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

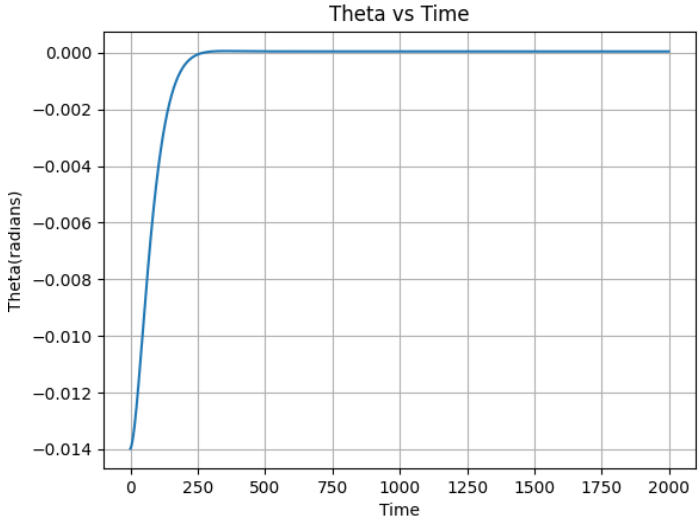
Assignment 2 PID Controller

A. When designing the PID controller, I began by defining the error, derivative, and integral terms which would be scaled by the gains. The pendulum's target angle (set point) is 0, meaning the error is equal to the angle. At first, I set a self.prev_error value in the PID_controller_object class in order to derive the derivative at each step (getting the error difference and dividing it by the change in time, dt=0.005s). However, I later noticed that theta_dot was provided in the state that is passed to the controller, so I used that as the derivative value instead. Lastly, I created a self.prev_integral attribute in the controller class to calculate the integral at each step, since this is a summation of all the previous errors multiplied by dt.

Once I defined those values, I started tuning the gains. I began by setting the derivative and integral gains to zero (Kd and Ki) in order to find a good enough Kp value. I tried to get a value with a quick enough response time that doesn't cause too much oscillation. This value was around Kp = 1.35. Afterwards, I added the derivative gain to control the overshoot and eliminate oscillation. For this, the ideal value was Kd = 0.45 when Kp was raised to 1.9. Lastly, I added the integral gain to correct time invariant errors and the ideal value for that I found to be Ki = 0.008. Therefore, the final gain values for my PID-controller are:

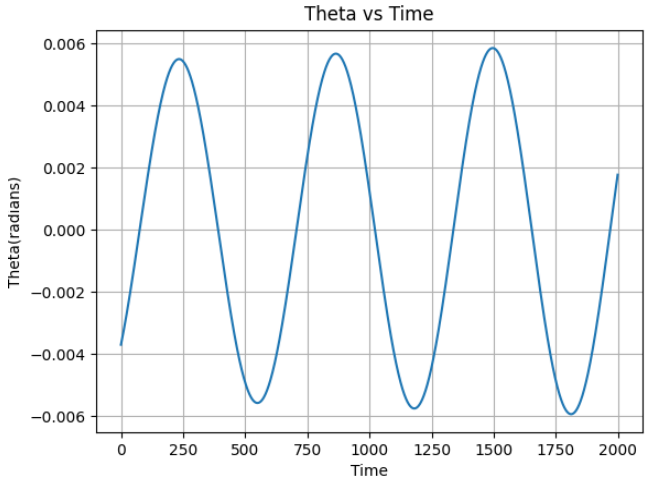
- Kp = 1.9
- Kd = 0.45
- Ki = 0.008

Below is a theta vs time plot for a 10 second run (2000 timesteps) including noise. As we can see, the system responds and adjusts relatively quickly, does not oscillate about the target, and has a very low steady state error.

