Lab 4 (Kalman) Write-Up

Brooke Maxwell, Ian Baenziger

Time

Ian Baenziger- 8.5- building the filter, modifying existing code, tweaking, writing report

Brooke Maxwell- spent 8 hours. Filter visualization, coding pigeons, tweaking, writing report

Kalman Filter

We implemented the Kalman filter based off its description in the book as well as the instructions in the assignment specification. We did make some modifications to the numbers provided in the assignment specification.

Our initial state vector (μ_t) assumed the target tank started near the enemy base with no velocity or acceleration. Our initial covariance matrix (Σ_T) asserted that we were confident that the tank would not begin moving, by setting the velocity and accelerations standard deviation to .1, but not as confident of the position, with the position standard deviation set to 25. Perhaps the most important design decision we made was modifying the *F* matrix. The *F* matrix is the matrix that is supposed to represent how the server would update the tank's state. As it is used in predicting the next location of the enemy tank, it was extremely crucial to have an accurate *F* matrix for targeting. This is what we used for our f matrix:

```
1,  4, 2,  0,  0, 0
0,  1, 4,  0,  0, 0
0,  0, 1,  0,  0, 0
0,  0, 0,  1,  4, 2
0,  0, 0,  0,  1, 4
0,  0, 0,  0,  0, 1
```

The top row represents that we expected the next x position would be the current x position plus four times velocity in the x direction plus two times acceleration in the x direction. We set our time interval at four seconds (why we did so is explained in the next section) which explains why we would multiply our velocity by four to add to the final x prediction. In an environment with perfect physics, the acceleration value should be equal to $\Delta t^2/2$. However, our environment limited a tank's final velocity, meaning that the full force of acceleration would not be able to influence the tank. For this matrix and all other considerations of acceleration, we choose to limit the effect of acceleration on our final prediction.

We also adjusted the numbers of the uncertainty matrix. This is our Σ_x matrix:

```
25, 0, 0,   0,  0, 0
0,  5, 0,   0,  0, 0
0,  0, .2, 0,  0, 0
0,  0, 0,  25,  0, 0
0,  0, 0,   0, 5, 0
0,  0, 0,   0,  0, .2
```

The values along the diagonal represent what we expected the standard deviation of our uncertainty for each value in the state matrix. The first value represents that we expected a 25 pixel standard deviation of our uncertainty for our x position. The number 25 was based off of receiving a position noise of 5 from the server. The rest of the values were based off of trial and error and estimation.

Targeting

Through empirical testing, we noticed the tank is only able to fire every four seconds. We based our targeting cycle on this observation. The first part of the cycle involved passing the target readings to the Kalman filter. Then we would use the updated filter to attempt to predict the next position of the target. The next four seconds would be devoted to turning the tank to the predicted location. When the four seconds were up and the tank could fire again then a shot would be fired, the position reading taken again to update the filter, and the filter would then predict the next target position. The next four seconds would turn the tank to this new position and the cycle would repeat.

This produced some surprising effects. We originally believed it would be best to update the Kalman filter at every tick, producing a filter with extremely accurate predictions. Empirically, only updating the filter every four seconds resulted in a more accurate tank. We suppose that updating the filter every tick overfit the prediction and did not allow enough slack for unexpected issues (such as small errors in the tank controller rotation and aiming, or the lack of factoring bullet speed).

The above paragraph illustrates a conscience effort we undertook as we tweaked the filter: we wanted some leverage for “fudge” factors. For example, it was difficult to get the tank to land exactly at the correct angle to fire to the exact position. Instead of trying to program a perfectly aimed tank, we accepted that there would always be some error due to the tank angle being slightly off. Also, we did not calculate, record, or factor in the speed or distance traveled of the bullet before reaching the predicted position. Keeping our kalman filter predictions just slightly loose allowed us to not devote excessive amounts of time to tweaking while still producing impressive results.

