Kalman Filter Lab Write-Up

by Edward Ekstrom and Spencer Weight

Declaration of Time Spent

<ins>Edward Ekstrom</ins>

- Implemented
    - Kalman Filter Implementation
    - Targeting Implementation
- Wrote these sections of the report
    - Design Decisions - Kalman Filter and Targeting
    - Testing Against Other Team's Pigeons

Edward spent <ins>11</ins> hours on the lab

<ins>Spencer Weight</ins>

- Implemented the stationary, conforming, non-conforming, and non-Kalman shooting tank
- Wrote these sections of the report
    - Design Decisions - Pigeons
    - Stationary Agent
    - Conforming Agent
    - Non-Conforming Agent

Spencer spent <ins>6</ins> hours on the lab

Design Decisions

Pigeons

The stationary agent was designed to move to a spot and stay there. We selected the origin coordinates (0,0) to be the spot where the stationary agent would stop. The movement of the tank was modeled after the agent0.py movement until the tank stopped.

The conforming agent was designed to do just a vertical line which would be perpendicular to the shooting tank. We felt that a line perpendicular to the shooting tank would be the hardest to shoot because the shooting tank would be forced into rotating as fast as possible to keep up. This would provide a decent challenge and a great example of how well our Kalman filter was working.

The non-conforming agent was designed to travel to random coordinates in a 400 by 400 box that has each side bordered by the bases. The agent chooses a random goal in that box and then moves towards it using the moving function from agent0.py. The tank tries to outwit the shooting tank implemented with the Kalman filter by using a random pattern of coordinates chosen for each goal. We decided that it was best to keep the non-conforming agent at a speed of 1.0. This allows for the tank to make some wide, arcing turns that can confuse the Kalman filter as to where the tank will be.

Shooting Tank

For the sake of experimenting and having a control to experiment with, the initial shooting tank algorithm was modeled after the attack_enemies function provided in the agent0.py file. This version of the shooting tank performed well against the stationary tank, but would mostly score lucky shots on the conforming agent and the non-conforming agent.

Kalman Filter

The design of our Kalman filter changed several times as we tested it against our pigeons. We originally set it up just like the lab description shows but were not able to hit our targets. We decided to edit some of the values in our matrices to see if that would help. First off we had to change the mu matrix to reflect where the enemy tanks base was. So we changed the y value from 0 to 200 since that was the center of the enemy base.

Our next change was to stop using a constant time for dt in the F matrix. Since the time for each tick varies (and therefore the change in time since the last update also varies), it did not make sense to have a constant F matrix like the lab spec says to have. So our solution was to take the time since the last tick and us that as the dt value in the F matrix.

Next, we made some insignificant changes to the initial SigmaT matrix. We do not even know if those changes made a difference. It does not seem that they did.

After all of the changes we made to the Kalman filter, our agent was doing much better. So next we set out to improve our targeting implementation.

Targeting

Originally, we were doing exactly what the spec says, namely to use F matrix multiplied by Mu_t to get the predicted X and Y values. This, however, does not work very well since the dt in the F matrix does not match how long it will take for the bullet to get to where the target tank is going. So to improve this, we used geometry to get an estimate of how long the bullet will be traveling before it gets to where the it would meet the tank. We then set dt in the F matrix to that value and matrix multiplied it by Mu_t and took the x and y coordinates out of the resulting matrix. This change yielded the best results in accuracy.

Now that our agent was hitting our pigeons consistently, there was only one last change to make. We found that if our agent did not hit the pigeon pretty quickly, that it would start missing over and over again by a small margin. We hypothesized that this was because our Kalman filter would become overly confident in its estimation after a lot of readings. To combat this over confidence, we decided to reset out SigmaT matrix after 60 seconds of not hitting our target. This change made our agent significantly better because now if it did not hit the target right away, it was not doomed to an eternity of barely missing its target.

Stationary Agent

Since the stationary agent initially began at or around the coordinates of it's home base, it would have to move itself over to the origin before it would stand still and wait to be shot. The Kalman filter that the shooting tank uses accounts for all movements of the tank. Thus, the initial exodus of the stationary tank to its waiting spot causes the Kalman filter to become confused and so the shooting tank will fire a shot at where the stationary tank was. After the stationary tank arrives at its waiting spot, the

Kalman filter has to catch up a bit. With the addition of the the noise, the shooting tank takes up to 5 shots before finally honing in on where the stationary tank is.

