Mohanna Shahrad

Assignment 4: Final Project

COMP 417 - Introduction to Robotics and Intelligent Systems
School of Computer Science

McGill University

The backbone of the code used in this project is taken from the previous assignments. Modifications made specifically for this project will be discussed in the following sections.

1 Visual Tracking

As mentioned above, the same *InvertedPendulumGame* class as assignment 2 is used in this project with slight modifications. The visual tracking part is implemented using added helper functions to *inverted_pendulum.py*. The high-level mechanism is as follows. In the main for loop of the function *game_round*, for each time step, the following function calls are made:

- First the helper function *thresh* is called which performs thresholding on the surface array. The main logic of *thresh* is to put a lower and upper mask on the surface array based on the fact that the cart's color (blue) is different from the pole's color(red). The output of this function is an image array in which all values are zero except for the mask. Therefore, we will have the location of the cart and pole separately.
- Next, another helper function *findCenterOfMass* is used to find the cart's and pole's center of mass. For this, *skimage* is imported and specifically, *measure.regionprops* is used to find the centroid.
- Finally having the centers of masses for both cart and pole, a call to *findAngleBetweenLines* is made to find the angle between the lines passing the centers of the system and the cart to estimate θ.

Figure 1 illustrates the actual value of θ and the estimated θ using the visual tracking mechanism above for around 500 time steps and figure 2 shows how the visual tracking process works for 9 consecutive frames.
The experiment used to produce figures 1 and 2 was done in a manual mode. For simplicity, a stride of 10 steps is used to show the consecutive frames. (These frames were created by the helper function *drawCenterOfMasses* that draws a $\star$ at the center points of both the cart and the pole in addition to the green lines that connect the centers for better readability of the graphs)

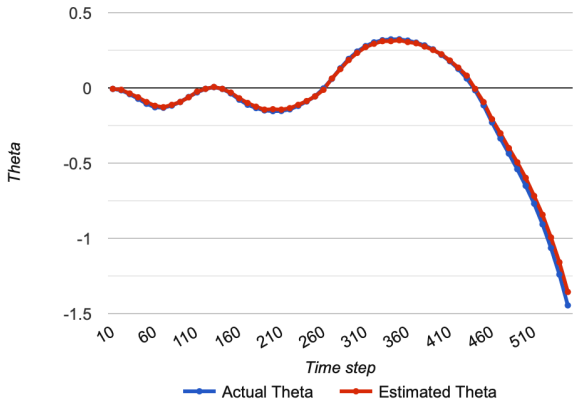
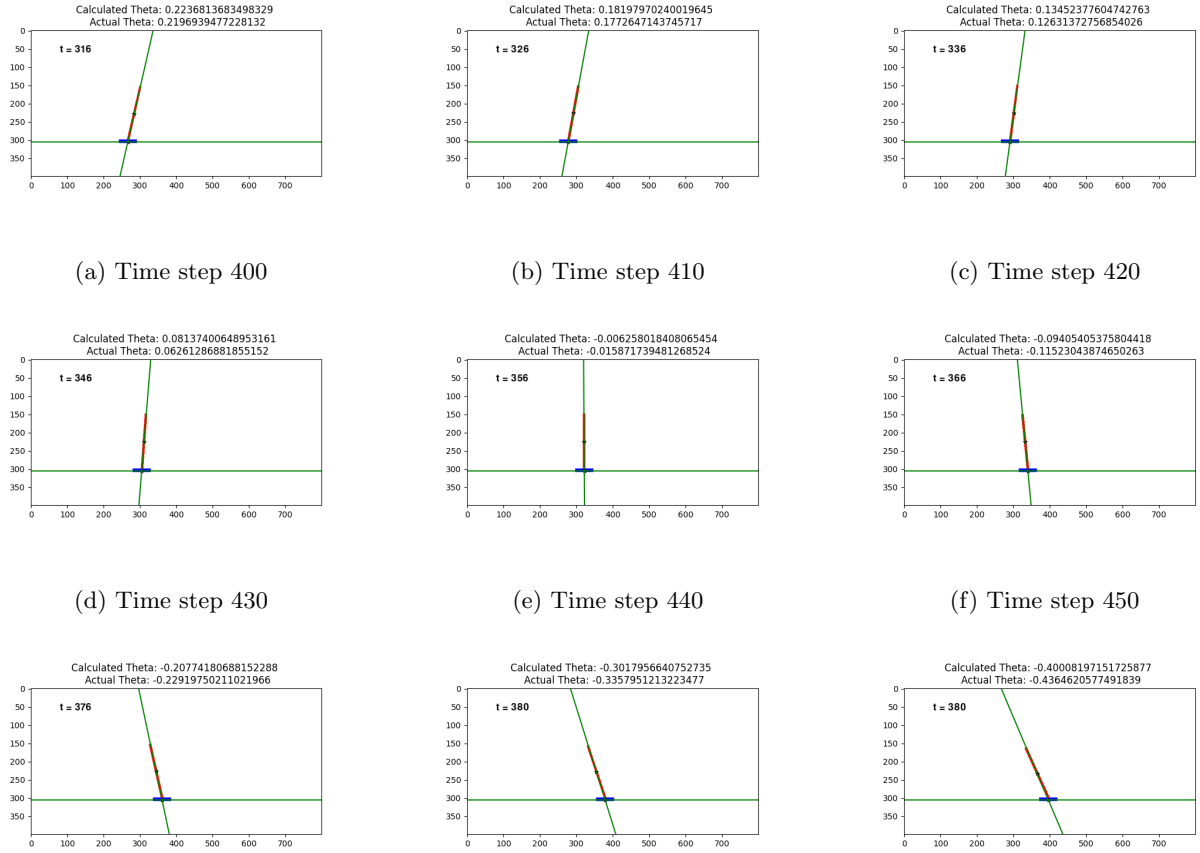


