A6 - CS 4300

Fall 2016
Leland Stenquist - 1,3,5
Ryan Keepers - 2,4,6

1. Introduction

In the assignment we explored the power of the Kalman Filter. The Kalman Filter is an algorithm which takes in two models and uses them to try and get them to represent a real world model. In this example we have the model from a sensor combined with a trace(estimate) model. These two try to model a line for the function CS_4300_A5_driv_lin. For the function CS4300_driver_proj they try to model a projectile.

In this experiment there are a few question that we would like to answer:

1. Does the Kalman filter model a line well?
2. Does the Kalman filter model a projectile well?
3. Would the Kalman function model more complex systems well?
4. If the trace model starts with no velocity, is it, using the Kalman Filter, able to approximately catch up to the actual model?

2. Method

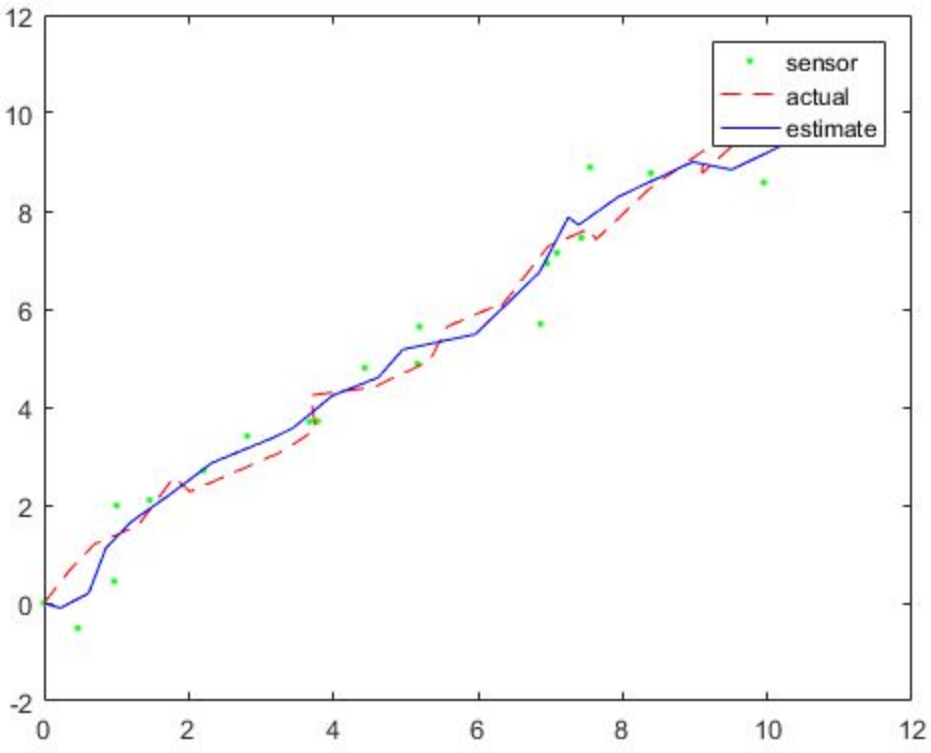
Our program utilizes two primary functions. First, CS4300_A5_driver_lin creates the linear estimator. Second, CS4300_driver_proj creates the projectile estimator. These functions operate independently of each other, although they both perform similar tasks. Their two primary differences are that while driver_proj takes gravity into account, driver_lin does not, and driver_lin utilizes size 2x2 matrices (velocity is not recalculated, outside of a bounce, since it is not affected by gravity or noise) while driver_proj utilizes size 4x4 matrices (to account for both position and velocity).

In both cases, the functions operate as follows: First, we initialize any variables that need initialization. Second, a loop is created iterating over the total run of the time, stepping in increments defined by the delta_time variables. On each iteration of the loop the ideal model position is updated based on its “actual” position. Then the actual position is updated by adding noise to the ideal. A reversal of velocity (due to the actual projectile position bouncing) is checked. Then the sensor position is updated by adding a greater amount of noise to the actual. Finally, the Kalman Filter is called to create an estimate of the projectile’s position.

3. Verification of Program

Verification of the program was visual. What we did was run the program with different inputs. We then graphed the results to see if we got what was expected. This verification will be in two steps. First the function, CS4300_A5_driver_lin, will be verified. Then the function, CS4300_driver_proj, will be verified. Our Kalman Filter can also be verified by looking at the data in these graphs and seeing that they follow certain patterns. For example the the estimated velocity should always have to catch up to the actual velocity. The graphs should demonstrate this.

