Special Topic

z5308039 Sean Holschier

Executive Summary

This report shows the implementation of a particle filter in order to attempt to estimate AIS location data from a dataset of maritime ships around Singapore. The implementation features 7 different weighting methods, in an attempt to increase accuracy. Other factors including particle count and movement variance are also analysed. Results are graphed and discussed, where several observations are made and suggestions for improvement are outlined.

AIS Data and Particle Filters

Sean Holschier z5308039

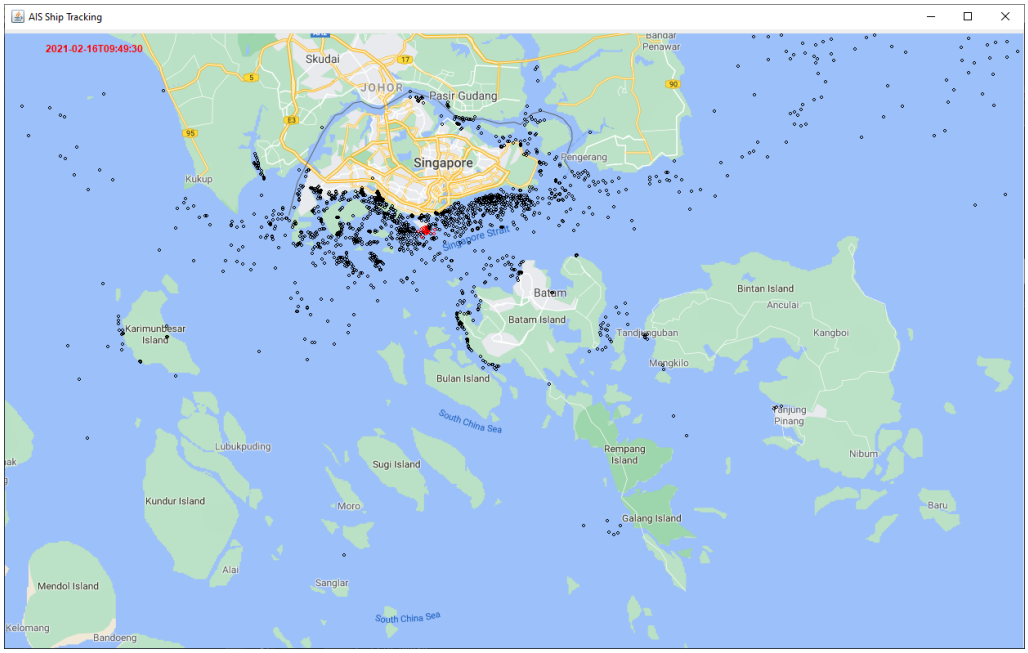
Introduction

In this project, I aim to create a particle filter which attempts to create an accurate estimation of the location of maritime ships that have missing AIS data in real-time. Data of all ships in the area will be given in real-time. Particle Filters aim to find the most accurate location of an entity through limited information and under the assumption that the information is not accurate and contains some noise. This is done by creating many particle 'estimations' of an object, moving them in a similar manner to the object and then assigning each particle with a weight to measure its accuracy. This weight is then used to resample all the particles for the next step, with a Darwinian 'Survival of the fittest' approach. I have compiled an explanation of particle filters, along with an example of a particle filter used for real-time object tracking, at https://1drv.ms/b/s!AjFCJC5sHfB5hVZoTJq-xbrWe_5D?e=CBi42Q.

The initial implementation will feature a very broad method of weighting for the particles, and I aim to increase the accuracy over the course of the project utilising different heuristics. In order to test a rough efficiency, I will be graphing the points of the highest weighted particle for each step in a filter and placing them on top of the points of the ship data that the program was given. In order to create gaps in the data given to us, only 10% of the ship data for the first day are inputted into the particle filter. The blue dots shown in the graph will show all the logs that the target ship submitted on that day, however, only a smaller amount of this data is shown to the filter.

Initial Implementation - Distance

For the initial implementation, more focus was given to the construction of the objects and the display of the GUI, so a rudimentary weighting method and movement method was used. The GUI displays a map of the area around Singapore, taken from Google Maps, which black dots signifying a ship. The timestamp for each frame is displayed on the top left corner. When the target ship does not have a log for more than 20 minutes, the particle filter will begin to run, with steps at every 10 minutes until a log is reached. The particles for each step are shown as red dots in the GUI.



1. Example of Project during a filtering step

The resampling method used in this particle filter is called the 'resampling wheel method' (Madrigal, 2012), a type of multinomial resampling is employed. This is a method which aims to remap the set of particles onto a new set of particles by essentially spinning a wheel, where particles have probabilities of getting new particles mapped onto them based on the weighting of the particle.

```
public void resample(){
    HashMap<String, Ship> newParticleShips = new HashMap<String, Ship>();
    float beta = 0f;
    //A random particle index is chosen
    Integer counter  = (int)Math.floor((random.nextFloat()*(19)));
    //beta is a random value between 0 and double the highest weight.
    //The particle that aligns with beta after subtracting subsequent weights is selected to have a particle mapped to it
    System.out.println(bestPart().getLastLoc(time).getLatitude() + "," + bestPart().getLastLoc(time).getLongitude());
    for (Integer i=0; i<particleNum; i++){
        beta += random.nextFloat()*2f*bestPart().getWeight();
        while (beta > particleShips.get(counter.toString()).getWeight()){
            beta -= particleShips.get(counter.toString()).getWeight();
            counter = (int)loopBack(counter+1, 0, 19);
        }
        newParticleShips.put(i.toString(), new Ship(i.toString()));
        newParticleShips.get(i.toString()).copyRecord(particleShips.get(counter.toString()).getLastLoc(time));
    }
    particleShips = newParticleShips;
}
```

2. Code displaying the resampling wheel method

Weighting

For the weighting of each particle, the distance (d) to the closest past location of any ship was used. The equation used to determine the weight was: $Weight = 1000^{-d}$. 1000 was used over the initial 2 as the constant as I felt the drop as the distance increased needed to be larger.

```
public void weight1(){
    for (Ship s: particleShips.values()) {
        s.setWeight(Math.pow(1000,-(getClosestShip(s))));
    }
}
```