The original shooting tank agent was based on the attack_enemies function provided in the agent0.py file. This version of the shooting tank performed better at hitting the stationary tank.

Conforming Agent

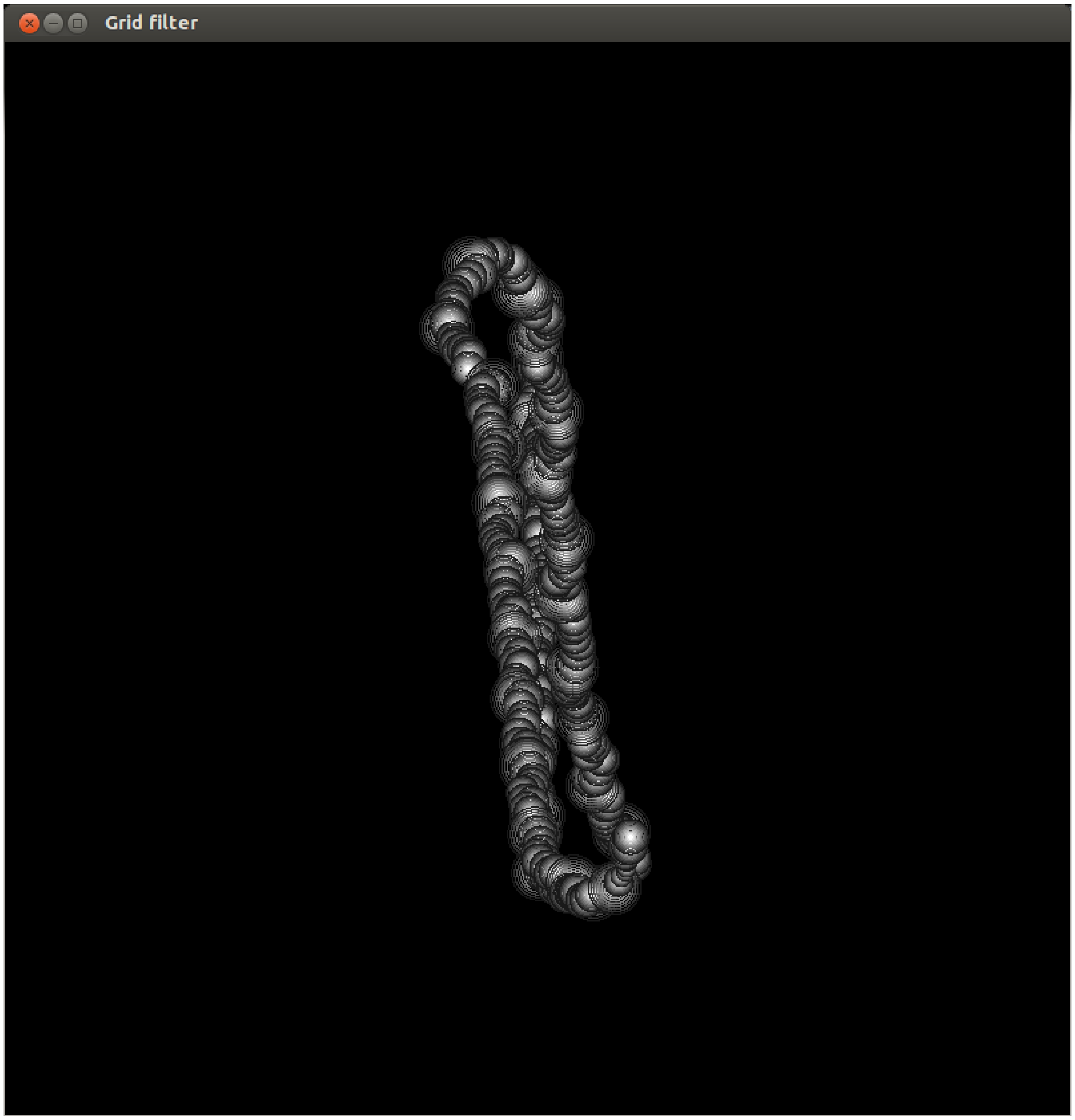
Building

The conforming agent was created by copying the agent0.py code and modifying it so that the agent would only move between two points on the map in a back and forth fashion. The agent has no need to shoot at other tanks, so the attack_enemies code was removed. The speed of the tank was set to a permanent 1.0 to make hitting it more difficult.