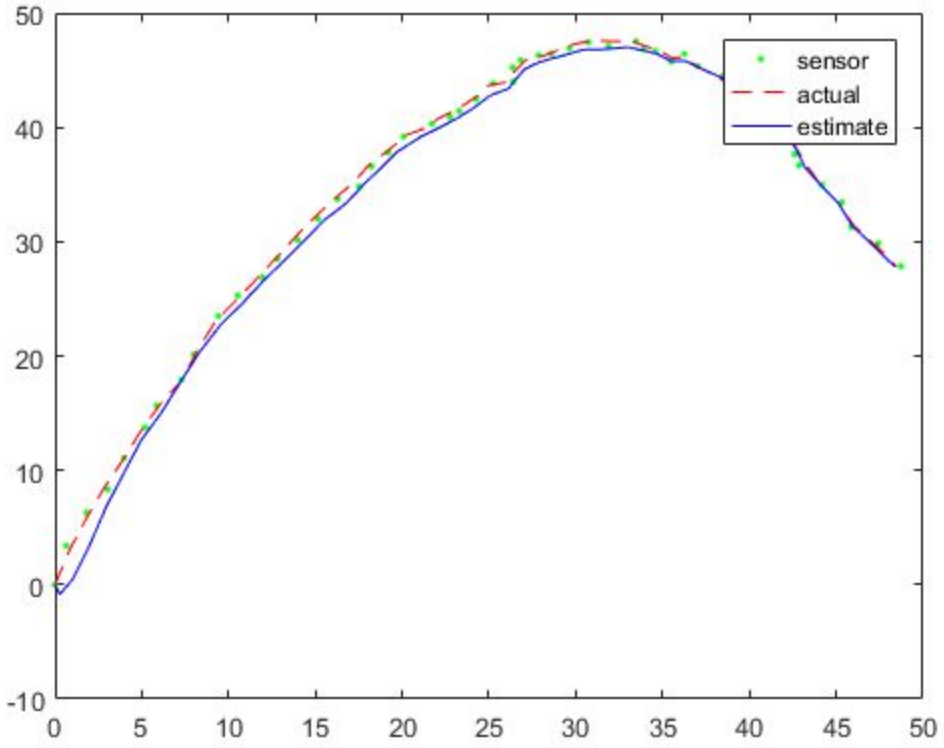
CS4300_A5_driver_lin Verification



Plot 1

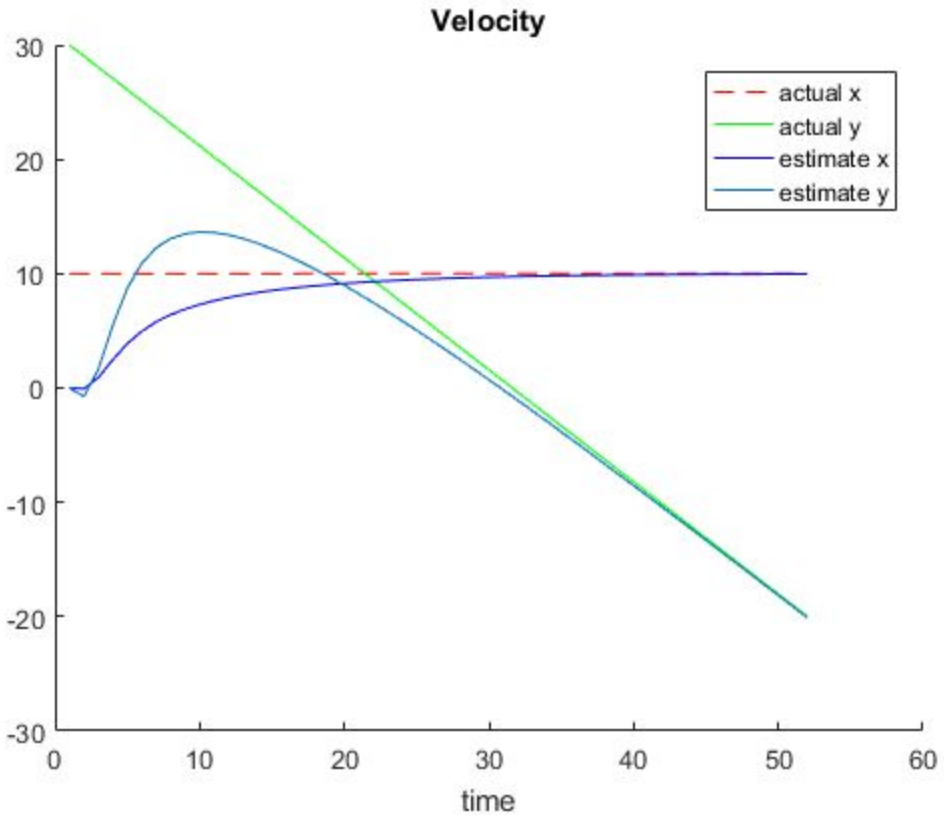
Plot 1 verifies that the function creates three models that are linear in nature. The estimate drops initially because it has no velocity, but using the Kalman filter it begins to climb and find the actual.

