**Assignment 2 Report**

**NOTE: I submitted the minimal functionality by the deadline. This is my full functionality late submission, submitted 4 days after the deadline. The assignment handout said “You can submit an improved solution within one week of the initial date” but didn’t specify a penalty.**

**NOTE: To reuse code, I extracted my TODO code for detect\_ball() and get\_fan\_rpm() into helper.py**

**Setup**

I used Python 3 and the graphic library PyQt6.

**OpenCV Ball Tracker**

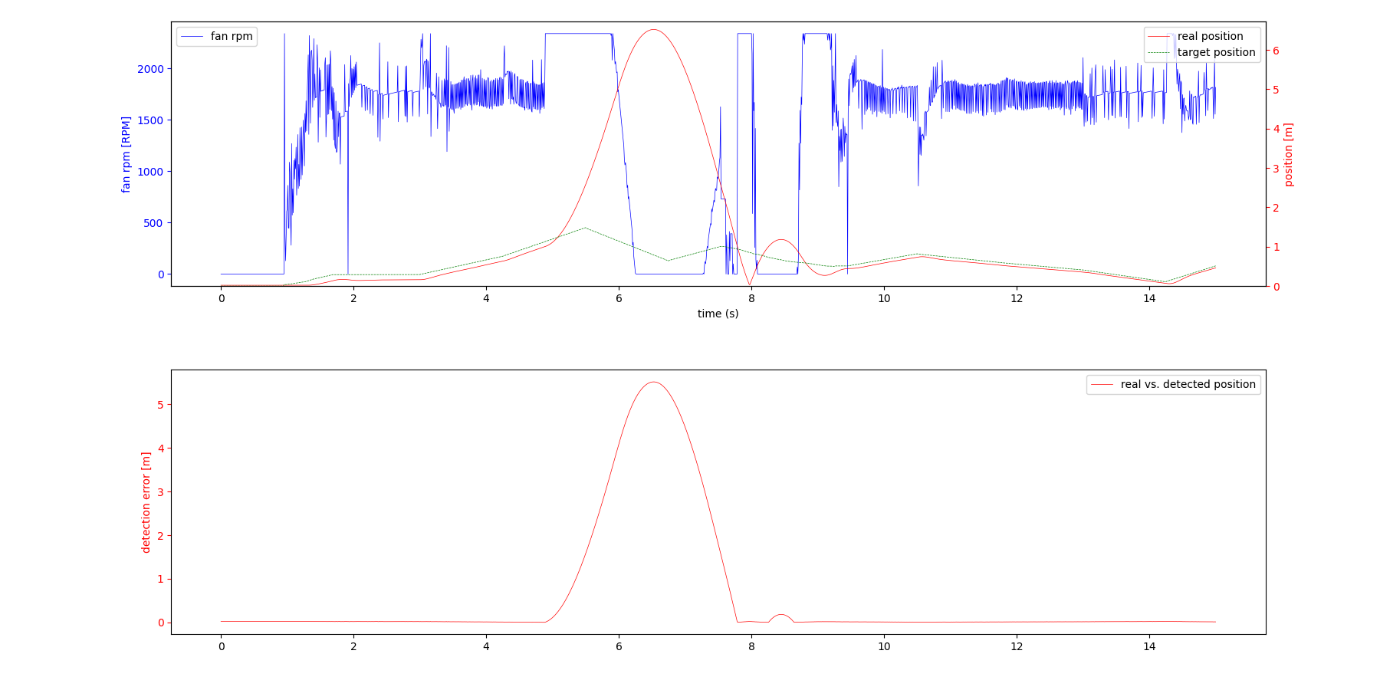
For calculating the PID, I followed the formulas and suggestions in the lectures. Instead of integrating the error term, I approximated by calculating the area average of error height and previous error height times the time elapsed:

Ideally, I would find the sum from a window of time, since I do not want to be considering events that happened 5 minutes ago, but the test does not run for very long, so the accumulation is not too significant.