3. Code showing the first weighting method

Movement

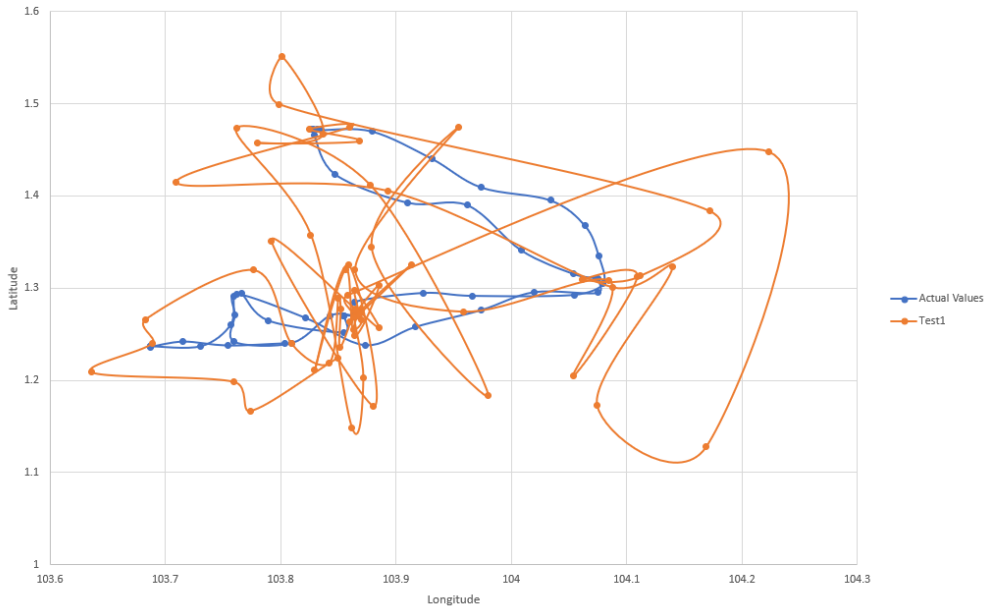
For the movement of each particle, the last log by the ship was used to determine the speed and course. Each particle was also given random noise to allow for change in movement for each ship.

```java
public void stepShip(Duration duration, ShipLog lastLog, LocalDateTime time) {
    Double lati = lastLog.getLatitude();
    Double longi = lastLog.getLongitude();
    Double speed = lastLog.getSpeed() + (float)random.nextGaussian()*2;
    Double course = lastLog.getCourse() + (float)random.nextGaussian()*30;
    long timeLength = duration.toSeconds();
    Double dist = (speed * 0.514444) * timeLength;
    Double distLati = (dist / 110574);
    Double distLongi = (dist / 111320);
    Double newLat = (lati +(float)Math.cos(Math.toRadians(course))*(distLati));
    Double newLong = (longi +(float)Math.sin(Math.toRadians(course))*(distLongi));
    addRecord(time, newLat, newLong, speed, course);
}
```

4. Code showing movement of ship

Effectiveness of Initial Implementation

As the following graph shows, the weighting by distance is not a very accurate or effective measure. The weighting tends to ensure that the particles remain in a similar area but also tend to stray away from paths that the ship actually took, as it weights itself based on the distance from all the ships around it. This would not make a realistic implementation since it only checks whether ships had been close to the particle location in the past.



5. Scatter plot of Test1 values against actual values

First Update - Course

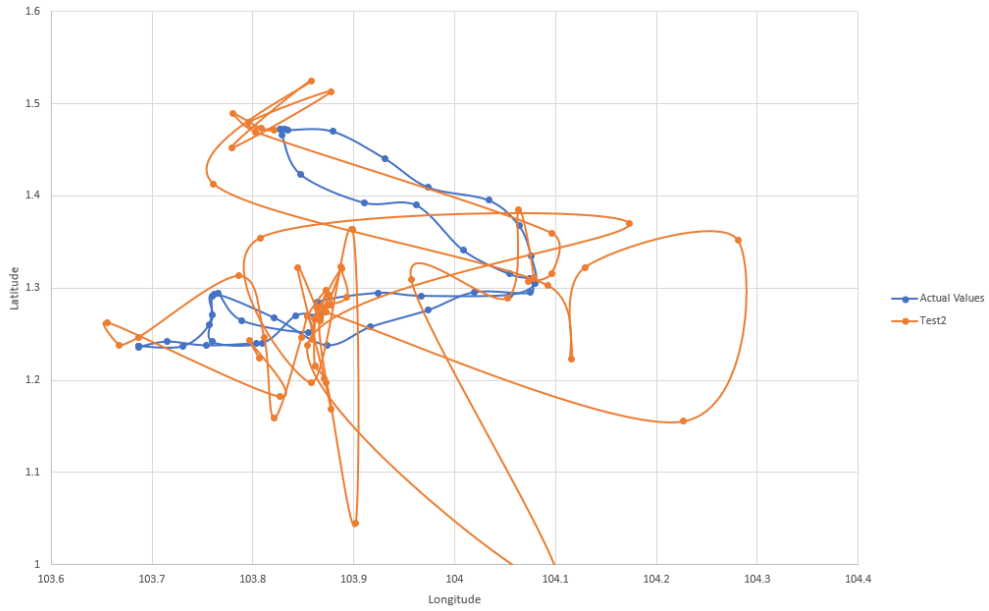
A new heuristic for the weighting was added, where particles are weighted based on the course of the ship logs nearby. Where c is the difference in course (in degrees) between the particle and the ship log with the least difference that was recorded within an ~5.5km radius (roughly 0.05 latitude and longitude), the weighting is calculated with the equation: $Weight = 0.8^c$. With this weighting method I wished to check the effectiveness of weighing based on how similar the course was to ships around the particle.

```java
public void weight2(){
    for (Ship s: particleShips.values()) {
        LinkedList<ShipLog> closeShips = getCloseShips(s);
        Double courseDiff = 360.0;
        for (ShipLog f: closeShips) {
            double diff = Math.abs(f.getCourse()-s.getLastLoc(time).getCourse());
            if (diff<=courseDiff) {
                courseDiff = diff;
            }
        }
        s.setWeight(Math.pow(0.8,(courseDiff)));
    }
}
```

6. Code showing the second weighting method

Effectiveness of First Update

It is difficult to say whether the update has had a positive change in the accuracy of the filter, as we can see some erratic movement as wild as the initial implementation. Some particles have strayed from paths that any other ship logs have taken, as distance is not of importance in this weighting.