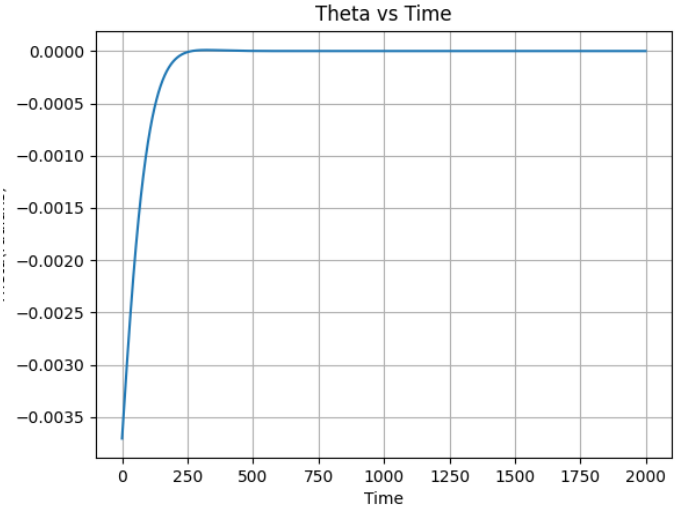


B. The following is a description of how I tuned the different controllers, what results I found, and what challenges I faced. Below each controller description is an image of a theta vs time plot that represents the pendulum's angle over a 10 sec run.

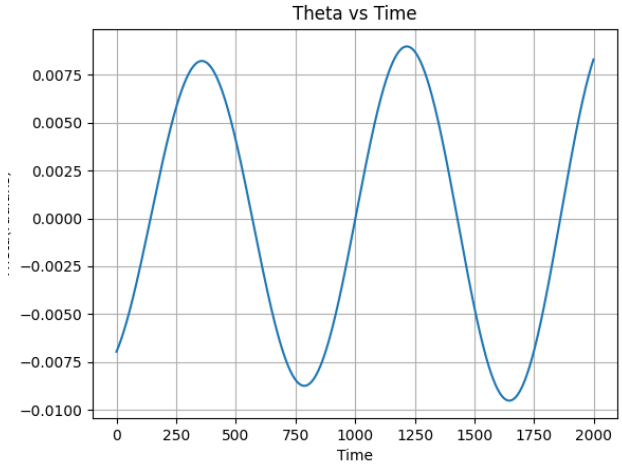
a. **P-controller:** This was the first type of controller I tried to implement. As I was trying to tune Kp, I found that smaller values were better, but anything below 1.1 did not work well. Also, a negative gain caused the pendulum to fall right away. A proportional gain that was too low took too long to respond to errors, while those that were too large corrected too much and caused large oscillations. I found a value of Kp = 1.45 to be as ideal as possible. This value proved quite difficult to find, as it required a lot of tuning. As the response time improved, the range of oscillation increased, so I found myself constantly trying to find a good balance.



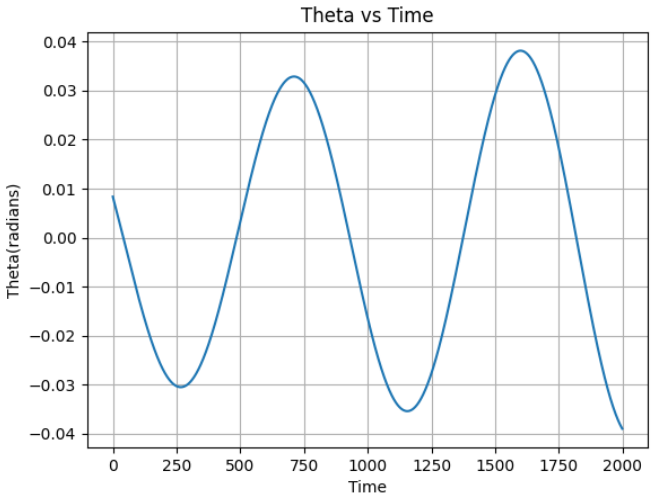
b. **PD-controller:** This controller was the easiest to tune, as I found a good Kd value relatively quickly using the same Kp as in the P-controller as a starting point. When I set Kd = 0.45, the system seemed to respond well, with no oscillation and little overshoot. A negative Kd value caused immediate failure, while a value between 0.1 and 0.4 still allowed for a high overshoot and a bit of oscillation at the start before steadying. I also found that slightly increasing the proportional gain resulted in an even better plot, therefore for this controller, Kp = 2.



c. **PI-controller:** For this controller I started by setting the proportional gain to the value from the P-controller, which was Kp = 1.35, then I tried to tune Ki. This proved to be the most difficult controller to set since it is not as easy to detect the contribution of the integral gain. The integral portion controls the system's response to changes over time, which is especially beneficial when there is noise. However, it can be difficult to find a proper gain as its effects are not as obvious. I did find that a large Ki value caused instability, and a low Ki value did not allow the system to respond to changes well enough. In the end, I found Ki = 0.005 was the ideal gain.

