presented as a comma-delimited string of 4 items
    - Only a single entry is presented at a time, and cleared after 3 seconds if not overwritten.
3. **[SECTION 4 & 5]** A single (1) display that combines all input streams, and only outputs GPS events in the defined range of Latitude and Longitude.
    - The defined range is controlled by a control panel consisting of a latitude input field, a longitude input field, and a button to set the restriction.
        - The latitude and longitude restriction will probably require the use of the **snapshot** primitive.
    - Labels show the current setting of the restriction.
    - The GUI element displays the data in an identical form to the display field in (2), as long as the data is in the range.
4. **[SECTION 3 & 5]** A distance field for each tracker that outputs the total distance travelled over the last 5 minutes for each tracker.
    - Your GUI should display a label for each tracker, with the current distance travelled next to it.
    - This only includes events that are within the currently set Lat/Long range.
    - The value should be a distance in meters, rounded up to the nearest integer meter.
        - **Important: The tracker altitude data is in feet. You should use map to transform this data.**
    - Distance calculations may be calculated without considering the curvature of the Earth but must take the altitude into account.
      (See Latitude and Longitude section below)
    - **Hint:** You might want to look at the FRP primitive snapshot.
    - **Hint:** A sliding window of 5 minutes, however you implement it, will make this calculation easier. You only calculate distances between known positions, i.e. two distinct events

Your GUI should use framing and clear structure to group similar elements and make it easy to understand what is going on.

Section 1 Last Tracker Updated (Point 2)

This section shows a display field of each event that happens as a comma-delimited string of 4 items that is cleared after 3 seconds if not overwritten.

Testing this section was just corresponding to the last events that were printed in the terminal with the tracker that was showing as the ‘last event’. This test shows that it works:

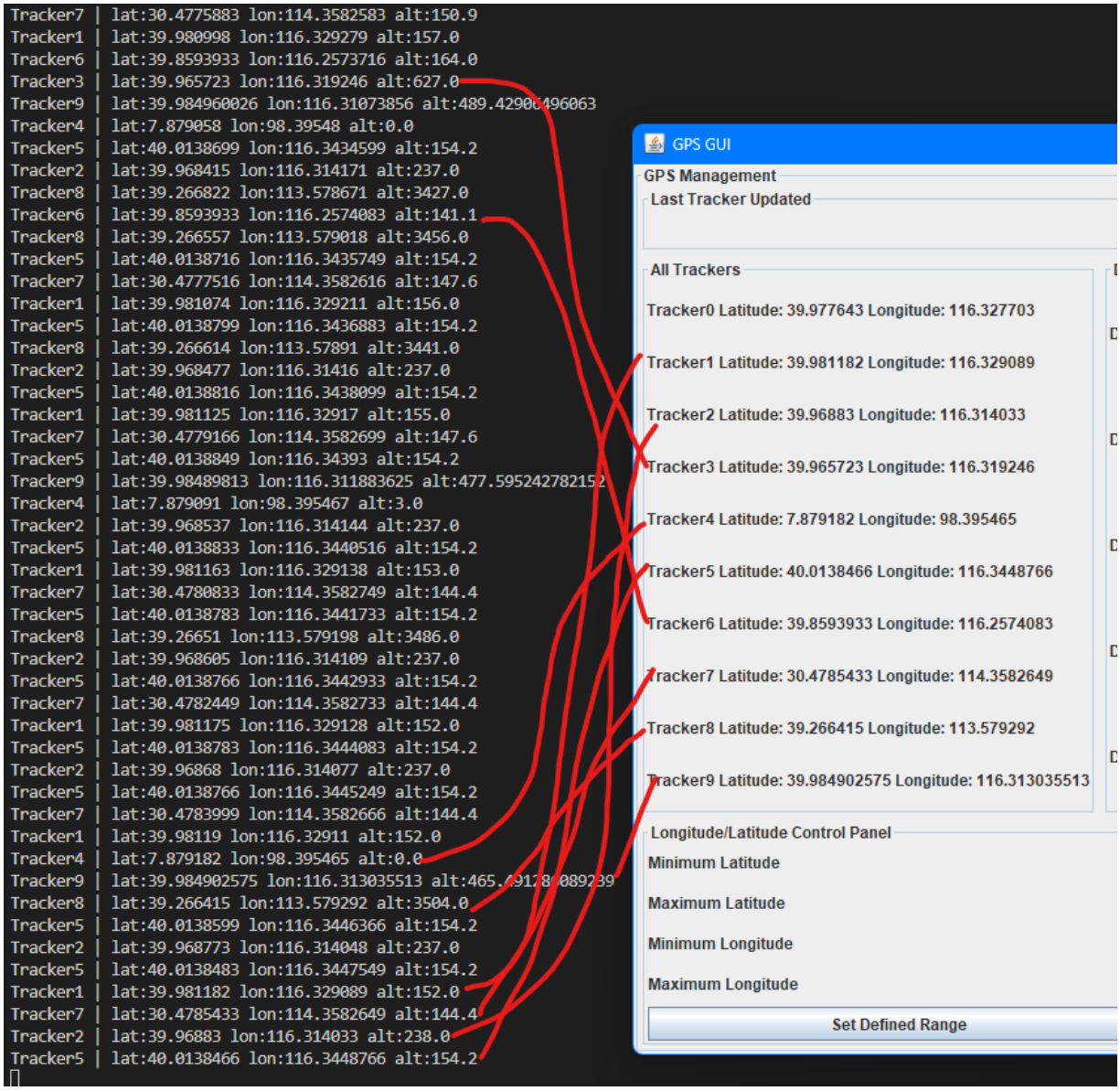


We can inspect that every 3 seconds if not updated it will clear itself by placing a print statement and modifying the code in the loop to stop events from happening. From this it has been verified that the timer every 3 seconds works.

Section 2 All Trackers (Point 1)

This section shows ten simplified tracker displays. The altitude of this data has been removed with Sodium FRP operations.

Testing this section was just corresponding to the events that were printed in the terminal with the tracker that was showing as the ‘last event’. This test shows that it works:



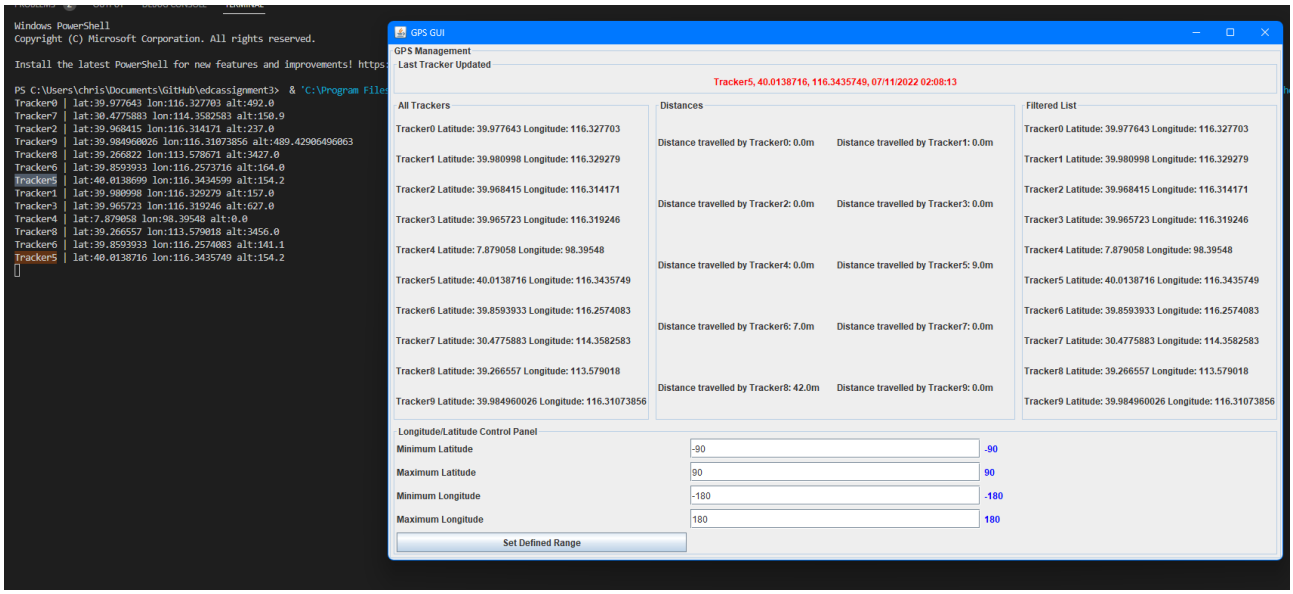
Section 3 Distances (Point 4)