Instead of differentiating, I just divided the error velocity by the time elapsed:

The given PID constants were good, however, the ball would overreact when a new target was set. So I changed . The downside is that the PID controller takes a little longer to get used to the gravity. Ideally, I’d set the bias to 1725 instead, so that it immediately acclimates to the gravity, but PID does not technically have a bias component, so I didn’t bother with it.

There is an issue if the target is set above 1m. What happens is that the ball can no longer be detected, so the program stops updating the detected ball position. Therefore, it still thinks the detected height is 1m, and with such a wrong assumption, the PID would be way off. See the figure below:



For the noisy mode, I needed to be able to tell in my code whether there is noise in my frame or not. Upon looking at the noiseless and noisy picture, the noiseless frame has a fairly constant mean (when you average out all the values in the matrix), and the noisy frame has a less constant but significantly different mean, due to all the noise. When my code detected the noisy mean, I filtered the image many times using image convolution, then normalized the image, since it lost its color:

A picture containing text

Description automatically generatedA picture containing text

Description automatically generated

After doing that, I was able to detect the ball again. The above method of filtering and normalizing was suggested by the TA. Also, the kernel that I used was a 9x9 array of 1s divided by .

**Kalman Filter Ball Tracker Implementation**

For Kalman filter, my state consists of height and vertical velocity. The previous state has a normal distribution with mean , and the objective is to transform it to one with mean . Afterwards, I transformed it into the best estimate by multiplying its distribution with that of the observed height. Then the best estimate becomes the previous state, and so the cycle continues.

To transform from previous to predicted state ( is gravity, is fan acceleration):

The above variance equation is my way of treating un-tracked influences as noises (in the code I tweaked it a bit to avoid computing square roots).

To transform from predicted to best estimate state with the help of the observed height:

**Experiment Results**

OpenCV without noise:

Graphical user interface, chart, line chart

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Trial | Average detection error (m) | Average steady state error (m) |
| 1 | 0.006458539108196891 | 0.04938537344784299 |
| 2 | 0.016402688681107454 | 0.04750210534261403 |
| 3 | 0.008364370801852214 | 0.0543288634942988 |
| Avg | 0.01040853286 | 0.05040544743 |

OpenCV with noise:

Histogram

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Trial | Average detection error (m) | Average steady state error (m) |
| 1 | 0.018203473417231844 | 0.015484608153760158 |
| 2 | 0.013705019257017685 | 0.04277534250288239 |
| 3 | 0.016595414511756886 | 0.045753631259190554 |
| Avg | 0.01616796906 | 0.03467119397 |