The Pigeons

Building the Pigeons

The empty world we used allowed the pigeons to be out of range of our tank, so we first had to make sure they began in range. We rotated the pigeon until it pointed to an angle within .5 radians of the center, then moved it forward until it was within a range of 300 from the targeting tank. After that, each tank had a different behavior.

Our stationary tank would of course stop and wait to be hit. Our conforming pigeon would straighten out and drive at a constant (randomly generated) speed until it hit a wall or was destroyed. The nonconforming pigeon would move while turning at a slow speed for four seconds, then speed up and turn in the opposite direction for five seconds and repeat endlessly until hit. This gave it a weaving path that made it hard to track, since it was changing velocity and direction constantly.

Hitting Our pigeons

We allowed for a slight time delay in order to make sure the pigeon was within range, and only shot once it had stabilized within its motion or lack thereof. We also shot every four seconds, long enough to reload the shots, but without checking to make sure we were within a likely range of hitting the pigeon. We chiefly used the “quantity over quality” method to give ourselves a better chance of hitting the pigeon.

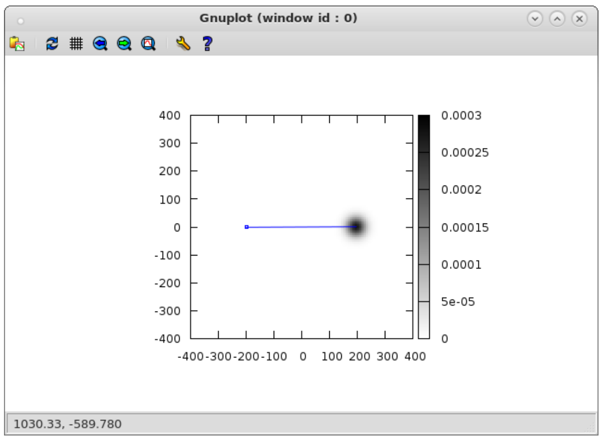
Stationary Pigeon

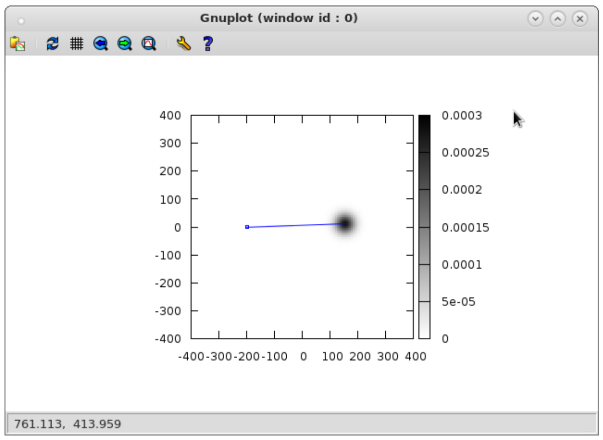
The stationary tank was trivial to hit. We were able to destroy it within a couple of shots every time even with our looser implementation of the filter. Our filter did not narrow down much, as explained in the implementation of the filter above, but that did not inhibit its ability. We directed our tank at the target location of the tank, which was the center of the density filter, and that was sufficient to successfully destroy the tank.