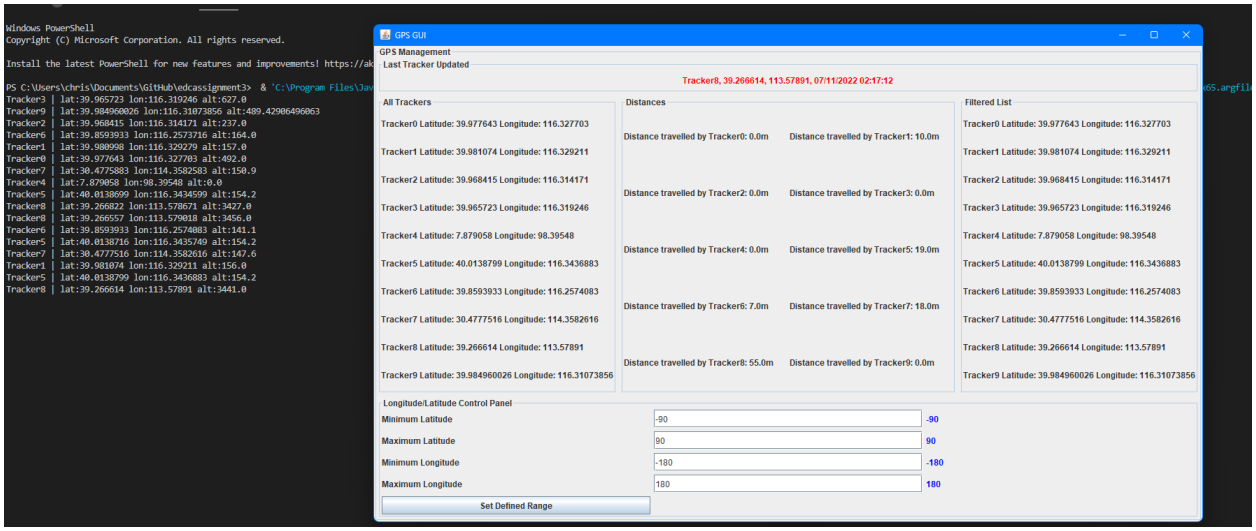
This section shows ten tracker labels with their corresponding distance traveled.

It only includes events that are within the Lat/Long range from Section 5.

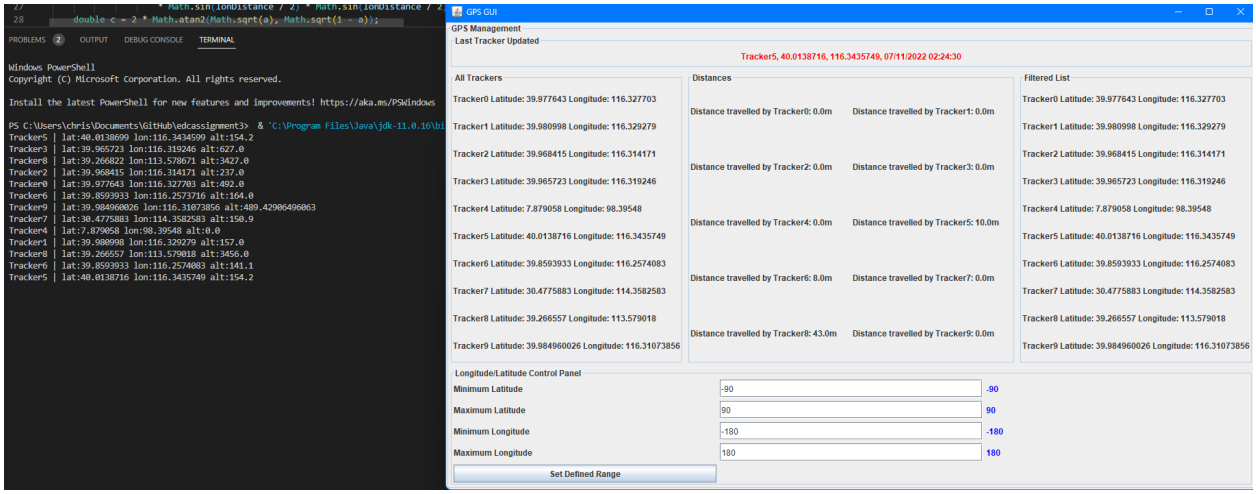
The value should be in meters (rounded up).

To test this section, I first just looked at the values of the distances traveled in correspondence to what was displayed on the output of the GPS Service.

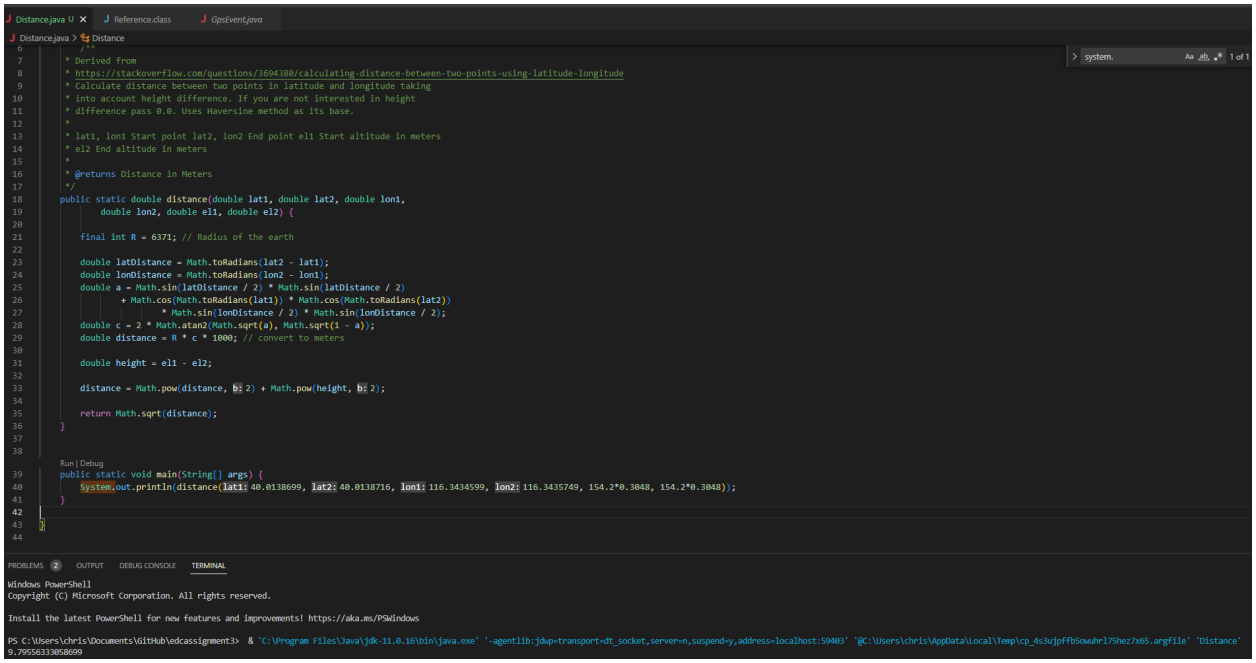




In this scenario, I got tracker5 for example and saw that the rounded value was 10.0 m.