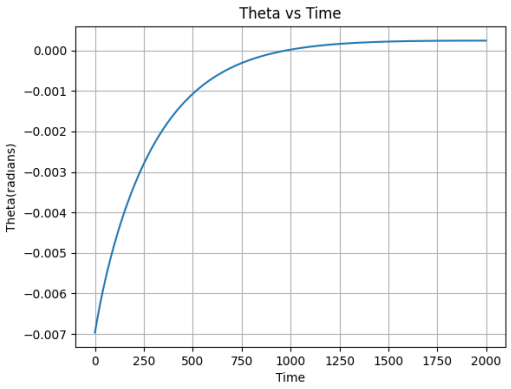


d. **DI-controller:** This type of controller does not work well to steady the system. After some tuning I found values Kd = 20 and Ki = 40 to be the best. However, even with this controller the system is unstable. This is due to the lack of proportional error correction. The derivative correction only takes into account the error differentiation, without actually looking at the set point. The integral looks at the accumulation of the error over time. But even with these gains, without any immediate correction for the instantaneous error, the system can never properly fix itself in all cases.

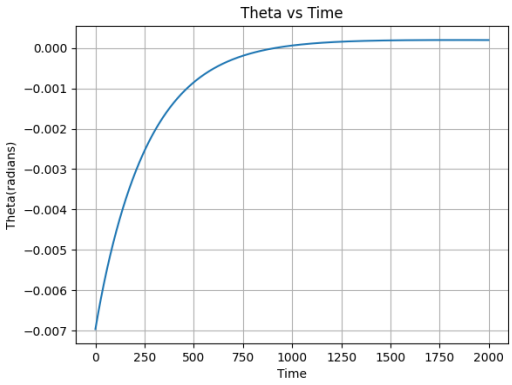


C. Adjusting the system and the environment's values affects the controller's efficiency. For instance, increasing either of these values by some small amount increases the response and setting times, while also increasing the steady state error. Below are plots when the pendulum's mass is 0.6 and gravity is 13, respectively. In both cases, as the values get too high, the system's response to errors gets worse and eventually reacts extremely poorly.

mass of pendulum = 0.6

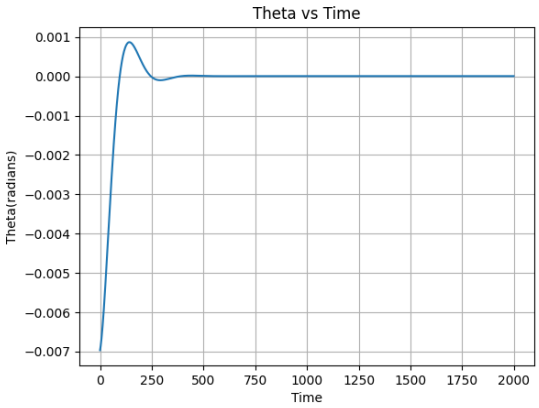


acceleration due to gravity = 13

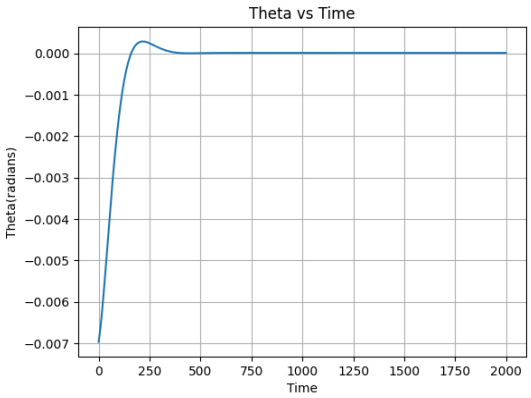


In addition, when acceleration due to gravity decreases, the system overshoots more and takes more time to settle. Below are plots when g = 6 and g = 0.