Figure 1: Comparing the true and estimated values of theta for 550 time steps



(a) Time step 400 (b) Time step 410 (c) Time step 420

(d) Time step 430 (e) Time step 440 (f) Time step 450

(g) Time step 460 (h) Time step 470 (i) Time step 480

Figure 2: Visual tracking procedure for 9 consecutive frames

2 Kalman Filter

This part is implemented in the newly added class named *KalmanFilter* in *KalmanFilter.py*. This class has all the Kalman Filter's matrices as its parameters and only has one function namely *predict_update* that performs the predict/update phases. The operations and updates done in *predict_update* are as below:

- State estimate is updated from system dynamics:

```
x_pri = np.matmul(self.F, self.x_last) + self.B * u
```

- Uncertainty estimate grows:

```
P_pri = np.matmul(np.matmul(self.F, self.P_last), np.transpose(self.F)) +
    self.Q
```

- The difference between expected and actual value of sensor reading is computed:

```
r_hat = z - np.matmul(self.H, x_pri)
```

- The covariance of sensor reading is computed:

```
S = np.matmul(np.matmul(self.H, P_pri), np.transpose(self.H)) + self.R
```

- The Kalman gain is computed:

```
K = np.matmul(np.matmul(P_pri, np.transpose(self.H)), inv(S))
```

- The state estimate is corrected by multiplying residual times gain:

```
x_post = x_pri + np.matmul(K, r_hat)
```

- Uncertainty estimate shrinks:

```
P_post = np.matmul((np.eye(2) - np.matmul(K, self.H)), P_pri)
```

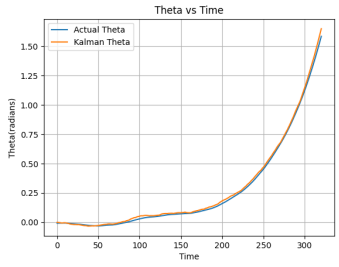
- Finally the values of *x_post* and *P_post* is stored into *x_last* and *P_last* for the next round.