Hitting

The shooting tank armed with the Kalman filter algorithm manages to hit the conforming tank after an average of 8 shots. Our conforming tank follows a straight line about 70% of the time. The other 30% is when the tank has to turn around in wide U-turns. The Kalman filter eventually adapts to this behavior and it was rare that we ever saw the Kalman filter ever take more than 10 shots to stop the conforming agent.

Here is an image showing our agent hitting the conforming pigeon:



At each reading it draws a ring around the estimated position with darker shades of gray indicating that it is possible but less likely that the tank is there, and white indicating that it is most likely there. The pigeon was shot at the bottom of its loop as it started heading back to the top for the second time.

In a later image you will be able to see that the ring has a larger radius at its origin. Since at first the Kalman filter has few readings, the variance is greater making the circle wider.

We came with another interesting test to determine the quality of our Kalman filter and targeting implementations. Since we would reset our SigmaT matrix after 60 seconds if it hadn't hit the tank yet, we thought it would be interesting to see how many times we would have to reset in the span of 10 minutes. In the span of 10 minutes, we never had to reset our SigmaT matrix. That means it took less than 60 seconds to hit the target every time.

This was a good indication that our Kalman filter and targeting implementations were high quality. To make our final conclusion about our implementations, we first had to see how the same test did against our non-conforming Agent.