7. Scatter plot of Test2 values against actual values

Second Update – Distance + Course

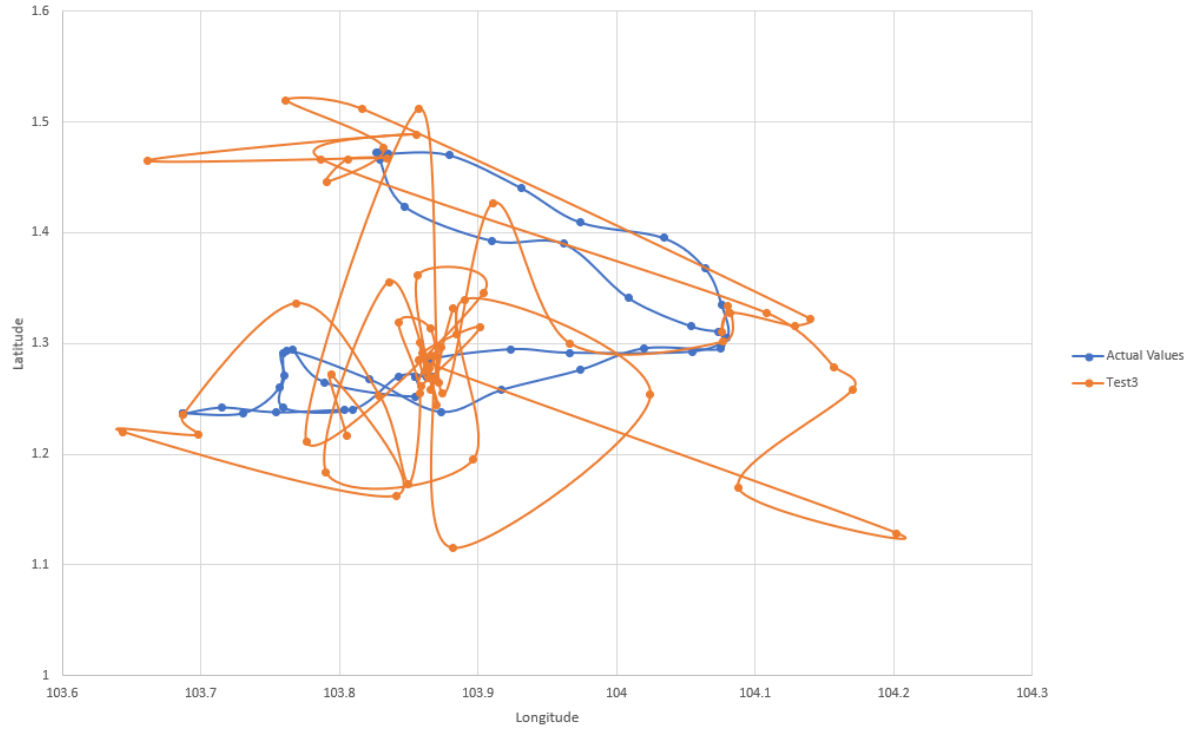
This third method for weighting combines the first two weighting methods, thus combines the distance the particle is from other particles, and the weighting based on the course of the ship logs nearby. The weighting equation is $Weight = 0.8^{c}+1000^{-d}$. The aim of this weighting method is to check whether there were ships near the particle in the past that were heading in the same direction.

```
public void weight3(){
    for (Ship s: particleShips.values()) {
        LinkedList<ShipLog> closeShips = getCloseShips(s);
        Double courseDiff = 360.0;
        for (ShipLog f: closeShips) {
            double diff = Math.abs(f.getCourse()-s.getLastLoc(time).getCourse());
            if (diff<=courseDiff) {
                courseDiff = diff;
            }
        }
        s.setWeight(Math.pow(0.8,(courseDiff))+Math.pow(1000,-(getClosestShip(s))));
    }
}
```

8. Code showing the third weighting method

Effectiveness of Second Update

One again we receive similar results to the first two implementations, and it is difficult to determine the accuracy of the results. While some particles are still straying far from the path the ship took, the number has seemingly decreased.



9. Scatter plot of Test3 values against actual values

Third Update – Distance + Velocity

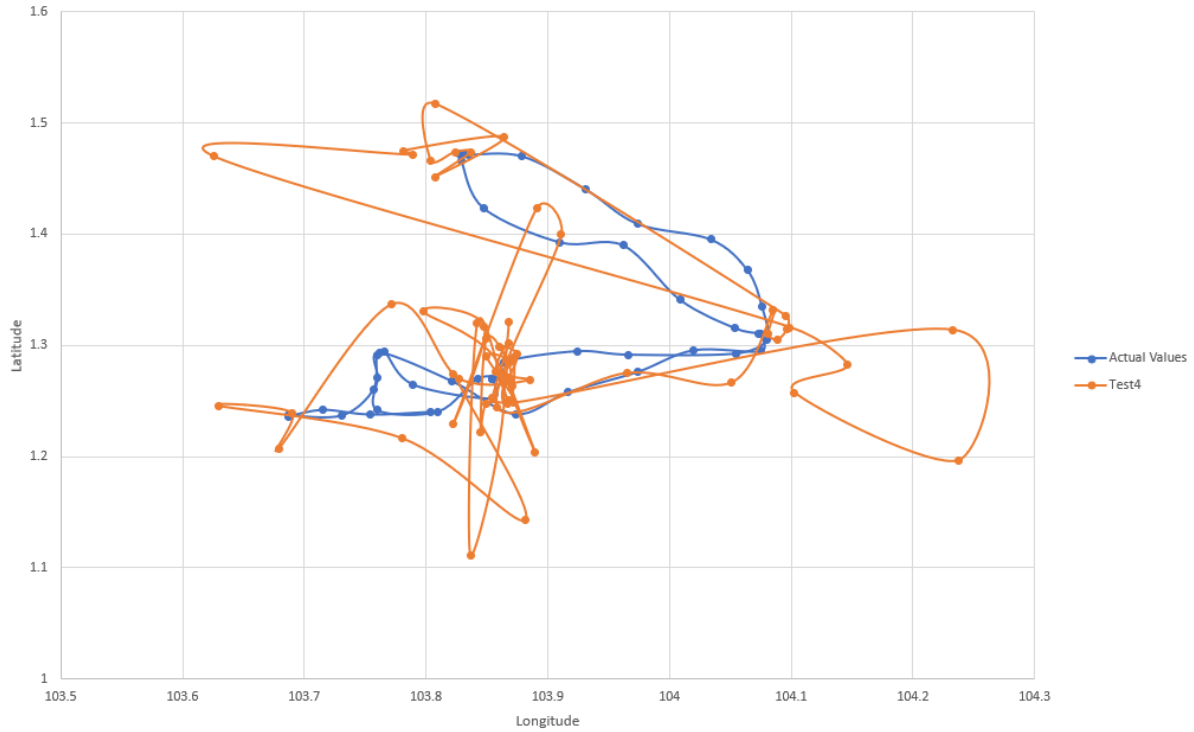
This introduced how close the velocity of ships around it was to the particle, as a factor to determine the weighting, in addition to the distance from the closest ship. Where *v* is the different in vector (in knots), the weighting equation is $Weight = 0.8^{v}+1000^{-d}$

```
public void weight4(){
    for (Ship s: particleShips.values()) {
        LinkedList<ShipLog> closeShips = getCloseShips(s);
        Double speedDiff = 30.0;
        for (ShipLog f: closeShips) {
            double diff = Math.abs(f.getSpeed()-s.getLastLoc(time).getSpeed());
            if (diff<=speedDiff) {
                speedDiff = diff;
            }
        }
        s.setWeight(Math.pow(0.8,(speedDiff))+Math.pow(1000,-(getClosestShip(s))));
    }
}
```