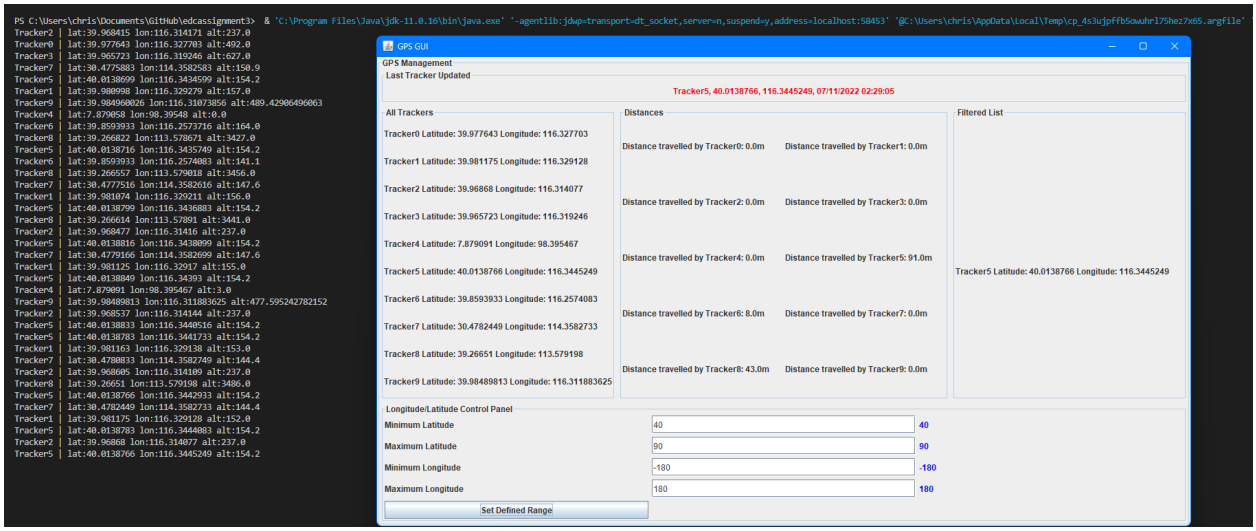
I then looked at the two strings that were shown highlighted at the beginning:
Tracker5 | lat:40.0138699 lon:116.3434599 alt:154.2
Tracker5 | lat:40.0138716 lon:116.3435749 alt:154.2
With the altitude multiplied by 0.3048 to convert to meters



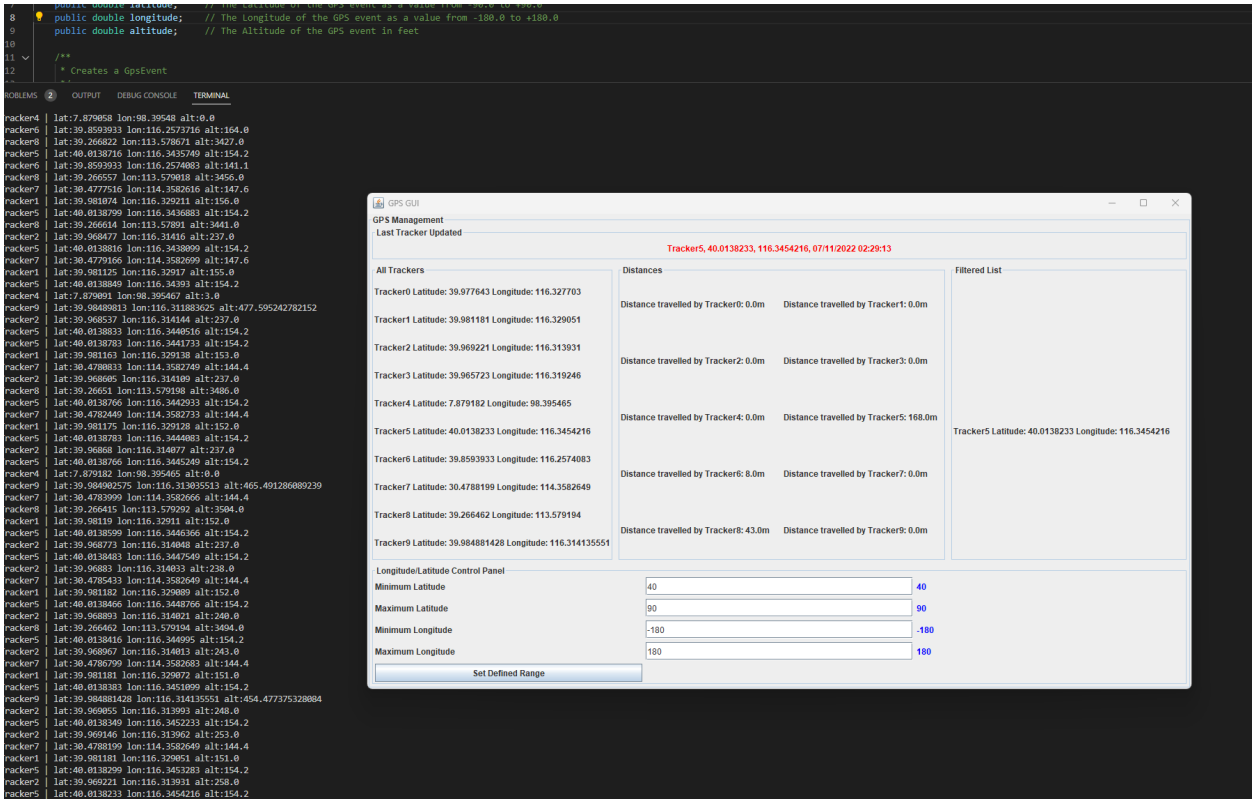
Which gives us 9.79556333058699 ~ 10 m
Therefore the rounding and distance calculation is correct. I repeated this for a few other strings.

Note this uses the distance formula from https://stackoverflow.com/questions/3694380/calculating-distance-between-two-points-using-latitude-longitude
Now to test that it only registers events that are within the range we set.

