Name: Amani Jammoul
McGill ID: 260381641
COMP 417 - Introduction to Robotics & Intelligent Systems

Final Project Report

Visual Tracking

In order to implement visual tracking, I created a class called VisualTracker() that determines the pole's angle from observing an image. It generates an image from the InvertedPendulumGame surface_array (as if this came from a camera). To localize the pendulum, it extracts the red channel from the image since that is the color of the pendulum (as seen in Figures 1 and 2). After extracting the red channel and cropping the image, the visual tracker uses cv2 to perform thresholding and to find the outline of the object. It uses cv2.threshold and cv2.findContours to find the outline of the pole (Figure 2). Lastly it uses cv2.fitLine to get a line through the pendulum. An example of how the line is drawn is shown in Figure 4. Once the angle of the line versus the vertical axis is calculated, that becomes the value of the observed angle.

Fig. 1: Image before color extraction

Fig. 2: Image after red color extraction

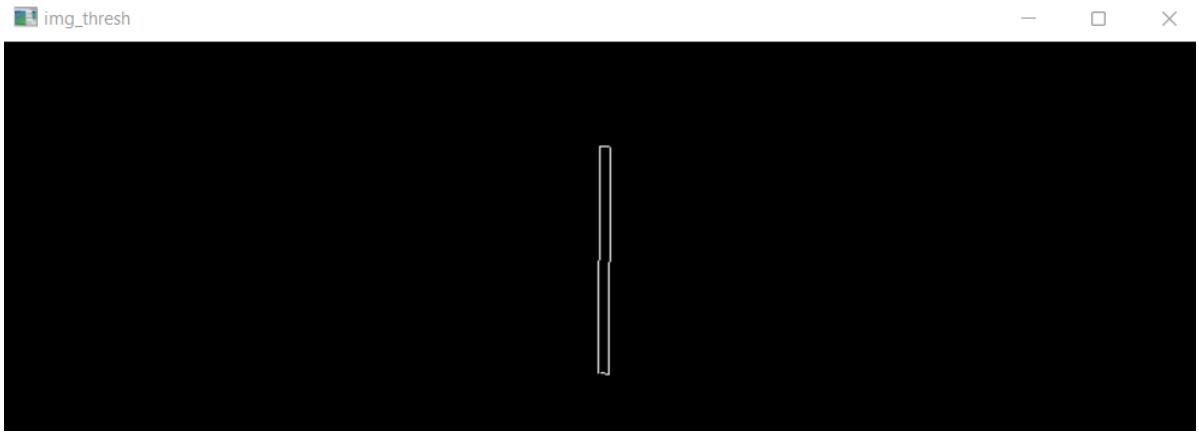

Fig. 3: Image after thresholding

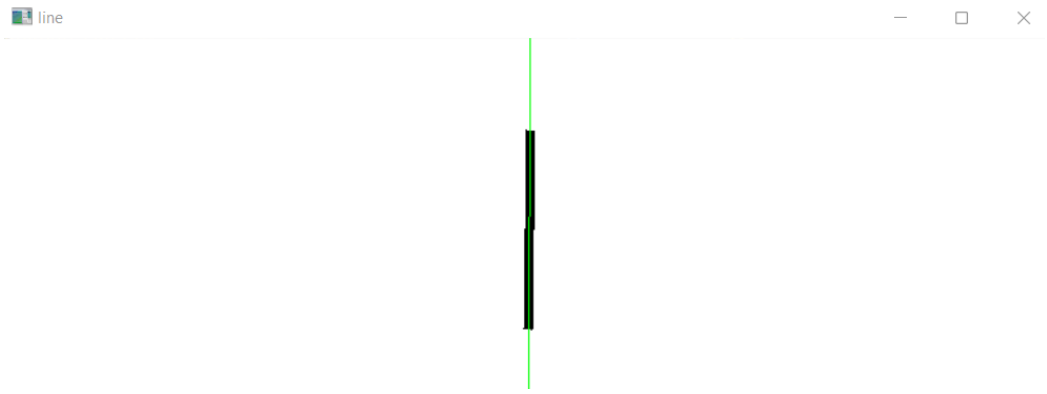

Fig. 4: Line used to calculate angle

The following are four other consecutive steps (with 20 timesteps apart).