The matrices Q, R, F, H, B are initialized in the function *game* and passed to *game_round* for each round of the game. After tuning these parameters programmatically and comparing the predictions with the actual values of θ and $\dot{\theta}$ (from the actual system's model provided from the previous assignments), the following values were selected:

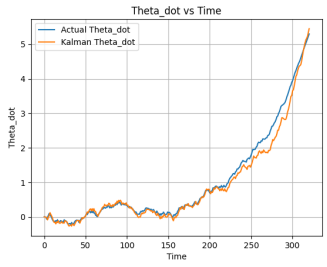
Parameter	Value
dt	0.017
F	$\begin{bmatrix} 1 & dt \\ 0.1 & 1 \end{bmatrix}$
H	$\begin{bmatrix} 1 & 0 \end{bmatrix}$
B	$\begin{bmatrix} 0 & -0.08 \end{bmatrix}$
Q	$\begin{bmatrix} 0.001 & 0 \\ 0 & 0.001 \end{bmatrix}$
R	$\begin{bmatrix} 0.01 \end{bmatrix}$

Table 1: Tuning results for the Kalman parameters

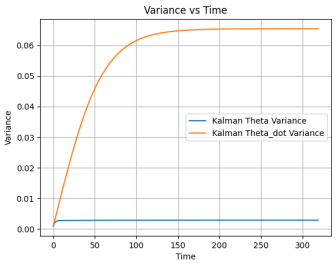
Figure 3 and 4 illustrates a summary of the Kalman Filter implementation's performance using a random controller and the manual mode.



(a) Estimating θ for a random controller

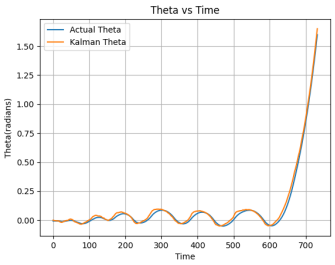


(b) Estimating $\dot{\theta}$ for a random controller

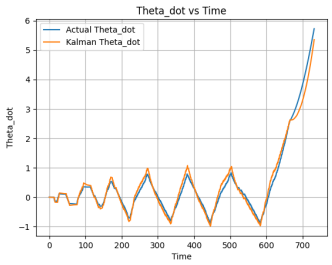


(c) Estimation of error variance per time for a random controller

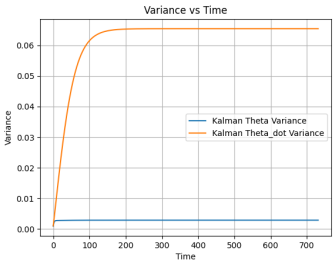
Figure 3: Kalman Filter results for a random controller



(a) Estimating θ for manual control



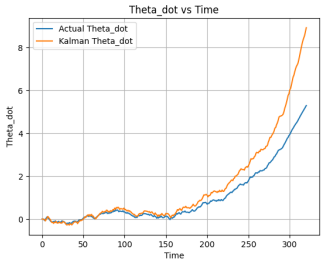
(b) Estimating $\dot{\theta}$ for manual control



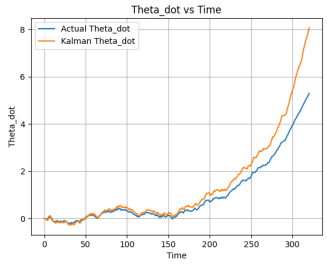
(c) Estimation of error variance per time for manual control

Figure 4: Kalman Filter results for manual control

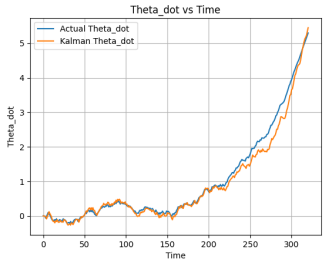
An interesting observation while tuning the Kalman parameters was the role of dt in estimating $\dot{\theta}$. Figure 5 shows $\dot{\theta}$ per time plots for different values of dt fixing the other Kalman parameters as in table 1. (Note that the plot shown in 5c use the same value $dt = 0.017$ as the final Kalman implementation in this project)