CS4300_driver_proj Verification



Plot 2

Plot two verifies that the three models are both parabolic in nature. Also in the model you can see that the estimate has to catch up to the sensor and the actual. This is because the estimates starts out with no velocity. If this were not the case we would know that something was wrong with our Kalman Filter and that we needed to revisit our algorithm.

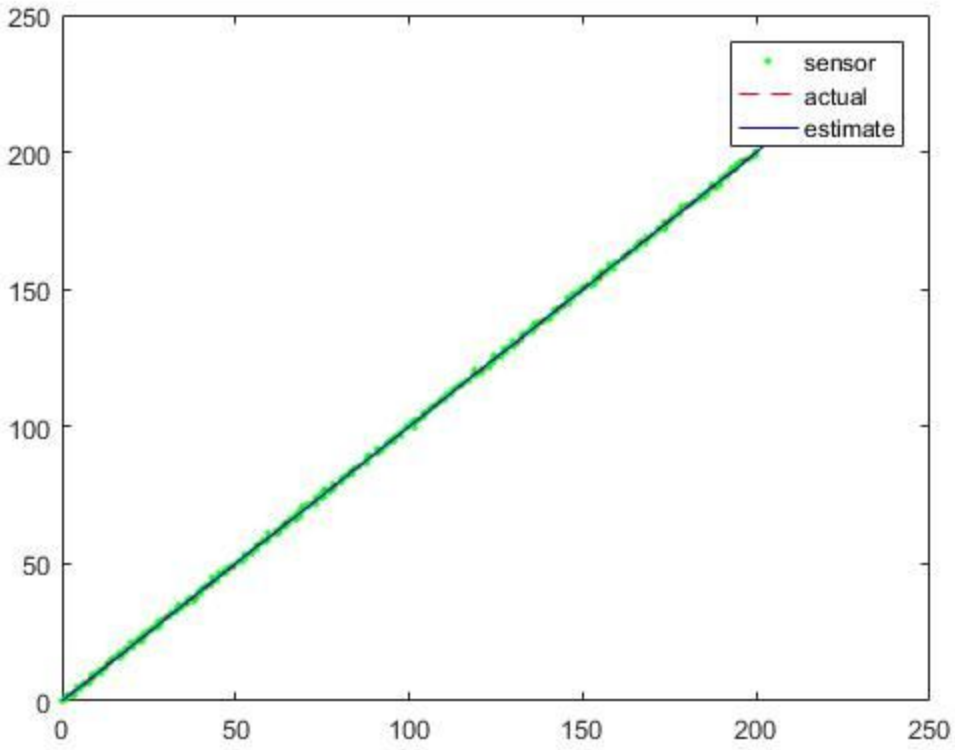


Plot 3

Plot 3 was generated with a very low variance. This was so that it would be easier to interpret. As expected the estimate velocities all start at zero. They should rise, possibly overshooting and then find the actual velocities. In this plot we are not seeing the overshooting, but we can definitely see the estimated velocities finding the actual velocities as expected.