10. Code showing the fourth weighting method

Effectiveness of Third Update

The addition of velocity has had a positive effect on the filter, with less particles straying far from the actual path. However, the filter is still struggling with the turns, especially when the ship takes a turn larger than ~150 degrees.



11. Scatter plot of Test4 values against actual values

Fourth Update – Distance + Course + Velocity

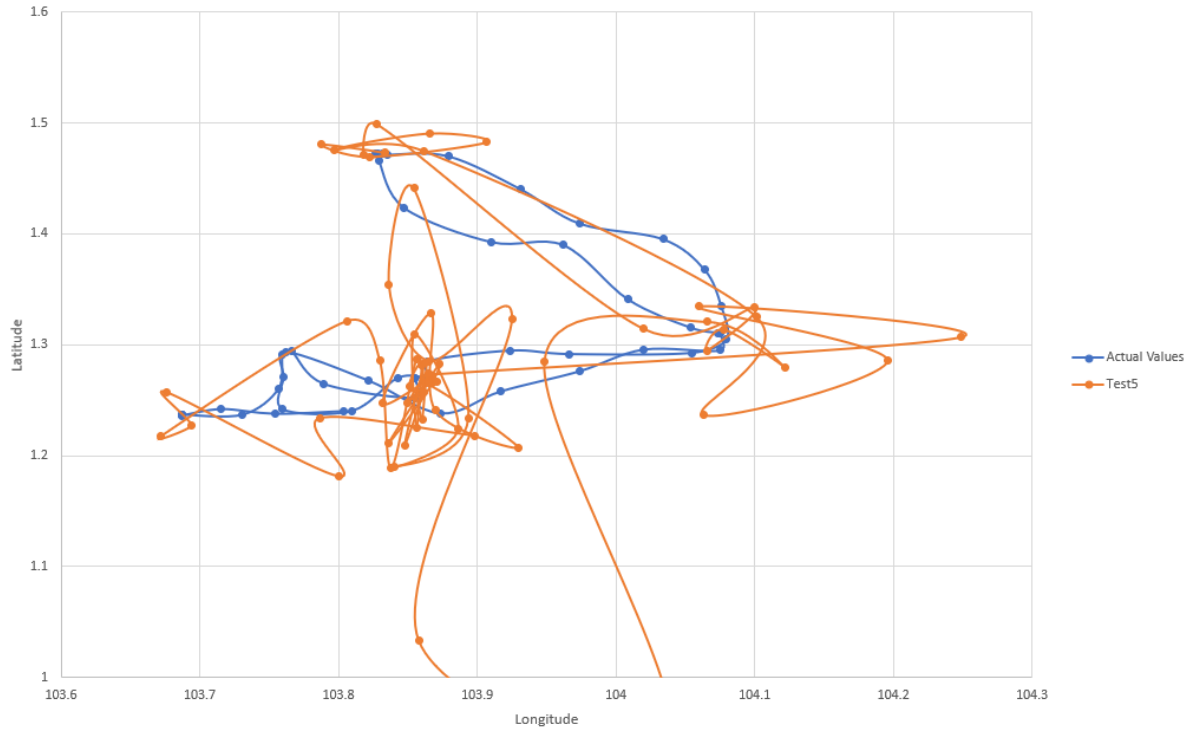
I then tried implementing all three factors into my weighting method, with the distance weight, course weight and velocity weight being added together. $Weight = 1000^{-d} + 0.8^{c} + 0.8^{v}$

```
public void weight5(){
    int counter =1;
    for (Ship s: particleShips.values()) {
        LinkedList<ShipLog> closeShips = getCloseShips(s);
        Double speedDiff = 30.0;
        Double courseDiff = 360.0;
        counter++;
        for (ShipLog f: closeShips) {
            double diff = Math.abs(f.getCourse()-s.getLastLoc(time).getCourse());
            if (diff<=courseDiff) {
                courseDiff = diff;
            }
            double sdiff = Math.abs(f.getSpeed()-s.getLastLoc(time).getSpeed());
            if (diff<=speedDiff) {
                speedDiff = diff;
            }
        }
        s.setWeight(Math.pow(0.8,(courseDiff)) + Math.pow(0.8,(speedDiff))+Math.pow(1000,-(getClosestShip(s))));
    }
}
```

12. Code showing the fifth weighting method

Effectiveness of Fourth Update

Compared to the third update, some of the far-straying particles have seemingly returned, however the filter seems to be approaching turns differently, as there seems to be more particles clumped around the major turns in the path.