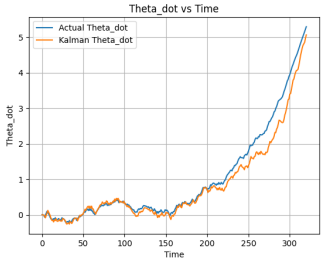
(a) dt = 0.001



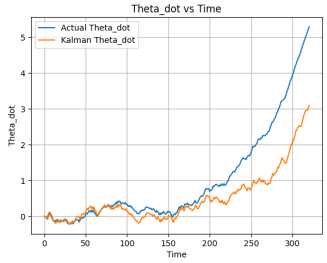
(b) dt = 0.005



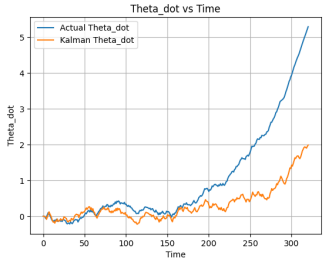
(c) dt = 0.017



(d) dt = 0.02



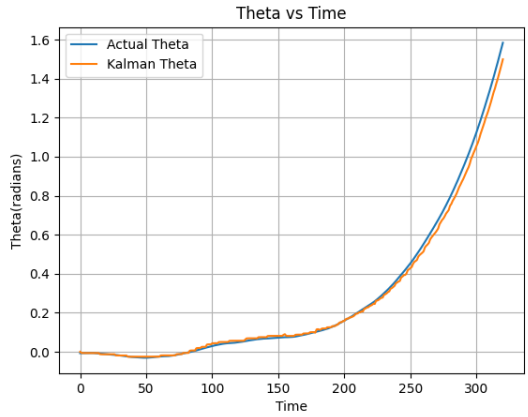
(e) dt = 0.05



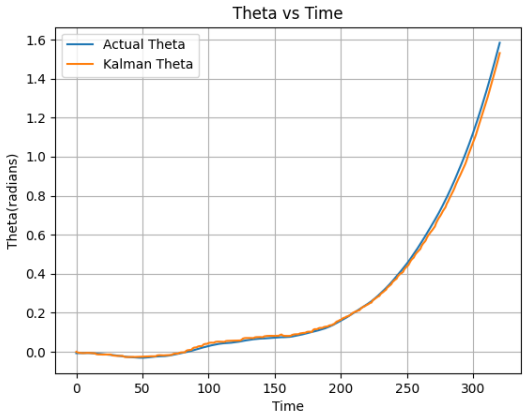
(f) dt = 0.1

Figure 5: Effect of dt on the performance of Kalman estimation for a random controller

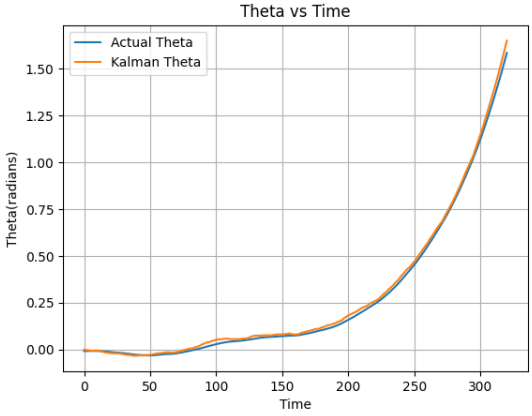
Finally, as reported in table 1, the value selected for R is greater than Q. This is because the measurements coming from the visual tracking part are noisy and therefore we prefer the model over the accuracy of the measurements. To better illustrate the role of R on the measurement noise, figure 6 shows θ per time plots for different values of R fixing the other Kalman parameters as in table 1. (Note that the plot shown in 6b uses the same value $R = 0.01$ as the final Kalman implementation in this project)