Non-Conforming Agent

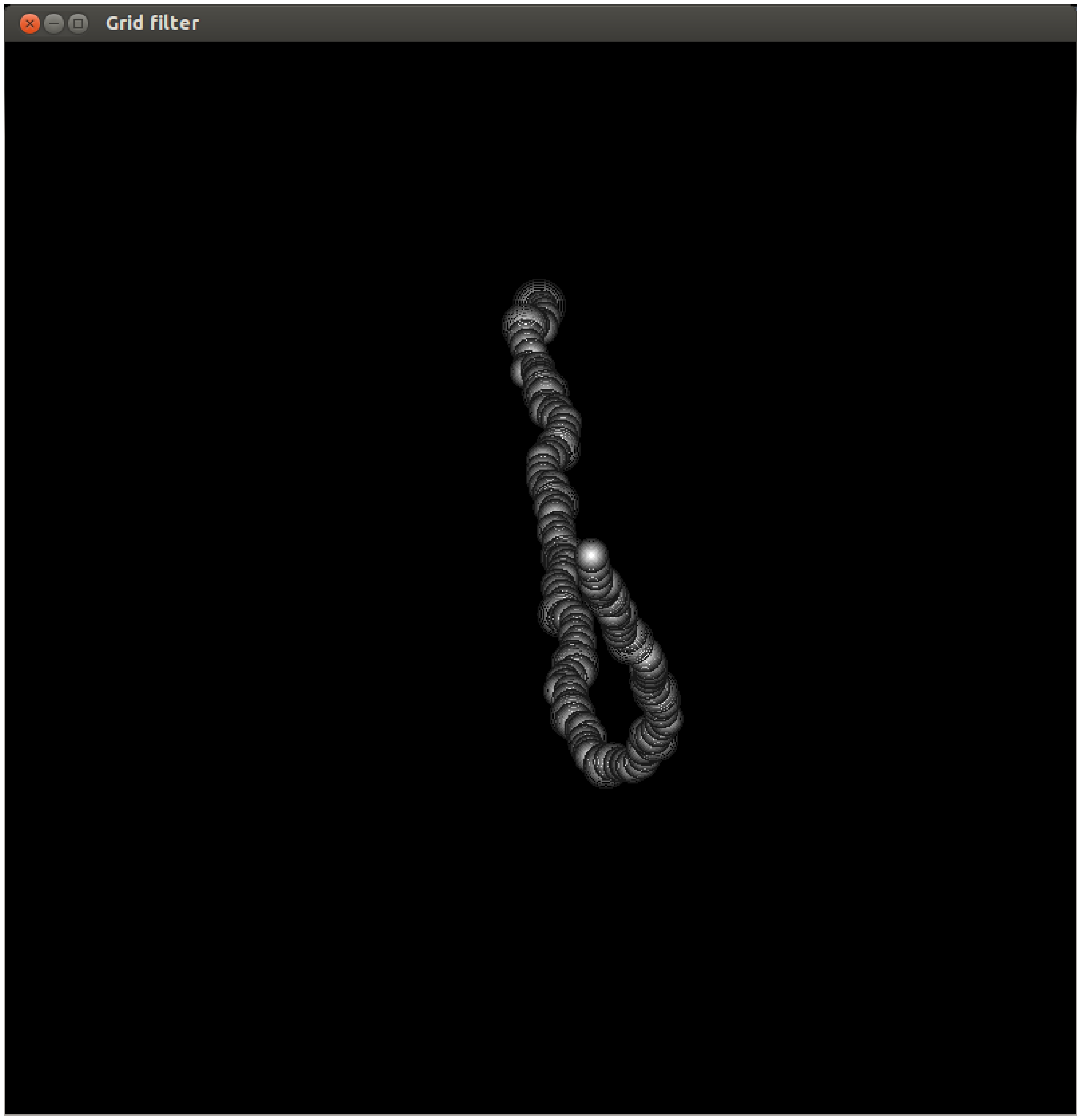
Building

The conforming agent builds off a lot of the same code as agent0.py and the conforming agent, but instead of only having two points to traverse between, it randomly picks the next point to traverse too. This tank doesn't have a need to fight other tanks, so the attack_enemies code was removed. The speed of the tank is set at a constant 1.0 to keep the tank moving fast enough make quick turns and avoid the tracking of the Kalman filter.

Hitting

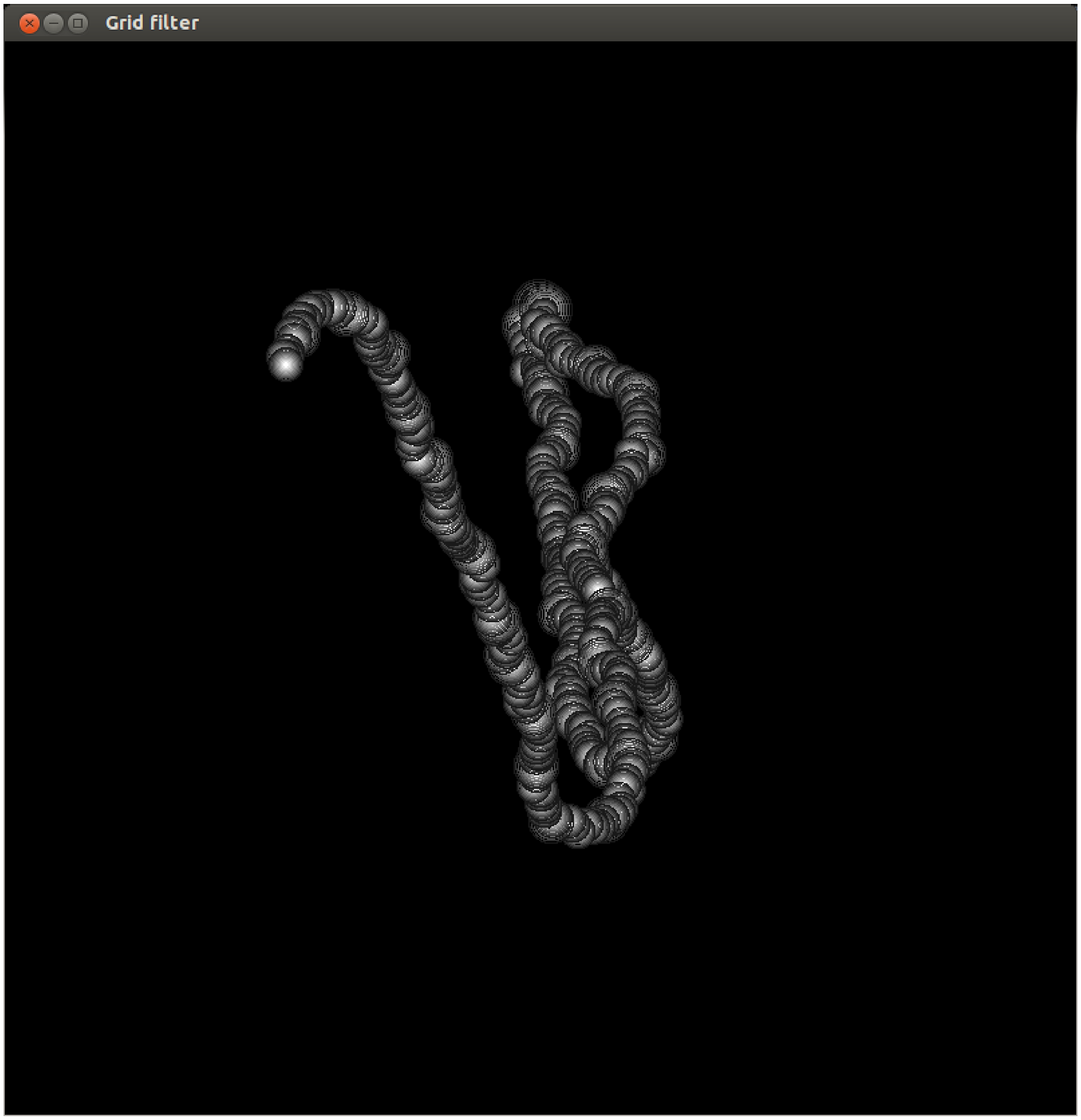
After some initial testing, it turns out that our non-conforming agent is actually quite predictable for the Kalman filter. The Kalman filter was able to hit the non-conforming agent after about 5 shots instead of 10. This supports our assumption that a perpendicular line is better at avoiding the Kalman filter than several diagonal lines. The behavior of the non-conforming agent that made it the easiest to hit was when the non-conforming agent would choose a goal near to where the shooting tank was stationed. This close proximity to the shooting tank allowed the Kalman filter to have more room for error and an easier time shooting the non-conforming agent. The non-conforming tank was hardest to hit when it was as far as possible from the shooting tank. Seeing as the range of the shooting tank is only 350 units, it is clearly the best place for the non-conforming agent to reside if it wants to avoid the Kalman filter powered shooting tank. When the non-conforming tank was in range of the shooting tank, being far away was still beneficial as it forced the Kalman filter to have to spend more time narrowing down the correct angle to shoot at.