13. Scatter plot of Test5 values against actual values

Fifth Update – Distance + Course + Velocity (Same ShipLog)

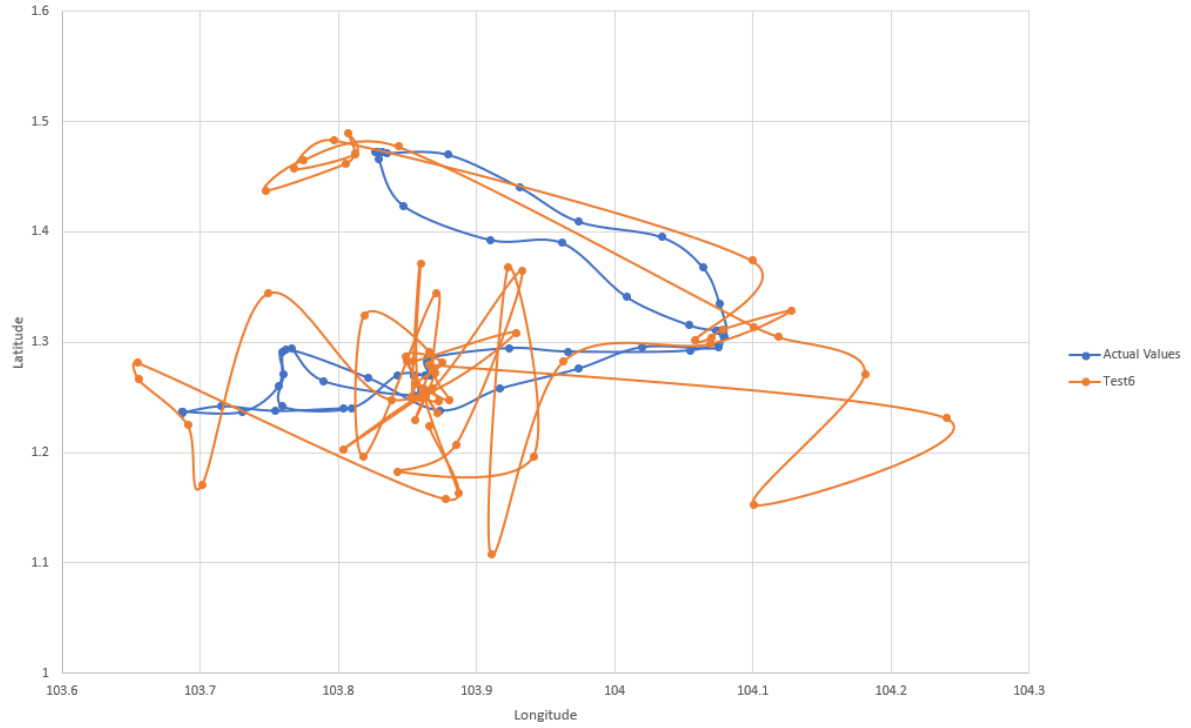
In this weighting method, rather than taking the best weight for each factor and adding them, I found the best weight when distance, course and velocity were all weighted against the same ship log. The equation is the same, $Weight = 1000^{-d} + 0.8^{c} + 0.8^{v}$, but *d*, *c* and *v* are all against the same log.

```
public void weight6() {
    for (Ship s: particleShips.values()) {
        LinkedList<ShipLog> closeShips = getCloseShips(s);
        double bestWeight = 0.0;
        for (ShipLog f: closeShips) {
            double currWeight = 0.0;
            double courseDiff = Math.abs(f.getCourse()-s.getLastLoc(time).getCourse());
            double speedDiff = Math.abs(f.getSpeed()-s.getLastLoc(time).getSpeed());
            currWeight = (Math.pow(0.8,(courseDiff)) + Math.pow(0.8,(speedDiff))+ Math.pow(1000,-(getDistance(f, s.getLastLoc(time)))));
            if (currWeight > bestWeight) {
                bestWeight = currWeight;
            }
        }
        s.setWeight(bestWeight);
    }
}
```

14. Code showing the sixth weighting method

Effectiveness of Fifth Update

We can see that the accuracy of the filter is beginning to rise, however the filter is still struggling at corners and turns. The number of large deviations from the actual path have decreased showing some improvement in the weighting method.



15. Scatter plot of Test6 values against actual values

Sixth Update

Rather than comparing the particle against the best suited ship log, here I have grabbed the 10 closest ship logs, and set the weight to be the sum of the weights against all 10 ship logs.