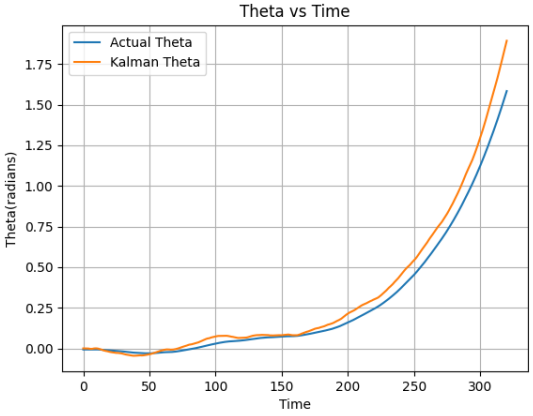
(a) R = 0.0001



(b) R = 0.001



(c) R = 0.01



(d) R = 0.1

Figure 6: Effect of R on the performance of Kalman estimation for a random controller

3 Designing the Controller

The bulk of the code used in this section is taken from the submission made for assignment two. However, the parameters kp, ki, kd are tuned again specifically for our new model. The approach for tuning these parameters was to programmatically change them within some bounds and select the combination with the best performance. Figure 7 shows the performance of the controller for $kp = 11$, $kd = 2$, $ki = 0.1$.

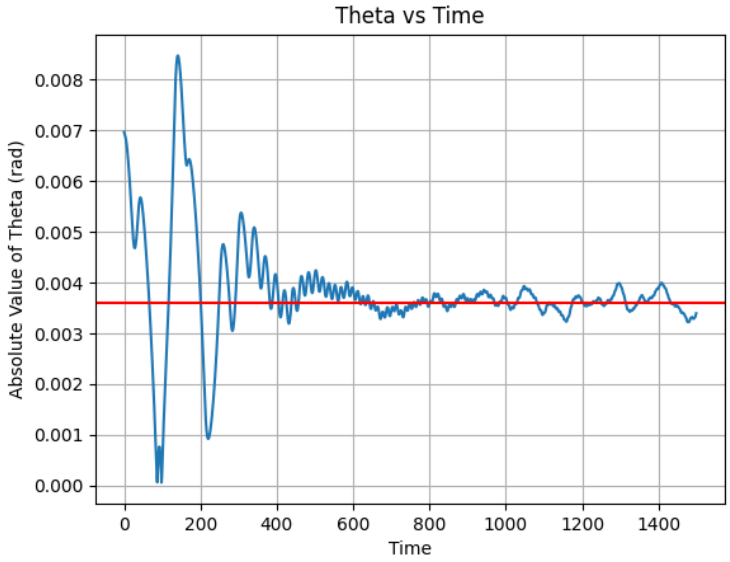


Figure 7: PID controller performance

There are three key observations for this part of the project. Comparing the performance of the PID controller:

- The first observation is that the balance point is somewhere approximately around 0.037 radians shown by the red horizontal line. In the previous PID controller that was designed, however, the pole was getting stable around 0 radians which is the optimal state. To explain this behavior note that figure 7, shows the actual values of θ taken from the system's model. (These actual values are only used for analysis and visualization purposes and they are not a part of the control mechanism) However, the input to the PID controller is the Kalman estimations for θ and $\dot{\theta}$. Hence, from the perspective of the controller, the error to minimize is $kalman\_\theta$. Therefore, even though the controller s attempting to keep the $kalman\_\theta$ as close as possible to zero, the actual θ of the system is a bit higher due to not having the optimal accuracy in the visual tracking and Kalman estimation parts. (Comparing the estimation values of θ with the actual θ confirms this behavior as the Kalman estimation was on average lower than the actual θ values)

- Next observation is that near the balance point, there are some small oscillations (smaller than 0.001 radians). Even with extensive tuning for the Kalman and PID parameters, these small oscillations didn't stop within the time window of 1500 steps. As mentioned earlier this is because of the reduced performance of the controller as it is working with the Kalman estimations as inputs instead of real values of θ.

- Finally, the previous two observations lead to the conclusion that to get the optimal performance with our controller in practice, much more extensive tuning is needed not only for the controller part but also for the first two phases of visual tracking and Kalman filter implementation.

Timeout	Simulated Annealing	Parallel Tempering	Quantum Monte Carlo	Tabu Search	
Simulated Annealing	0	0	0	0	0
Simulated Annealing	0	0	0	0	0
Parallel Tempering					
Quantum Monte Carlo					
Tabu Search					