Here is an image showing our agent hitting the non-conforming agent including the variance around where the Kalman filter is estimating it is:



At each reading it draws a ring around the estimated position with darker shades of gray indicating that it is possible but less likely that the tank is there, and white indicating that it is most likely there. As you can see, at the pigeons starting position the ring is larger but after a many readings the ring's radius decreases significantly. This is due to the variance decreasing as the Kalman filter gets more readings.

Here is an image showing our agent hitting the pigeon once again after it regenerates:



As you can see, it took our agent longer to hit the non-conforming pigeon this time. If you look closely, you can also see that our SigmaT matrix is reset when the pigeon regenerates and therefore the circle at the beginning of the path once again has a larger radius. The path is being drawn over the path of the previous pigeon that we shot.

To make a final decision about the quality of our Kalman filter and Targeting implementations, we ran the 10 minute test again against our non-conforming agent. This time we had to reset our

SigmaT matrix twice. Once again, this means that in 10 minutes, we went 60 seconds without hitting our target 2 times. This does not seem to be an indication of poor quality since during the other 8 minutes, we hit the target at least during each of those 8 minutes. Therefore, we conclude that our Kalman filter and targeting implementations are high quality.

Testing Against Other Team's Pigeons

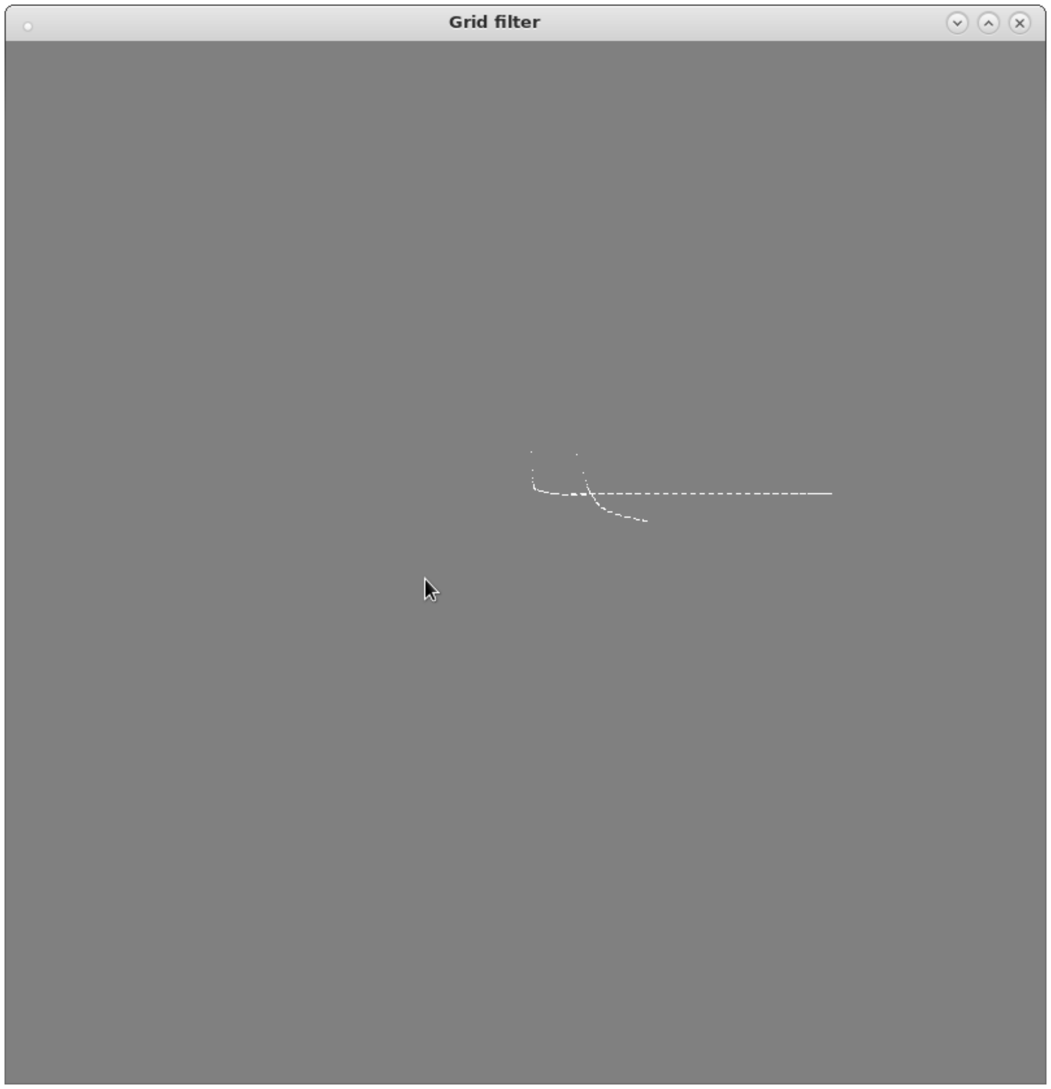
We tested our agent against the pigeons of Matt Nielson and Alex Lemon.

<u>Conforming</u>

Their conforming pigeon was significantly different than ours. Instead of turning around in a U-Turn fashion, theirs would just go in reverse. Another difference was that their conforming pigeon would make a line further away than our pigeon and their line was not exactly tangential to an imaginary sphere around our agent.

At first, our agent was having trouble hitting their conforming pigeon. After about 20 shots, it still had not hit their pigeon. Almost all of our agents bullets were a tiny bit in front of their pigeon. So I decided to play around with the friction constant C. It was set at zero for our pigeons but since I was shooting it in front of their pigeon I set it to -0.01. As soon as I changed that constant, we started hitting their conforming pigeon over and over again with only 3-5 shots each time.

Here is an image showing how their pigeon would move.