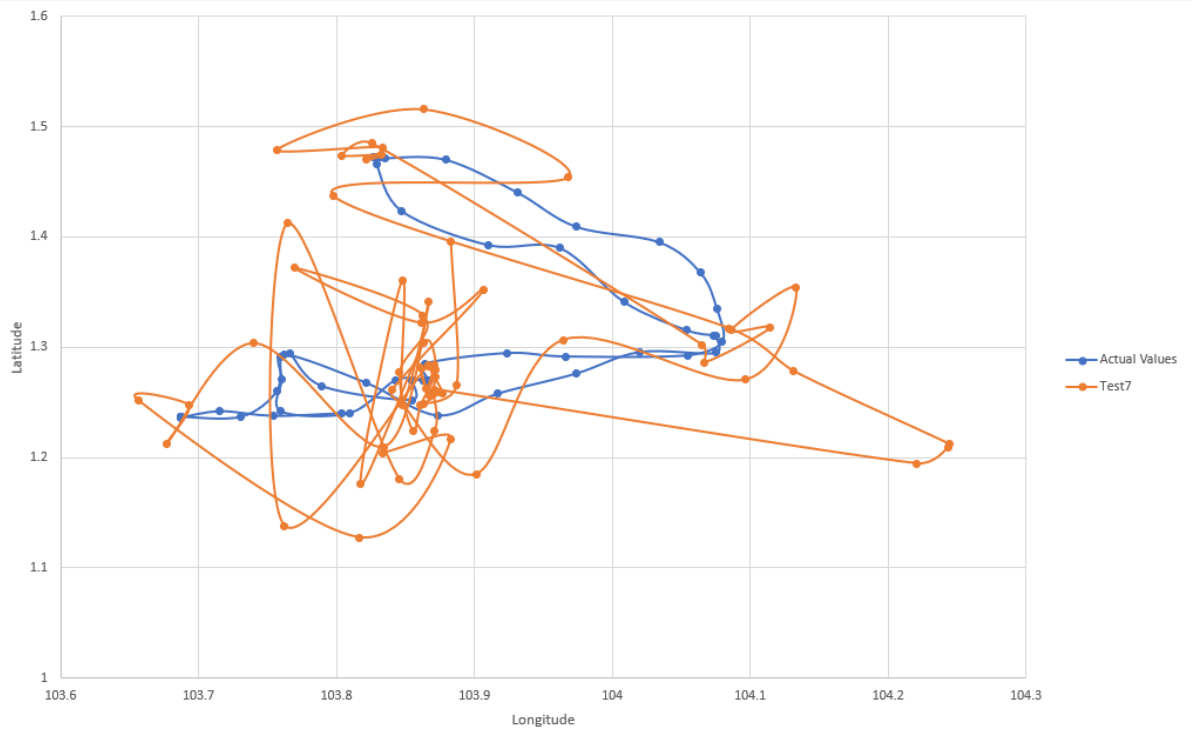
```
public void weight7() {
    for (Ship s: particleShips.values()) {
        TreeMap<Double,ShipLog> close10Ships = getTenClosest(s);
        double totalWeight = 0.0;
        for (ShipLog f: close10Ships.values()) {
            double currWeight = 0.0;
            double courseDiff = Math.abs(f.getCourse()-s.getLastLoc(time).getCourse());
            double speedDiff = Math.abs(f.getSpeed()-s.getLastLoc(time).getSpeed());
            currWeight = (Math.pow(0.8,(courseDiff)) + Math.pow(0.8,(speedDiff))+ Math.pow(1000,-(getDistance(f, s.getLastLoc(time)))));
            totalWeight += currWeight;
        }
        s.setWeight(totalWeight);
    }
}

public TreeMap<Double,ShipLog> getTenClosest(Ship ship) {
    ShipLog log1 = ship.getLastLoc(time);
    ShipLog dummy = null;
    TreeMap<Double,ShipLog> returnvalue = new TreeMap<Double,ShipLog>();
    for (int i=0; i<10;i++) {
        returnvalue.put(i+10.0, dummy);
    }
    for (Ship s: ships.values()) {
        if (!s.getShipID().equals(ship.getShipID())) {
            for (ShipLog log: s.getShipData().values()) {
                if(!log.getTimeStamp().isAfter(time)) {
                    Double tempDist = getDistance(log, log1);
                    if(tempDist < returnvalue.lastKey()){
                        returnvalue.remove(returnvalue.lastKey());
                        returnvalue.put(tempDist, log);
                    }
                }
            }
        }
    }
    return returnvalue;
}
```

16. Code showing the seventh weighting method

Effectiveness of Sixth Update

This weighting method showed a similar result to the last method, but I believe this method shows a better representation of the particles likeliness to be representing where the ship went, as it is gives the weighting from the fifth update, but against the 10 closest ships.

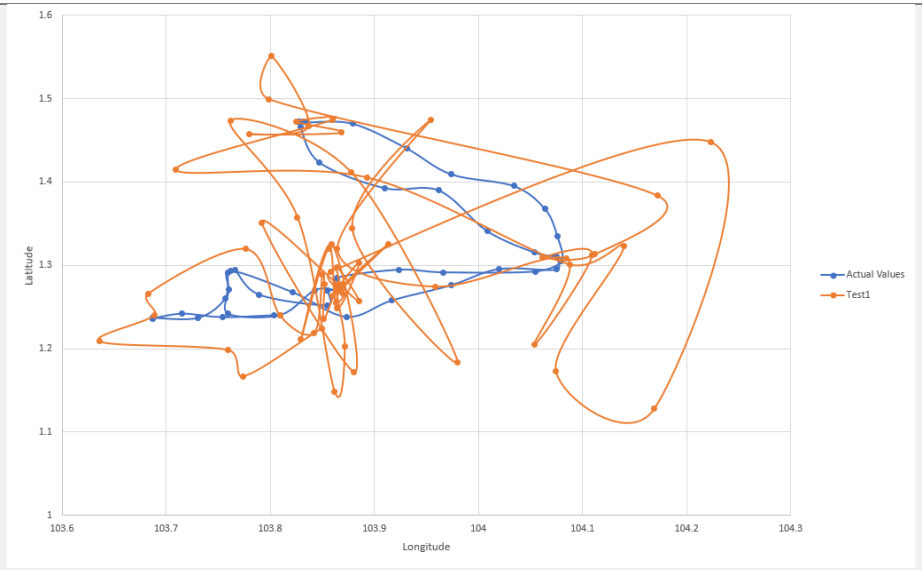
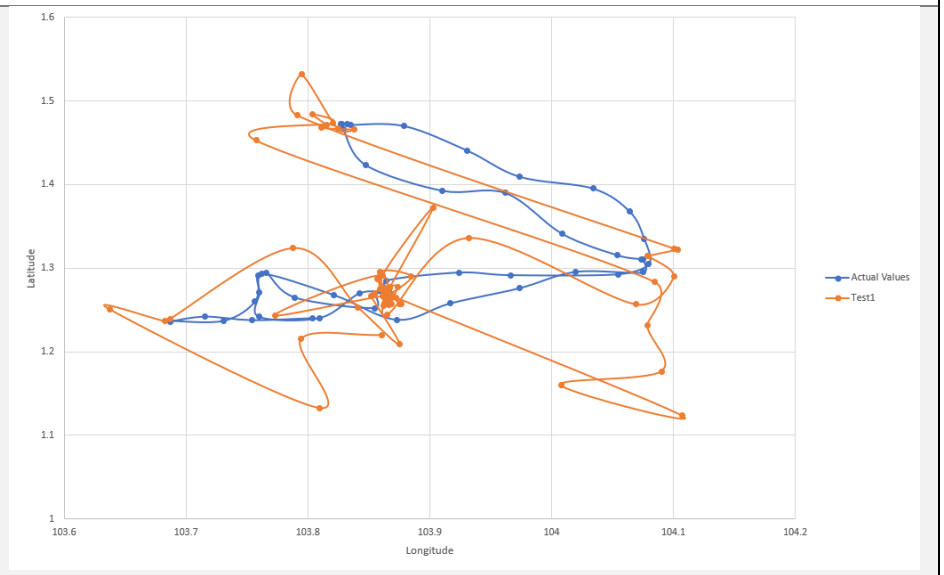
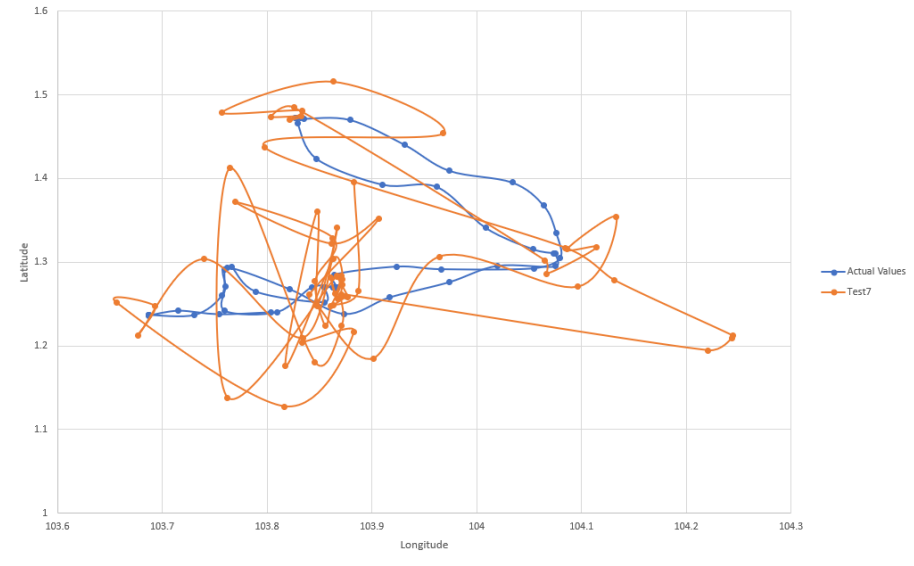
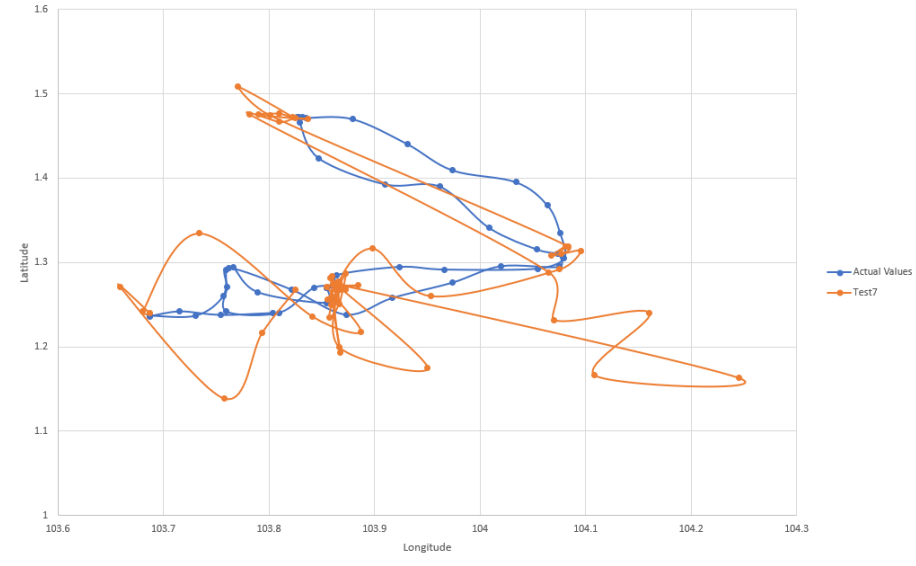