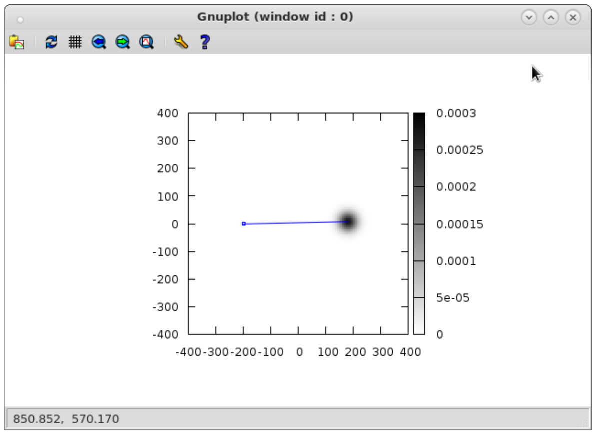
Conforming Pigeon

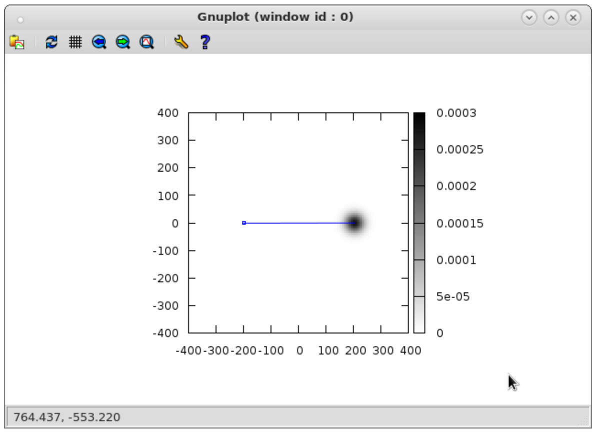
We were able to destroy our conforming tank consistently within 4 or 5 shots (about 20 seconds). We tracked the pigeon as with the stationary one, using its angle relative to our shooter, so we were able to stay within a close range of it as we tracked, giving us several close chances to hit it.

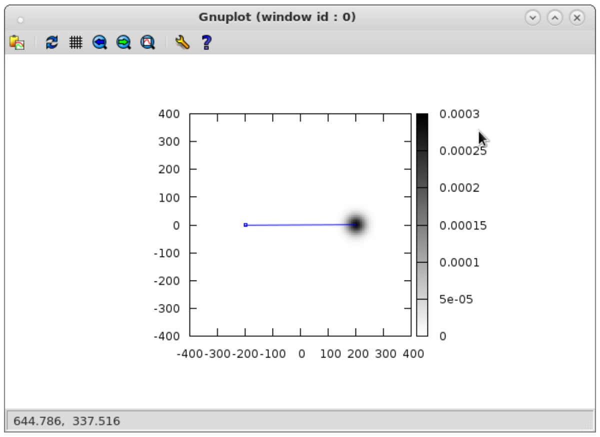
Below are the plots for tracking our conforming agent. It is hard to tell, but the pigeon is approaching the shooter slowly. We destroyed the pigeon in a total of four shots, with the final frame showing the reset to home caused by the pigeon’s death.