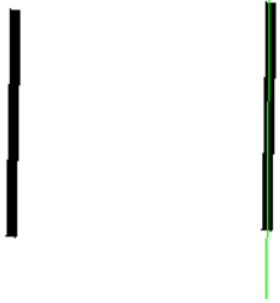

Fig. 5: Img at t=20

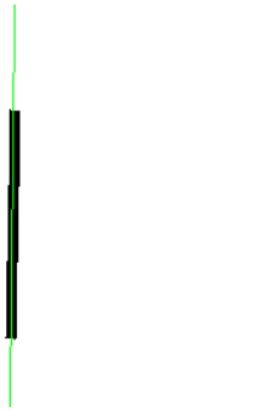

Fig. 6: Line at t=20

Fig. 7: Img at t=40

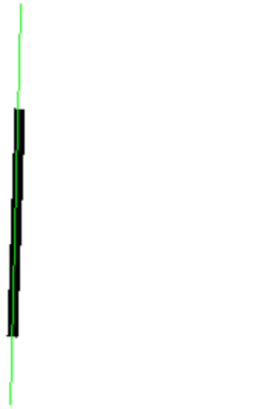

Fig. 8: Line at t=40

Fig. 9: Img at t=60

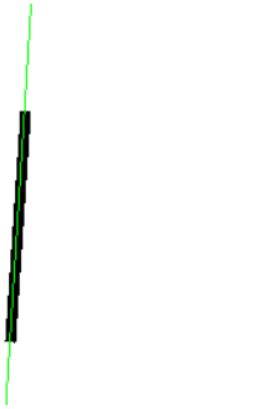

Fig. 10: Line at t=60

Fig. 11: Img at t=80

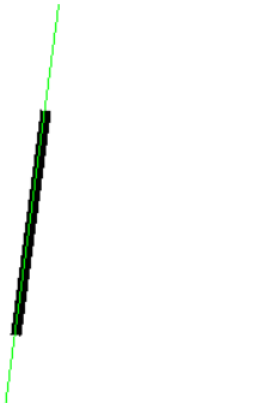

Fig. 12: Line at t=80

I tried different methods for observing the angle, such as using the contours to draw a rectangle around the pendulum. I also tried finding the center of the pendulum and the center of the cart, and using those to calculate the angle. However, compared to the true angles (which come from the pendulum's true state), none were more accurate than my final method. Figure 13 shows a plot of values of the difference between the observed angles and the true values versus time.

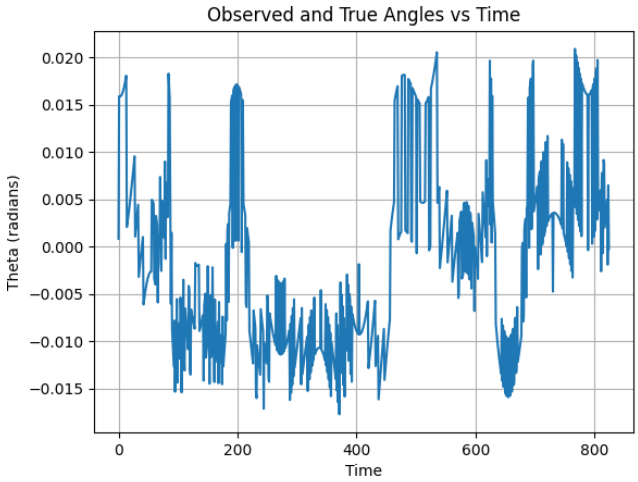

Fig. 13: Observed and True Angles vs Time

Kalman Filter

For this part I created a KalmanFilter class. Upon creation, F, B, H, Q, R, and P_0 are initialized to these values:

State Transition: F = [[1, dt]
[0, 1]]

Control Input: B = [[0]
[dt]]

Measurement Mapping: H = [[1, 0]]

Process Noise Covariance: Q = [[0.01**3, 0]
[0, 0.01**3]]

Measurement Noise Covariance: R = [[0.01]]

Initial State Covariance: P_0 = [[0.01, 0]
[0, 0.01]]

The initial state is set to [0, 0]. The measurement noise covariance R is greater than the process noise covariance Q (R > Q) because I assume more noise from the readings than from the model.

There is one function that performs the propagation and update steps.This function is passed u and z as parameters. The value of u is the control input, i.e. theta acceleration. The value of z is the observation, i.e. the observed angle from the visual tracker.

After the propagation step, the state estimate is:

$x\_(t|t\text{-}1) = F*x\_(t\text{-}1|t\text{-}1) + B*u = [[\ \theta \quad = [[\theta_{t-1} + \theta dot_{t-1} * dt$

$[\theta dot]] \quad [\ \theta dot_{t-1} + dt * \theta acc \quad]]$

The plots of the estimated state values along with the true state values are shown in Figures 14 and 16. From the state covariance matrix P, I plotted the estimation error variance of both theta and theta_dot. These are shown in Figures 15 and 17.

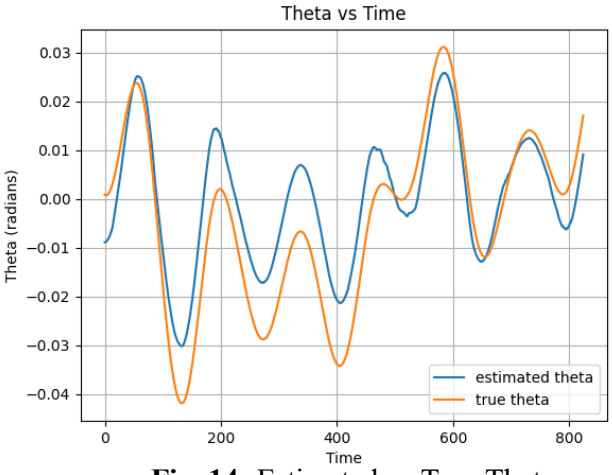