We quickly set the minimum latitude to 40 so that only tracker 5 meets the requirements



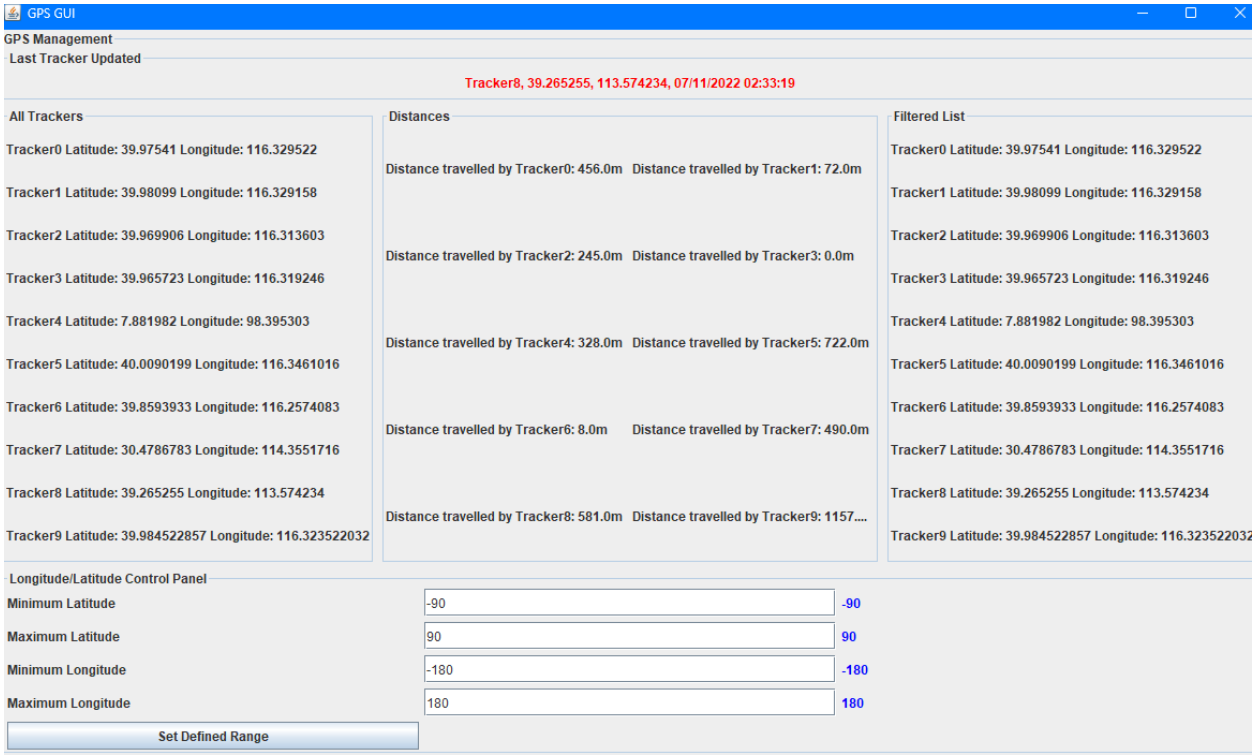
After a few seconds we find that everything else has remained the same except for tracker 5, which has gone from 91 m to 168m. Therefore we know that the filter is working and it only updates if the tracker is in range. We can also see from the terminal all the other events that are happening which do update on the All Trackers part of the program but does not register on the distance - which is a clear separation of both modules.



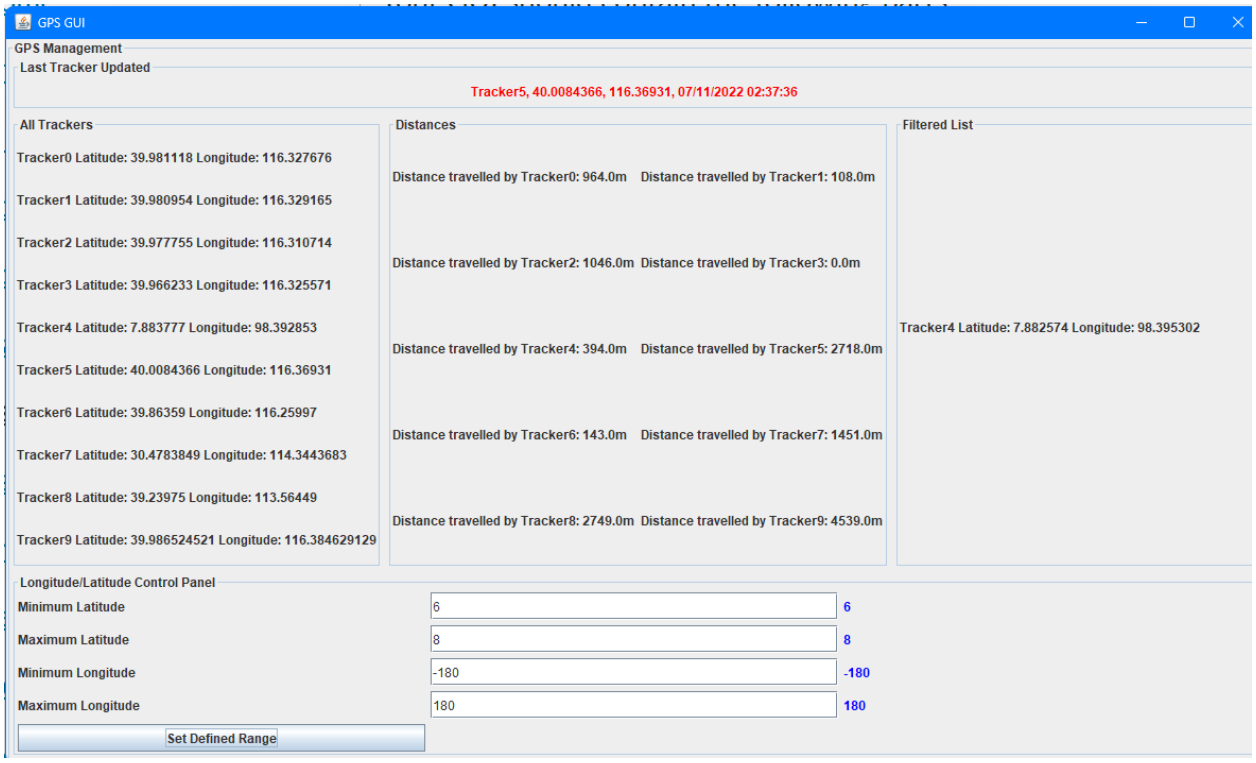
Section 4 + 5 Filtered List and Control Panel (Point 3)

Section 4 and 5 correspond to point 3 where the requirements for testing were that the defined range is controlled by a control panel consisting of a latitude input field and a longitude input field and a button to set this restriction.

As we can see there are the input fields of the minimum and maximum latitude and longitudes. The labels show the current setting of the restriction.