17. Scatter plot of Test7 values against actual values

Change in speed deviation

Every step in the filter, the speed of the last ship log, plus a random gaussian to a factor of 5-knots, is used to move the particles before weighting. In this section I wish to examine the effect of the factor of the random gaussian, and how much effect it has on how much the particles deviate from the path.

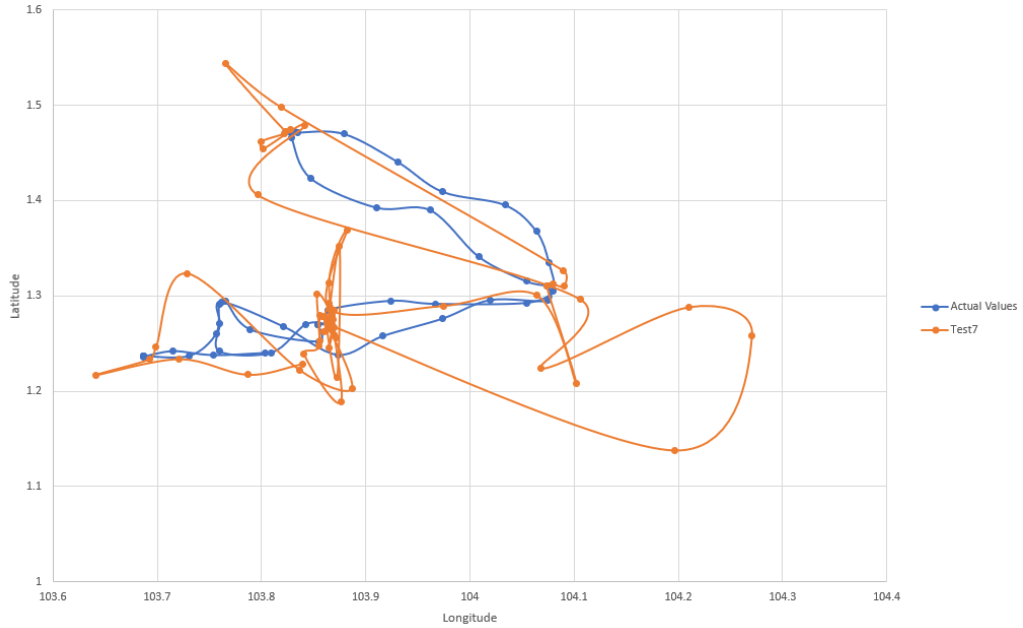
When the initial implementation and the fifth update are rerun with the 5-knot(9.26km/hr) factor being changed to 2-knots(3.70km/hr), a dramatic decrease in the number of stray particles being highly weighted were seen. This leads me to believe that the speed of the ship's last log is a great factor in determining where the ship might be headed, and that the increase in variance is greatly decreasing the accuracy. The 2-knot factor for test 7 shows the least amount of straying, except for the south-eastern part of the path, where the filter is led astray by the many ships heading past the Singapore port and up north. This can be viewed as the closest to accurate the filter has displayed out of all the tests.