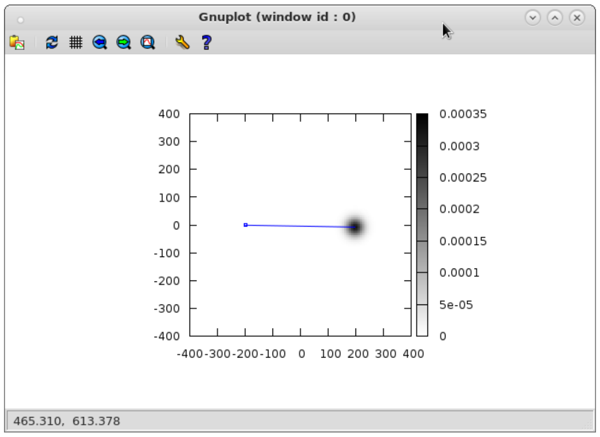








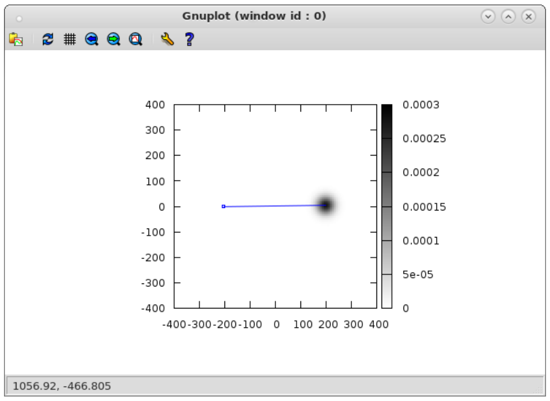


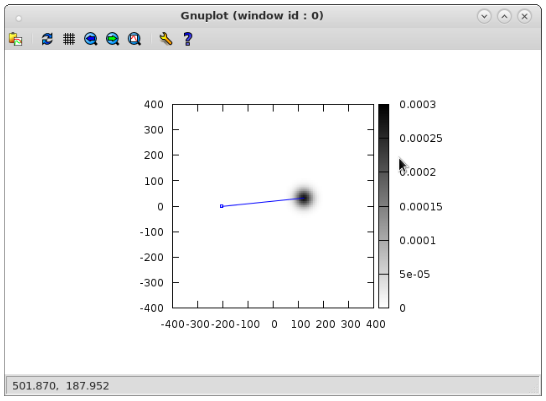


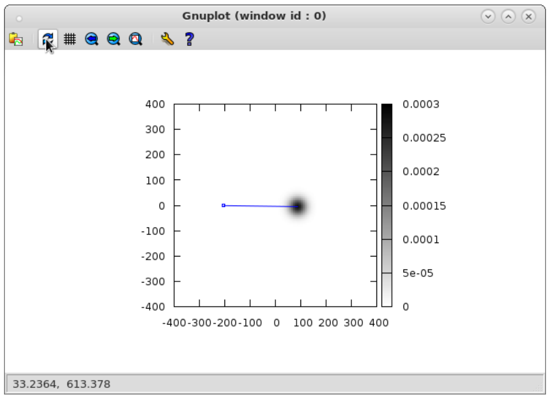
Non-Conforming Pigeon

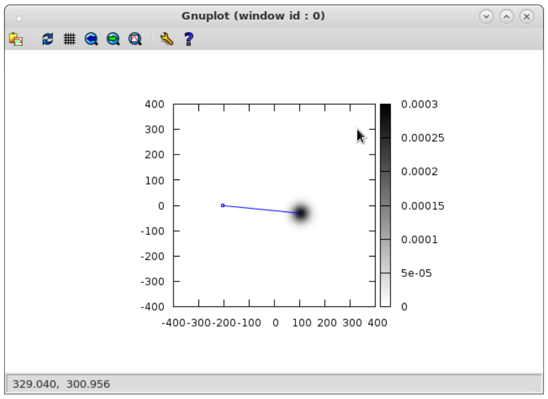
Our non-conforming tank was much less accurate. We were able to track it, but without consistency in hitting it. We were able to hit it eventually nearly always, but it took anywhere from five to twelve shots (between 20 and 50 seconds) to get success. It was harder to track given its variance, but we did surprisingly well at following it given the variance.

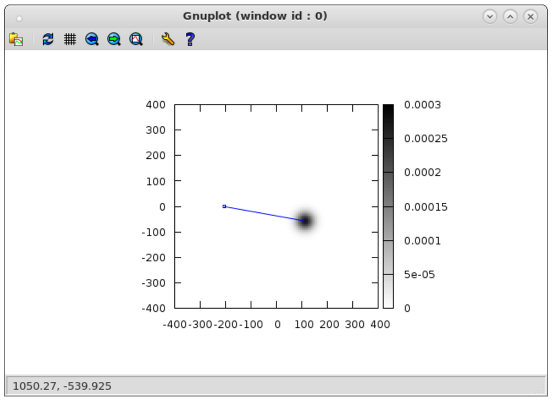
Below are plots for the last four shots taken at our nonconforming agent. As mentioned above, the filter only updates only every four seconds, which is why our graph densities barely decrease. We destroyed the pigeon in eleven shots, after a total of 48 seconds. The final frame shows the reset to home after the pigeon's death.