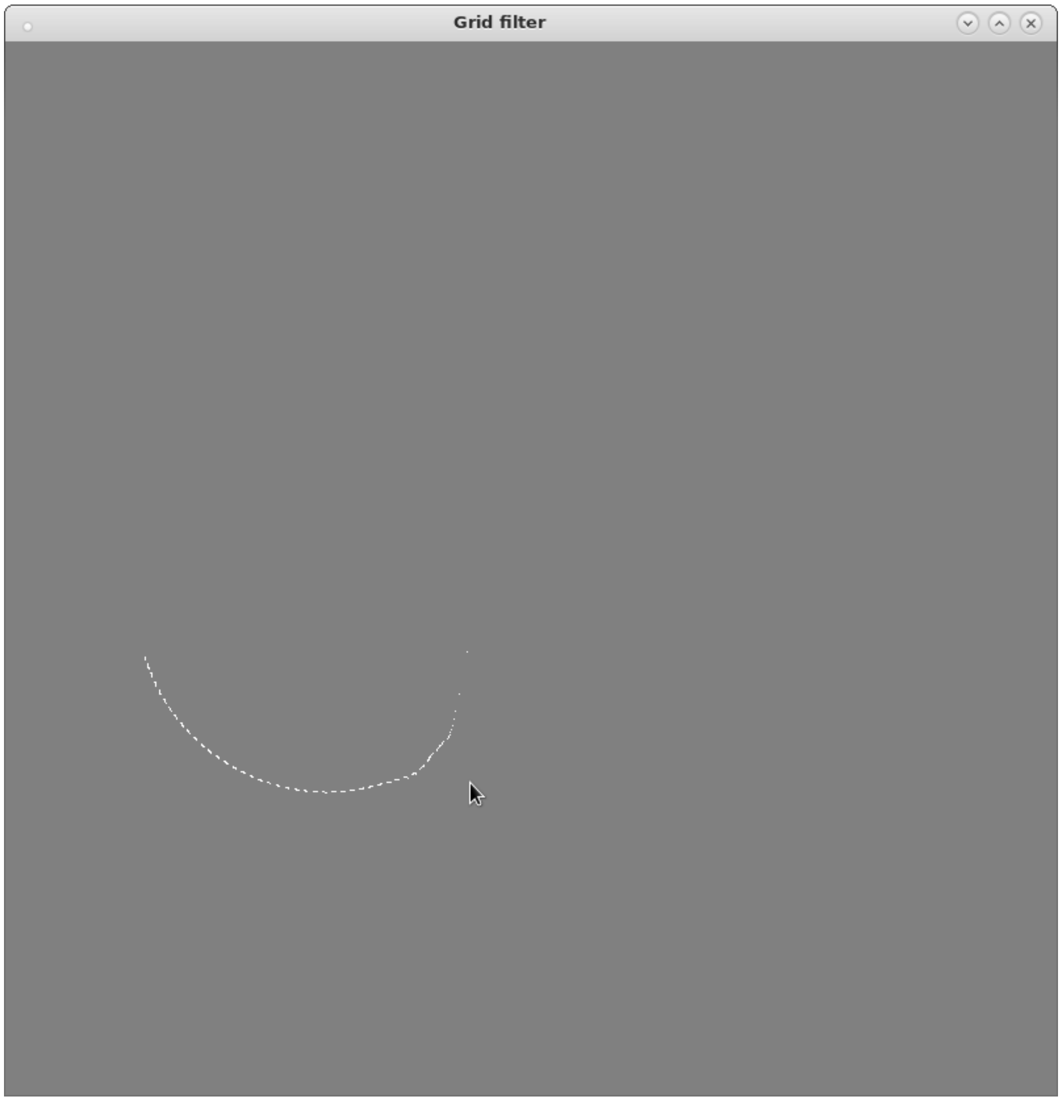
It shows one line that it was making until our agent shot it, then another line after it regenerated and started moving again. Then our agent shot it again.

Non-Conforming

Their non-conforming pigeon was also significantly different than ours. It would drive circles around our agent instead of acting randomly like ours. Right when we started the simulation our agent hit their pigeon, before it could even start driving circles around it. After it regenerated, it made it to our agent and started driving circles around it. Once it started driving circles around it, all of the shots were a little bit behind their pigeon.

Since we were shooting consistently behind, I decided to set our C constant back to 0. Once I did this, our agent would consistently hit their pigeon even as it was circling our agent.

Here is an image showing how their pigeon beginning to drive a circle around our agent but being hit about half way around:



Conclusions of Testing Against Other Team's Pigeons

We concluded from our tests against the other team's pigeons that our Kalman filter and targeting implementations were very high quality. We also concluded that for future work it would be good to asses how the agent was missing the target and adjust the friction constant accordingly.

Feedback

<u>Edward</u>

I thought this lab was okay. It was very frustrating that the lab spec mislead us multiple times, most notably stating that the F matrix should be constant. Another frustrating thing was that the lab spec said to use F matrix multiplied by Mu_t to project where the target will be but it fails to mention that you need to first calculate how long the bullet will take then put that value as dt in the F matrix before that will work.

<u>Spencer</u>

I think that this lab was a great lab to do. It helped me learn more about how to better predict where an agent will be if I want to hit it. I feel that this lab had some real world applications that I could use later in my life as a programmer.