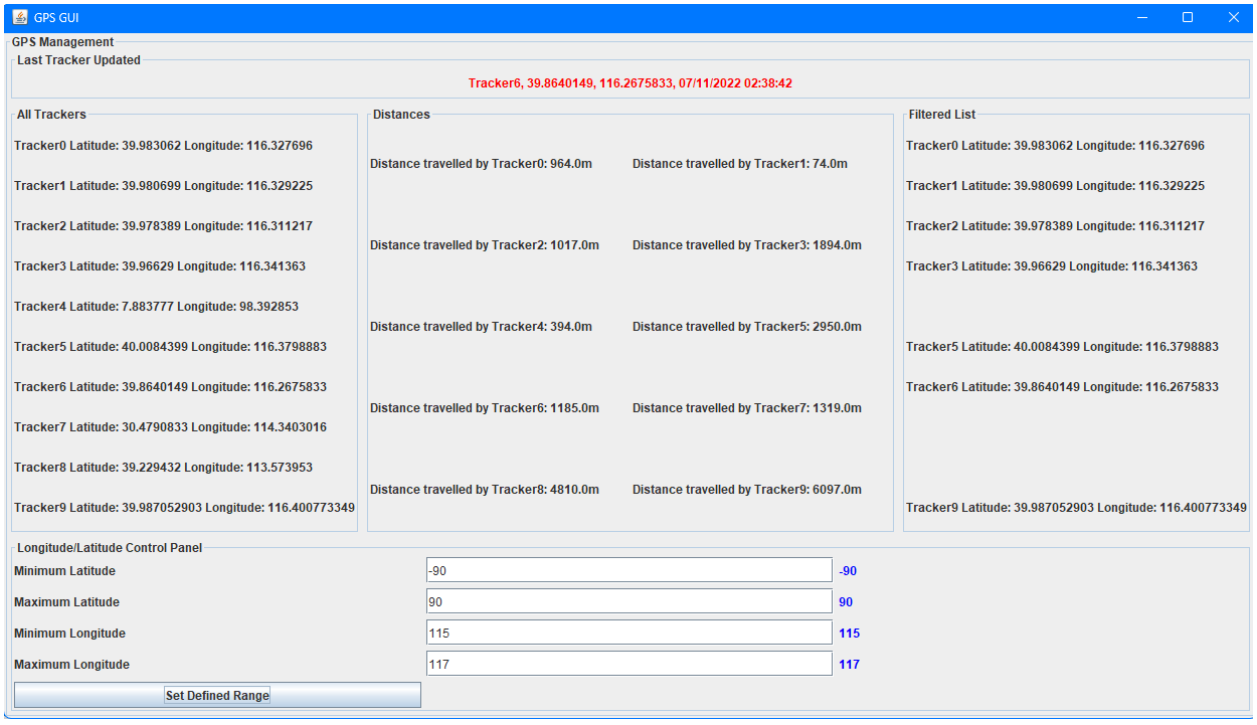


To test the normal use of this application, we can choose a tracker to filter out. For example, choosing to filter by latitude to show tracker4 as it is around 7 compared which is different from the others:



We can see the other trackers disappear as they are not in the range of the control panel.

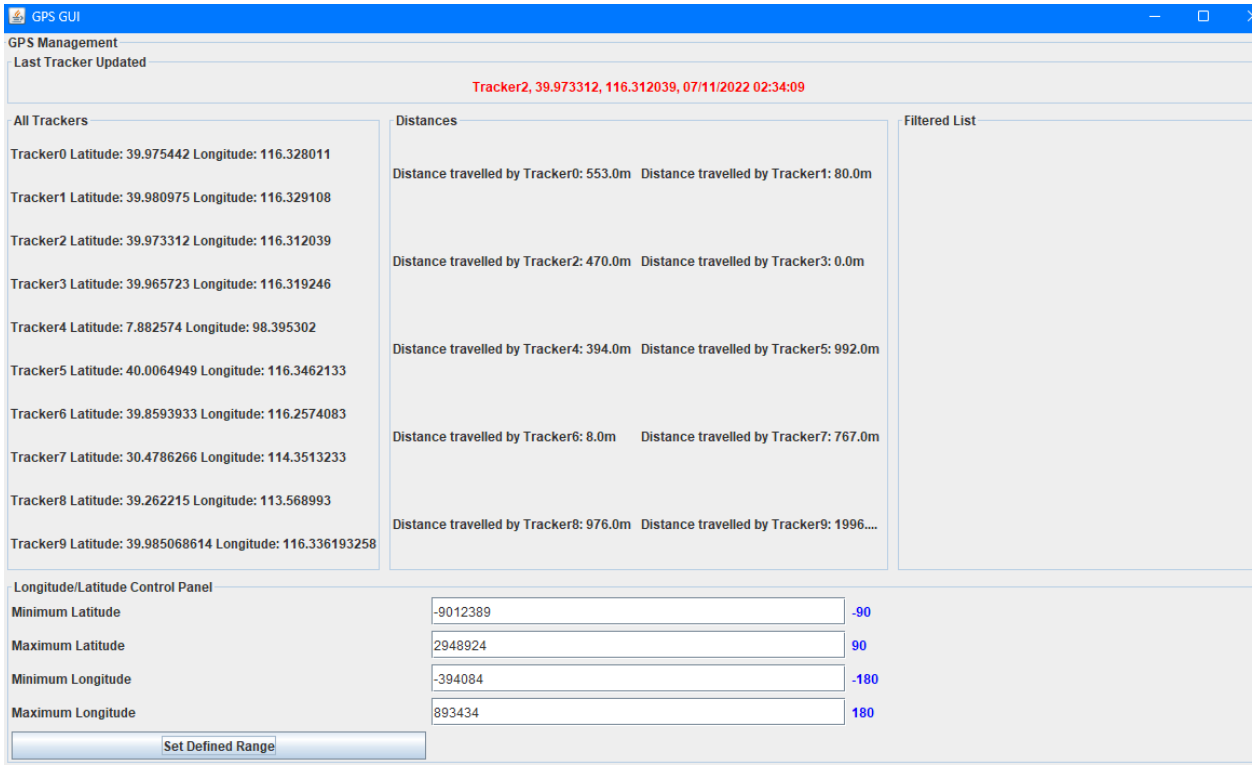
If we want to filter based on longitude, we can choose the trackers that fit in the 116 range.



As can be seen, this example test shows that the control panel works.

Edge Case / Error Handling:

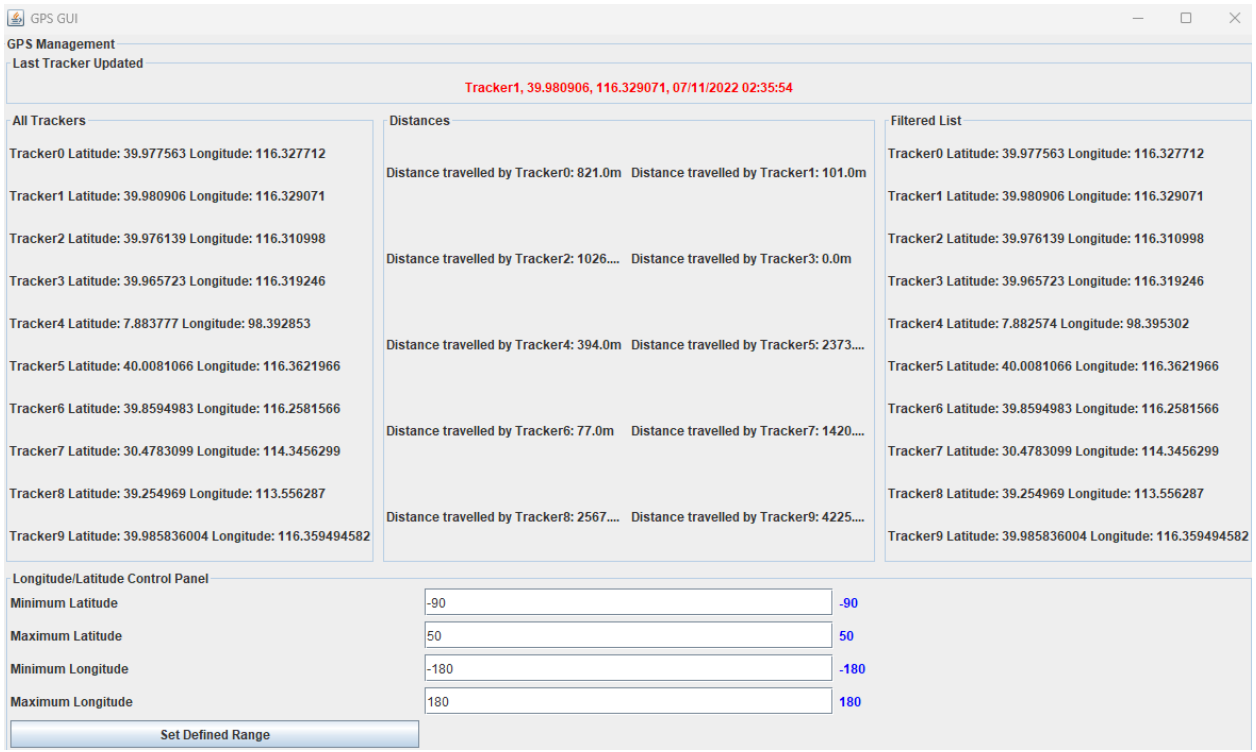
To test that the values do not go over the minimum and maximum values of -90, 90 and -180, 180 for latitude and longitude respectively, we try setting the defined range over.