Fig. 14: Estimated vs True Theta

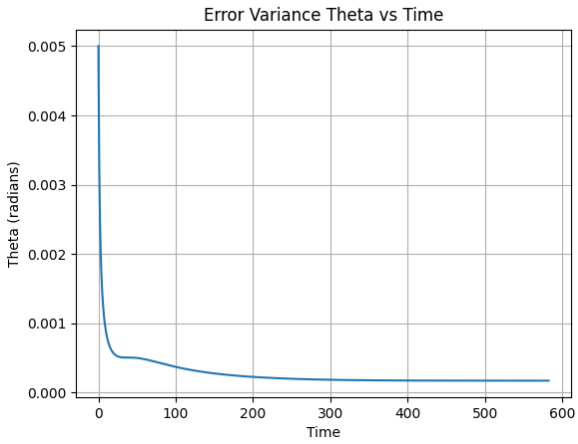

Fig. 15: Theta Error Variance

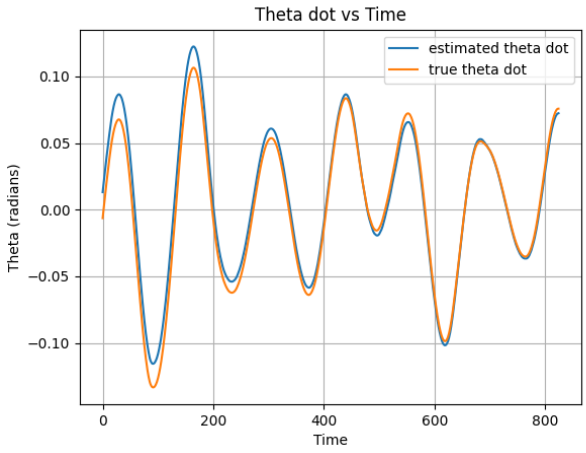

Fig. 16: Estimated vs True Theta Dot

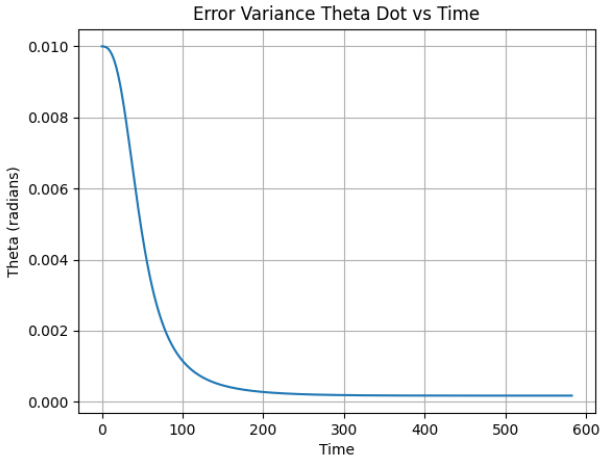

Fig. 17: Theta Dot Error Variance

PID Controller

The PID controller was very difficult to tune, due to the fact that we were using state estimations. At the start the estimations are not strong. As time goes on, the Kalman Filter estimations become better. However at the start, the pendulum would oscillate about the wrong value. With the addition of noise, it was difficult to correctly implement this controller. I tried to tune it in the same way as in assignment 2. I began by setting the gains to 0, then slowly incremented Kp until the pendulum oscillated. Afterwards I slowly incremented Kd until the pendulum seemed somewhat steady. Last I added Ki.

An interesting remark is that, when the estimated value were off, the pendulum would oscillate about an incorrect value. As time went on and the values improved, the pendulum started oscillating about theta=0. Figures 18 and 19 show an example.

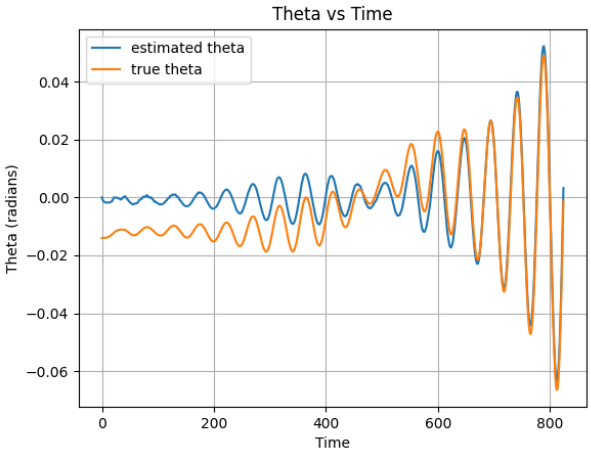

Fig. 18: Estimate vs True Theta

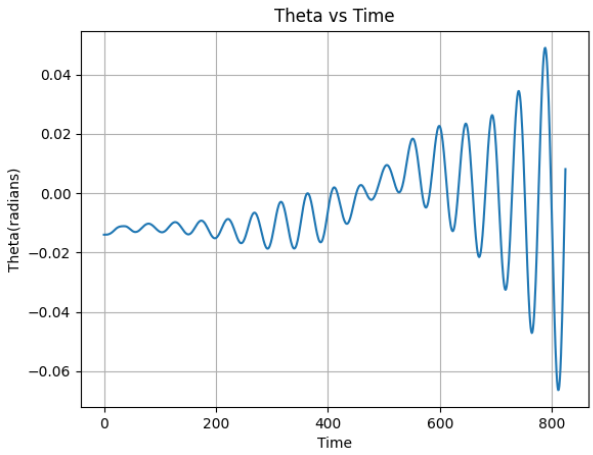

Fig. 19: Performance