	5-KNOT FACTOR	2-KNOT FACTOR
TEST 1 (INITIAL)		
TEST 7 (6TH UPDATE)		

18. Comparison of test 1 and test 7 under two different knot variances

Number of particles

In order to minimise computing time, the filter had been set to 100 particles per step. However, to account for more possible movements, I wish to examine the effects of increasing the number of particles to 1000. I will run this on the 6th update implementation, with the movement variance of 2-knots, as this showed the most promise out of all the implementations. The simulation with 1000 particles took over 30 minutes to run.



19. Scatter plot of Test7 values against actual values (1000 particles)

From this graph we can see that the number of particles had little effect on the filter. We do see an increase in particles going north in the centre, which is most likely due to the increased number of particles making it more likely for some to find high weighted spots across land, which seems to have happened here.

Discussion

The particle filter, in its 6th update, with a change in movement factor, displayed estimations that roughly represented the ship was in. While the filter had rough estimations, it cannot be said conclusively that the particle filter is accurate or successful. However, many observations can be made and improvements for future implementations and attempts can be suggested.

The larger the variance that is used in the movement part of the filter is, the more spread we can see in the particles. This can lead to results that stray from the more possible locations for the ship. This could possibly be countered by adding 'reasonableness' of the movement as a factor (some calculation of would the ship go into this area etc.)

The current implementation has no way of knowing whether a particle is in fact on land or not. This can cause particles to simulate ships crossing land masses. To prevent this, a bitmap could be implemented, allowing for a way to determine whether a particle is on land or not. Particles that have been placed on land could then receive a weighting of zero, as it is not a possible location.

As seen in nearly all of the tests, the filter had trouble with the south-eastern part of the path. This is due to the large number of ships that head north past Singapore, which affect the weighting of the particles. The ship chosen is the SEA FALCON 17, a pleasure craft (vesselfinder, 2021), which will have a differing path to many other ships, such as commercial crafts and oil tankers. Filtering ships based on data only from similar ship types could improve the accuracy of the particle filter, since the regular paths of oil tankers would have little influence of what path a pleasure craft would take. However, as the nature of the ship is not included in the dataset, it is difficult to classify and separate

the ships. A more comprehensive database including ship type/purpose that could be accessed by the particle filter could assist in improving the overall accuracy.

Conclusion

To summarise, in this project I explored particle filters and their application in regard to estimating AIS data, with an example of how one such filter could operate. Within this, I explored various weighting methods, and found how varying movement variance and quantities of particles affected the accuracy of the filter. In my report I found that utilising a Distance + Course + Velocity weighting method against the 10 closest ship logs, with a 2-knot variance factor was the most effective in minimising large deviations from the dataset. In my discussion I outlined methods for improvement, including classification of ships based on usage and type, utilising a bitmap to remove particles on land and a possible 'reasonableness' weighing factor, in order to increase the accuracy of the particle filter.

Bibliography

Buchner, J. (2019, 2). *johnhw/pfilter*. Retrieved from Github: https://github.com/johnhw/pfilter

Candy, J. (2007). Bootstrap Particle Filtering. *IEEE Signal Processing Magazine*, 73-85.

Chen, C., Wang, H., Ali, A., Lorenzelli, F., Hudson, R., & Yao, K. (2006). *PARTICLE FILTERING APPROACH TO LOCALIZATION AND TRACKING OF A MOVING ACOUSTIC SOURCE IN A REVERBERANT ROOM.* Los Angeles, CA: Electrical Engineering Department, Computer Science Department, UCLA.

Fernández-Madrigal, J.-A. (2013). Appendix B: Resampling Algorithms. In J.-A. Fernández-Madrigal, *Simultaneous localization and mapping for mobile robots introduction and methods* (pp. 407-411). Hershey, PA : Information Science Reference.

Hanson, B., Robeson, S., & Klink, K. (1992). Vector Correlation: Review, Exposition, and Geographic Application. *Annals of the American Association of Geographers*, 103-116.

Jin, C., Lin, Z., & Wu, M. (2019). *Augmented Intention Model for Next-Location Prediction from Graphical Trajectory Context.* Zhejiang, China: Hindawi Wireless Communications and Mobile Computing.

Jinan, R., & Raveendran, T. (2015). Particle Filters for Multiple Target Tracking. *International Conference on Emerging Trends in Engineering, Science and Technology*, 980-987.

Jurafsky, D. M. (2020). Hidden Markov Models. In D. M. Jurafsky, *Speech and Language Processing.*

Jurić, D. (2015, 4). *Object Tracking: Particle Filter with Ease*. Retrieved from CodeProject: https://www.codeproject.com/Articles/865934/Object-Tracking-Particle-Filter-with-Ease

Karunakaran, D. (2018, March 14). *Kidnapped vehicle project using Particle Filters-Udacity's Self-driving Car Nanodegree*. Retrieved from Medium: https://medium.com/intro-to-artificial-intelligence/kidnapped-vehicle-project-using-particle-filters-udacitys-self-driving-car-nanodegree-aa1d37c40d49

Kenton, W. (2021, 7). *Monte Carlo Simulation*. Retrieved from Investopedia: https://www.investopedia.com/terms/m/montecarlosimulation.asp

Madrigal, C. (2012, 06). *resampling-wheel-algorithm*. Retrieved from calebmadrigal.com: https://calebmadrigal.com/resampling-wheel-algorithm/

Nyugen, V., & Im, N. (2015). The Interpolation Method for the missing AIS Data of Shi. *Journal of Navigation and Port Research*, 377-384.

Srinivasan, S. (2019, 8). *Particle Filter : A hero in the world of Non-Linearity and Non-Gaussian*. Retrieved from TowardsDataScience: https://towardsdatascience.com/particle-filter-a-hero-in-the-world-of-non-linearity-and-non-gaussian-6d8947f4a3dc

vesselfinder. (2021). *SEA-FALCON-17-IMO-0-MMSI-563071630*. Retrieved from vesselfinder.com: https://www.vesselfinder.com/vessels/SEA-FALCON-17-IMO-0-MMSI-563071630