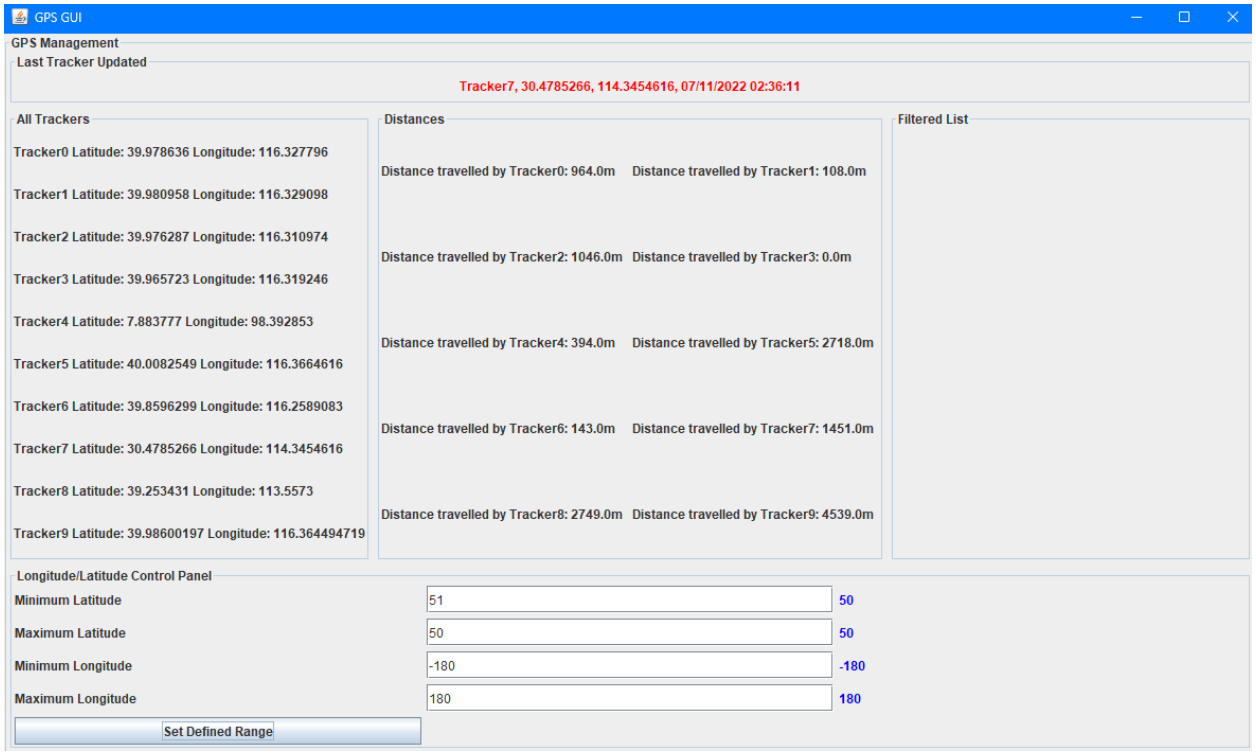
We can see that it doesn't register anything because it goes over the limit. Only registering the -90, 90, -180, and 180.

We can also test that the minimum can't go over the maximum and will be limited.

For example, in this case:



If we try and set minimum latitude to be larger than 51, we are capped at 50 instead.