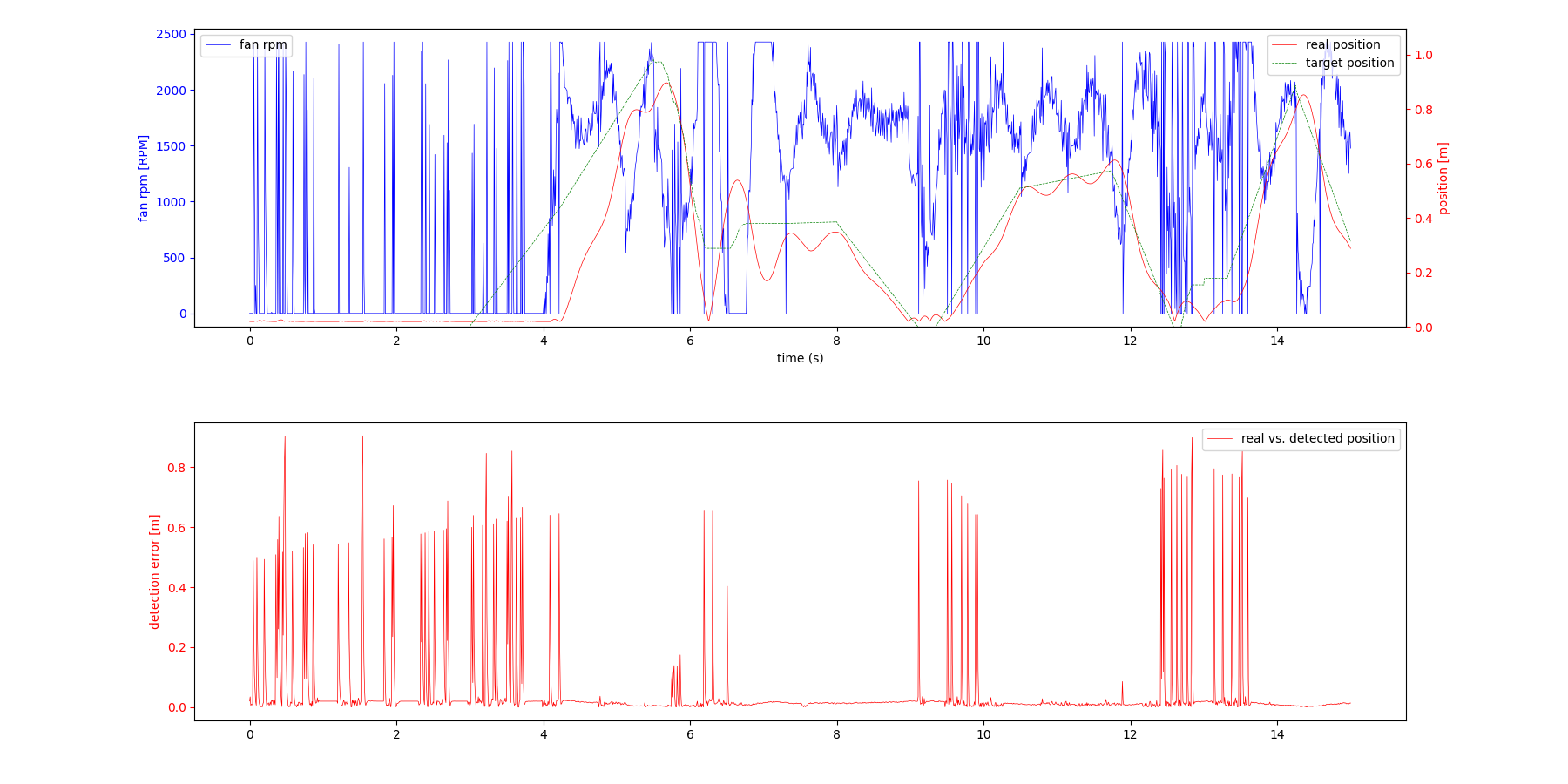
Kalman without noise:

Graphical user interface, chart, line chart

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Trial | Average detection error (m) | Average steady state error (m) |
| 1 | 0.007872551740600387 | 0.003516220369407656 |
| 2 | 0.015391006109141342 | 0.006433877057987004 |
| 3 | 0.012655300171356913 | 0.020460482468201917 |
| Avg | 0.01197295267 | 0.01013685997 |

Kalman with noise:



|  |  |  |
| --- | --- | --- |
| Trial | Average detection error (m) | Average steady state error (m) |
| 1 | 0.024685111924002363 | 0.08159845177537356 |
| 2 | 0.01074141609187581 | 0.02292301790736905 |
| 3 | 0.009675569201283839 | 0.06481824309216615 |
| Avg | 0.01503403241 | 0.05644657092 |

Note: For the noise detection error graphs, I am not sure why the detection error fluctuates at times. It could be a bug, since the average error reported by the console is low. Or there is indeed a very large error, but because it is at such a short amount of time, it does not factor into the average.

**Explanationof Results**

|  |  |  |
| --- | --- | --- |
| Mode, Algorithm | Final Average detection error (m) | Final Average steady state error (m) |
| Noiseless, Open | 0.01040853286 | 0.05040544743 |
| Noisy, Open | 0.01616796906 | 0.03467119397 |
| Noiseless, Kalman | 0.01197295267 | 0.01013685997 |
| Noisy, Kalman | 0.01503403241 | 0.05644657092 |

Between noiseless and noisy, there is about a 50% increase in average detection error. This makes sense of course, since the ball is harder to track with noise.

Between open and Kalman, the results do not say much. This could mean that my Kalman filter could be made more accurate. For example, I did not consider the variances of the different velocities and the observed velocity altogether. It is also possible that my model is inaccurate in a physics perspective. Finally, three trials are too little, since the deviation between trials is very great.