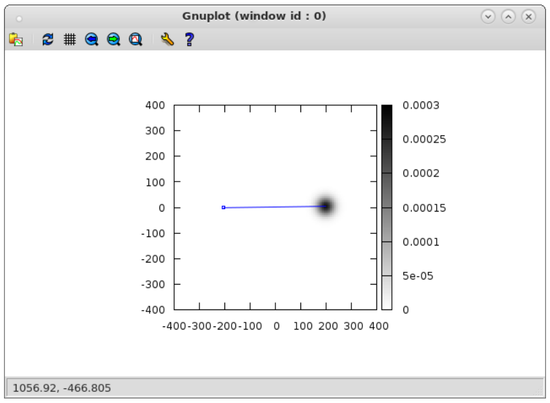












An Observation about Pigeons

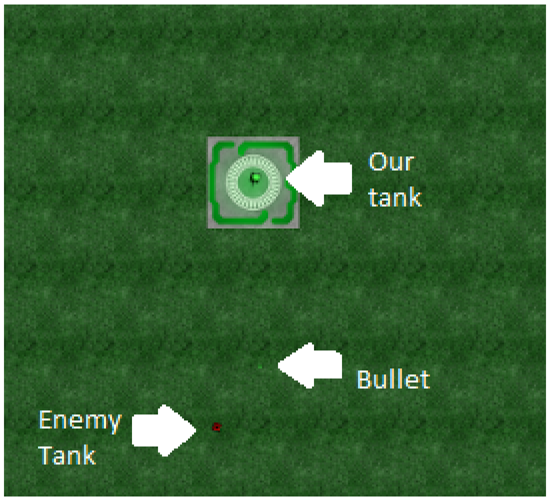
The algorithms for the pigeons were fairly simple, but we struggled to debug what seemed like erratic behavior for a while. Our tanks would just turn endlessly, without ever moving to within range and following their prescribed behavior. But then they would behave as expected. We discovered that debugging print statements interfered with the readings the pigeons received from the server or some related thing, because the erratic behavior only occurred when printing to the terminal. That meant that a few times we tricked ourselves into thinking it was broken, and spent unnecessary time trying to fix them.

Hitting Another Group's Pigeons

We tested our targeting against the conforming and nonconforming pigeons of Dustin and Brad. Our targeting agent was quite successful in hitting their conforming agent but was dominated and unable to hit their cleverly conceived nonconforming agent.

We ran our targeting agent against their conforming agent in two different scenarios. The first scenario was both tanks at starting positions. As their tank began to move, it took fewer than five shots to hit their non-conforming agent. We were pleased that our tank was successful even starting on the opposite side field than we tested. On our second run, we gave their tank a 30 second head start before activating our shooter. This caused difficulty for our agent. Their tank was near the edge of the map on our side when our agent began targeting, and our first few shots were erratic. We suspect that this was because our fitler expected their tank to start at the opposite base. After about 12-15 shoots, we hit their tank traveling backwards about 200 pixels north and 100 pixels west of our tank. This was another pleasing result as our tank was able to hit a target which switched direction in a predictable way, a situation we never before tested.

Against their non-conforming pigeon, our agent was far less successful. Their pigeon rotated in an orbit at full speed around our tank, so the target's velocity was constantly changing, which created difficulty with our filter's heavily reliance on a fairly constant velocity (see our update/*F* matrix). Even our loose implementation of the filter was not enough to offset our reliance on a constant velocity. The bullets did get closer, but after two minutes it was clear our agent could not hit this target. The photo below is of a bullet missing the orbiting tank. This shot is of one of our poorer attempts (we promise!). Our agent perhaps might have excelled at this task had we implemented our filter in polar coordinates, which would have treated the rotation as constant acceleration towards the center.



Summary
Again, the most difficult part of the lab was tweaking, and in debugging the peripheral elements. The filter took some careful consideration to implement it the way we wanted, but the fine-tuning took the most effort.