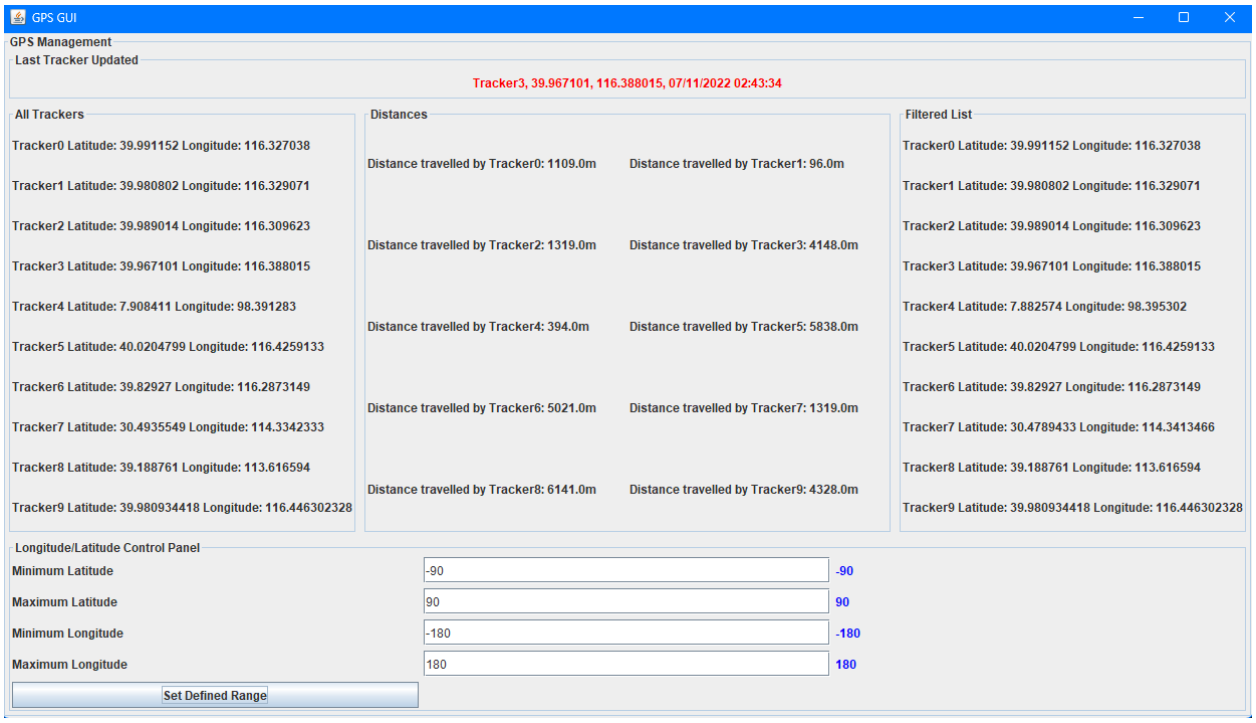


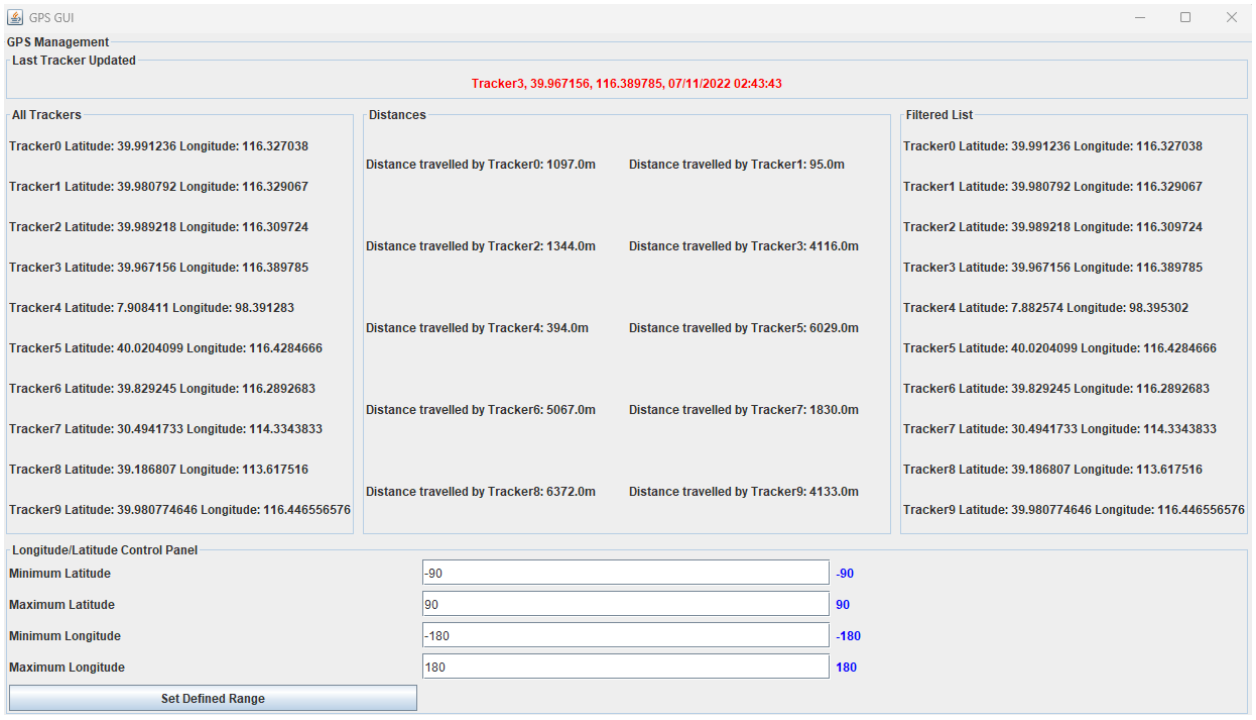
With the filter list not updating due to the invalid format.

Testing the 5 minute Window:

Quick Test

A quick test can be performed to see if there is indeed a window by seeing if there is a fluctuation both upwards and downwards of the distances after 5 minutes.





Which can be observed as so, with multiple trackers going down and some going up.

Complex Test

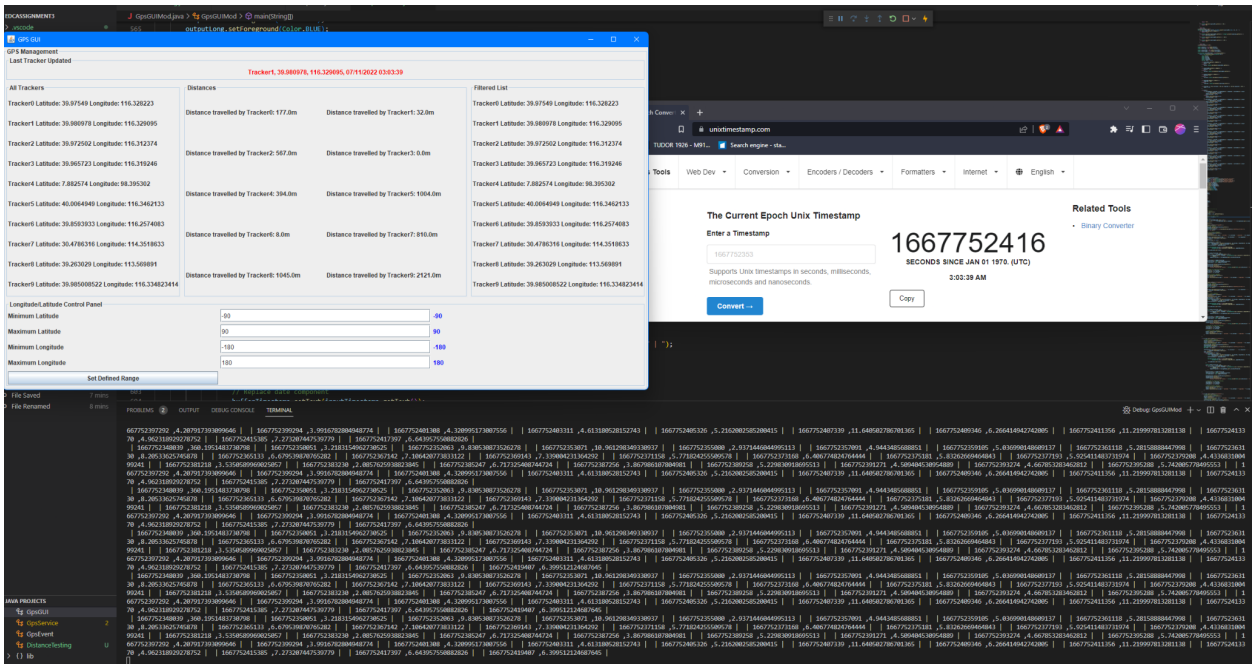
A more complex test would be to create a modification to the application as follows:

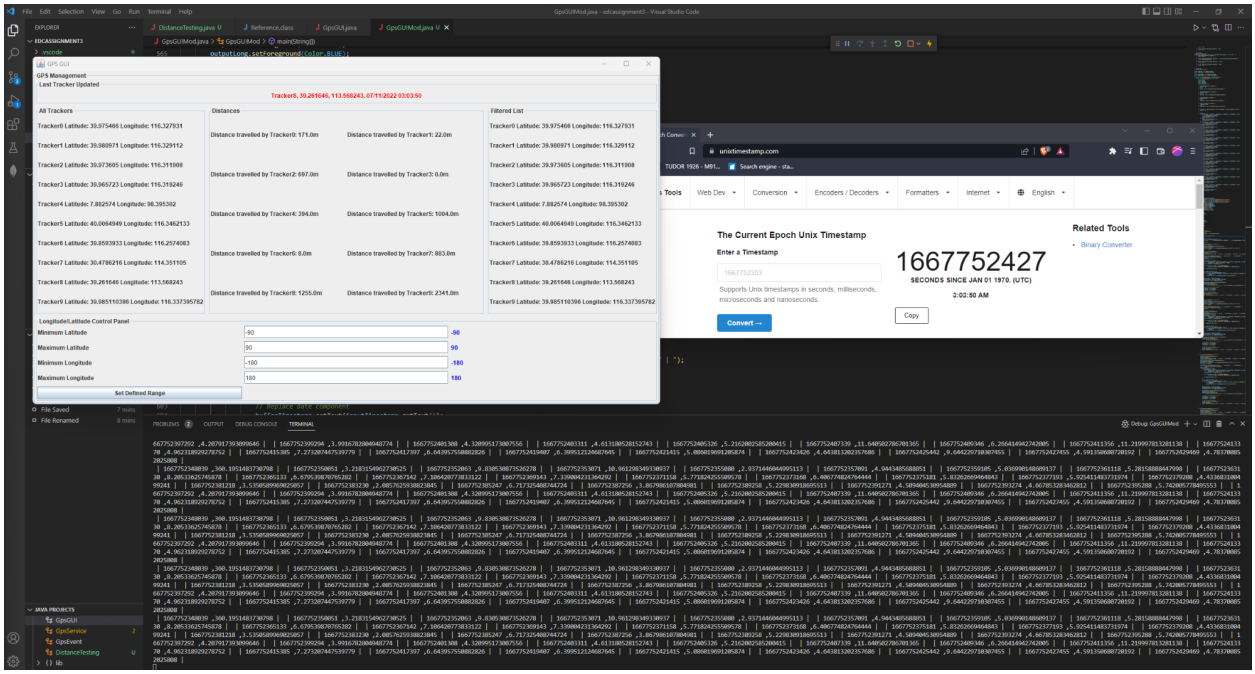
```
double tempDistTravelled = 0.0;
for (int i = 0; i < rollingWindow0.size(); i++) {
    if (rollingWindow0.get(i).timeAdded > (ZonedDateTime.now().toInstant().toEpochMilli()
            - 60000)) {
        tempDistTravelled += rollingWindow0.get(i).distanceTravelled;
    }
}
```