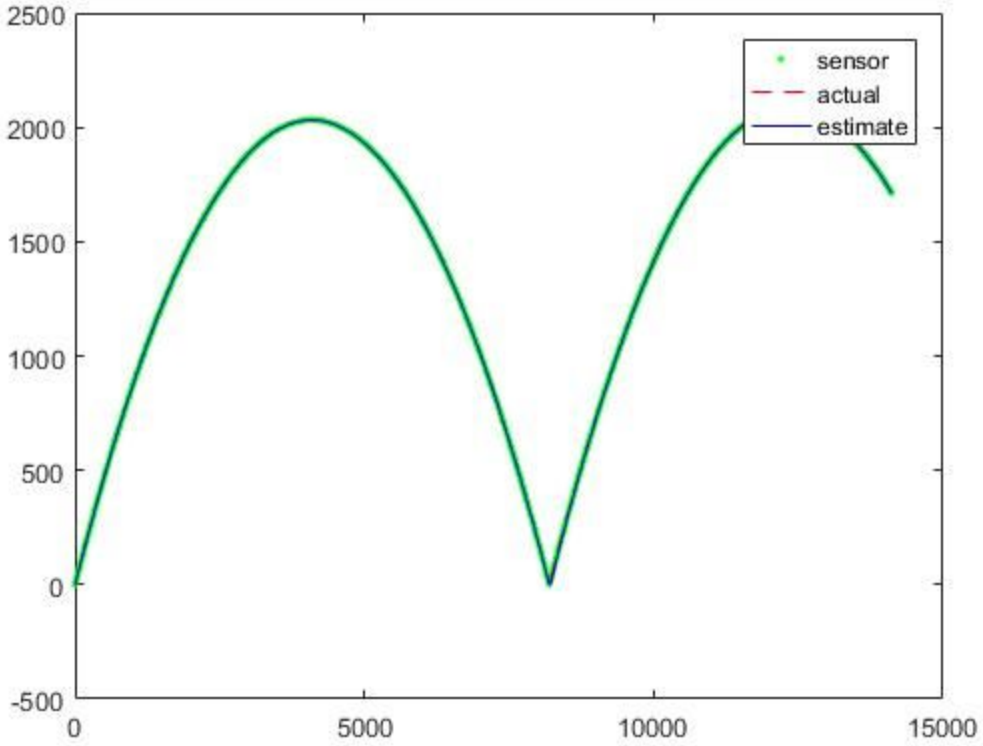
4. Data and Analysis

CS4300_A5_driver_lin
Max Time: 20
Delta Time: 0.05
Sigma2 process: 0.10
Sigma2 sensor: 0.25
Velocities: 10
Gravity: 9.8



Sigma 2 trace 2	Sigma 2 trace 400
0.0012 0 0 0.0012	0.0171 0 0 0.0171

CS4300_driver_proj
Max Time: 70
Delta Time: 0.05
Sigma2 process: 0.10
Sigma2 sensor: 0.25
Velocities: 200
Gravity: 9.8



Sigma 2 trace 2	Sigma 2 trace 400
0.1272 0 0.0172 0 0 0.1275 0 0.0089 0.0172 0 0.6350 0 0 0.0089 0 0.3925	0.1222 0 0.1131 0 0 0.1222 0 0.1131 0.1131 0 2.1606 0 0 0.1131 0 2.1606

5. Interpretation

We are trying to answer a few questions with these experiments. The first two questions we ask are about the ability of the Kalman Filter to at model a line and a parabola. The answer is that it seems to do pretty well. This is especially true, it seems, in situation where the model is run for

longer period of time with a small delta t. For example,if you view either plot in the Data and Analysis section you will notice that the estimate seems to model the actual pretty well. Both of those plots have a very small delta t compared to the time for which they are run.

A very interesting inquiry would be to figure out if, in a more complex system, the Kalman Filter would still work. We do not feel we understand it well enough, or that our experiments were thorough enough to say yes or no. We can say, though, that the Kalman Filter did seem to model our linear and projectile models decently.

The next question was about the trace(estimate) model and whether or not it would be able to catch the actual model when it started with 0 velocity for both x and y. Our experiments show that it was able to catch up. This was extremely apparent in plot 3 of the Verification section.

6. Critique

I think it would be erroneous to say that both of us fully understand the Kalman Filter and all of its details, even now that we're done with it. Though both of us have an understanding of Linear algebra sufficient enough to agree that all of the steps of the process are mathematically sound, the act of coding up the algorithm hasn't brought us closer to being able to relate why it works. Though we have gained an appreciation for how easily matlab handles the actions.

Given that extra time for the assignment was an option, I am currently struggling to decide whether or not an extension would actually help us solve any remaining potential bugs. For that matter, I've found it more difficult this time to even decide whether we have bugs or not. In example, we found that sometimes running the projectile driver with a long time period and low y velocity could create a situation where the projectile gains height at each bounce. But then again, it was stated in class that sometimes the projectile loses height after bounces, and this isn't fully explicable either. Alternatively, we can manipulate the parameters to produce more or less erratic looking line estimations (even without changing the Sigma2 settings for noise). It makes determining the quality of our outcome more difficult to pin down.

In the end, however, I do feel we both came away with an understanding of what the Kalman Filter estimation intends to do, whether or not we got it perfectly correct in this iteration. We understand the role of the sensor in representing real incoming data, versus that of the noisy actual situation, and why an estimation is needed to mediate between the two. If we are to apply the concept again, I suppose it's just a matter of having a stronger grasp of the ins and outs of the environment within which we're applying the algorithm.

7. Log

Ryan : 8 hours
Sections 2, 4, 6

Leland : 8 hours
Sections 1, 3, 5