Time window only 1 minute

```
s.listen((GpsEvent ev) -> {
    for (int i=0; i < rollingWindow0.size(); i++) {
        System.out.print(" | " + rollingWindow0.get(i).timeAdded+ " ," + rollingWindow0.get(i).distanceTravelled + " | ");
    }
    System.out.println();
```

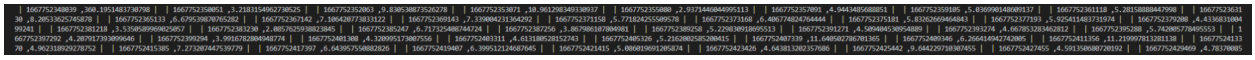
And a different output for tracker 0



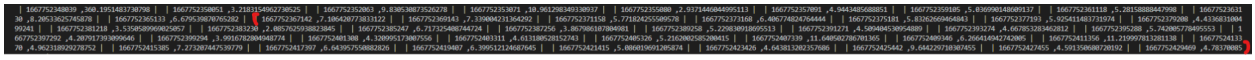


We see tracker0's value go from 177 to 171 from unix timestamp 1667752416 to 1667752427. 1667752427 - 60 = 1667752367.

If we look closely at the output of the timestamp to distance:



And we add up the corresponding amount that is higher than 1667752367:



~~| 1667752348039 ,360.1951483730798 | | 1667752350051 ,3.2183154962730525 | |~~
~~1667752352063 ,9.830530873526278 | | 1667752353071 ,10.961298349330937 | |~~
~~1667752355080 ,2.9371446044995113 | | 1667752357091 ,4.94434856888851 | |~~
~~1667752359105 ,5.036990148609137 | | 1667752361118 ,5.281588888447998 | |~~
~~1667752363130 ,8.20533625745878 | | 1667752365133 ,6.679539870765282 | |~~
~~1667752367142 ,7.106420773833122 | | 1667752369143 ,7.339004231364292 | |~~
1667752371158 ,5.771824255509578 | | 1667752373168 ,6.406774824764444 | |
1667752375181 ,5.83262669464843 | | 1667752377193 ,5.925411483731974 | |
1667752379208 ,4.433683100499241 | | 1667752381218 ,3.5350589969025057 | |
1667752383230 ,2.0857625938823845 | | 1667752385247 ,6.717325408744724 | |
1667752387256 ,3.867986107804981 | | 1667752389258 ,5.229830918695513 | |
1667752391271 ,4.509404530954889 | | 1667752393274 ,4.667853283462812 | |
1667752395288 ,5.742005778495553 | | 1667752397292 ,4.207917393099646 | |
1667752399294 ,3.9916782804948774 | | 1667752401308 ,4.320995173007556 | |
1667752403311 ,4.613180528152743 | | 1667752405326 ,5.2162002585200415 | |
1667752407339 ,11.640502786701365 | | 1667752409346 ,6.266414942742005 | |

1667752411356 ,11.219997813281138 | | 1667752413370 ,4.962318929278752 | |
1667752415385 ,7.273207447539779 | | 1667752417397 ,6.643957550882826 | |
1667752419407 ,6.399512124687645 | | 1667752421415 ,5.086019691205874 | |
1667752423426 ,4.643813202357686 | | 1667752425442 ,9.644229710307455 | |
1667752427455 ,4.591350680720192 | | 1667752429469 ,4.78370085
This meant these sections were the ones added up they would equal ~171

5.7718243		
6.4067748		
5.8326267		
5.9254115		
4.4336831		
3.535059		
2.0857626		
6.7173254		
3.8679861		
5.2298309		
4.5094045		
4.6678533		
5.7420058		
4.2079174		
3.9916783		
4.3209952		
4.6131805		
5.2162003		
11.640503		
6.2664149		
11.219998		
4.9623189		
7.2732074		
6.6439576		
6.3995121		
5.0860197		
4.6438132		
9.6442297		
4.5913507	SUM	
4.7837009	170.2305	

Which is correct when rounding up 170.2305

Meaning that the 1 minute = 60,000 UNIX millisecond window works, which can be generalized to the 5 * 60,000 = 30,000 UNIX millisecond time window.

Testing FRP operation for Feet to Meter Conversion:

```
// Altitude to Meter Conversion - Map
STextField alt = new STextField("");
Cell<String> textAlt = alt.sUserChanges.map(t -> {
    Double temp = Double.valueOf(t) * 0.3048;
    String out = String.valueOf(temp);
    return out;
}).hold(initValue: "");
SLabel altInMeters = new SLabel(textAlt);
```

This is the altitude code that converts feet to meters

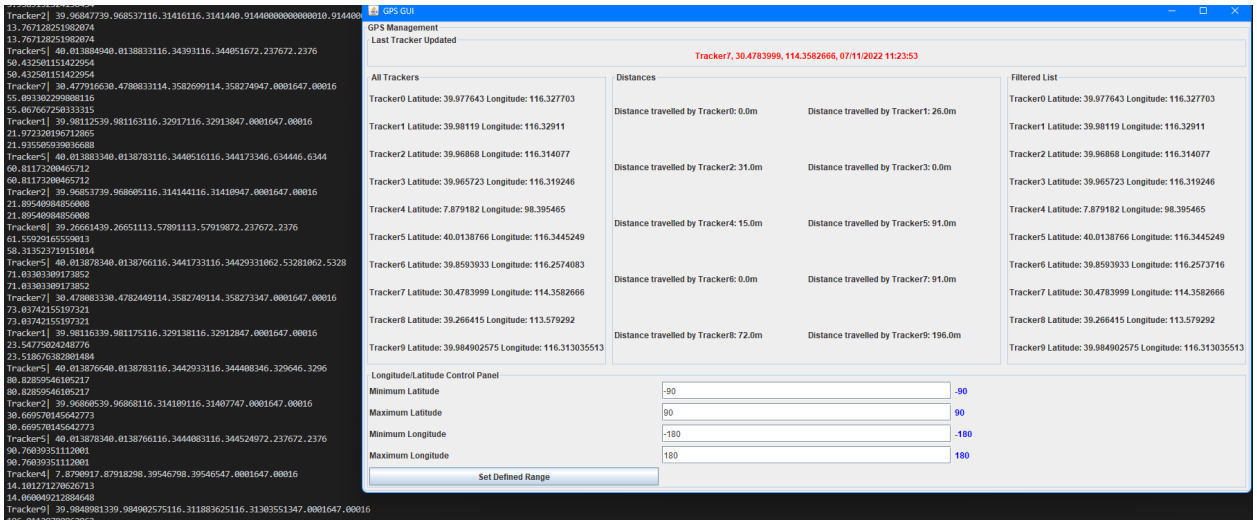
A modified GUI will be made to display the altitudes passed into the application and verified

```
alt2 = Double.valueOf(altInMeters.getText());
System.out.println(ev.name + "| "+ latitudes[9]+ ev.latitude+ longitudes[9]+ ev.longitude+ alt1 + alt2);


double updatedDistance = distance(latitudes[9], ev.latitude, longitudes[9], ev.longitude,
        alt1, alt2);

System.out.println(distTravelled[9]+distance(latitudes[9], ev.latitude, longitudes[9], ev.longitude,
altitudes[9]*0.3048, ev.altitude*0.3048));
distTravelled[9] += updatedDistance;
System.out.println(distTravelled[9]);
```

This makes it so the distance will be compared with the FRP operation and a normal ev.altitude call (multiplied by 0.3048)



It can be seen that the calculations are the same. Which means that the distance feet to meter map FRP conversion is working correctly.