acceleration due to gravity = 6



acceleration due to gravity = 0

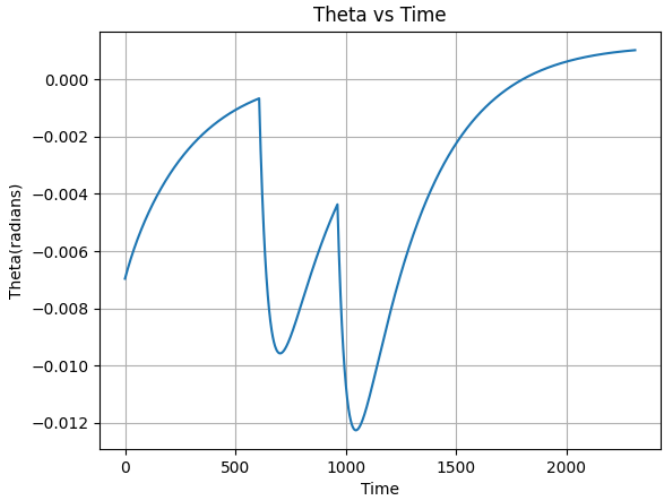


D. For this part, I added random disturbances as suggested in the instructions (with numpy.random.rand()). In essence, this returns a high action value with a certain probability. What I found was, if the probability of having a disturbance was low enough, meaning they were not introduced too often, the system was able to correct itself better. This is also true if the disturbing action value was low enough (causing less disturbance).

However, when the probability and/or disturbing action value was too high, the system failed to keep the pendulum upright.

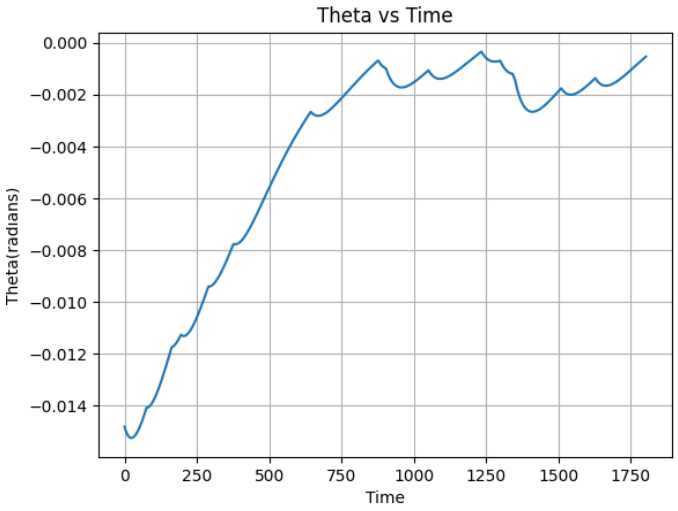
Below is a plot when the probability of disturbance was 0.001 and disturbing action = 10. As we can see, the system keeps trying to move towards the goal (angle = 0) before being pushed in the wrong direction (extremely). However, since these disturbances don't happen as often, the system has more time to correct itself.

high disturbance (action=10) with low probability (p=0.001)



Below shows a plot when disturbances occur more often (probability is higher), but the disturbance itself is not as high. As we can see, the system is pushed in the wrong direction more often, however it still, slowly, moves towards the goal.

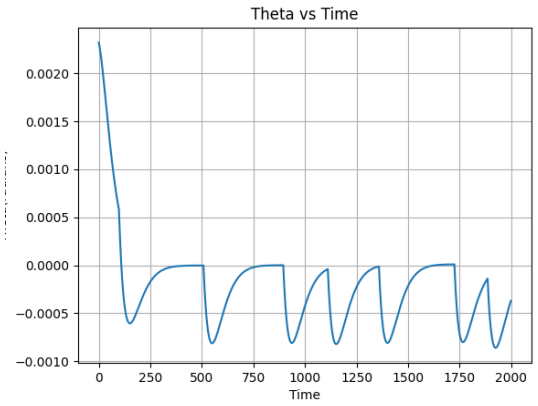
low disturbance (action=0.1) with high probability (p=0.01)



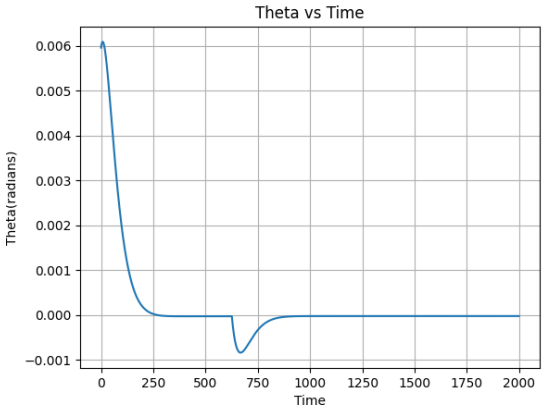
Of course, when the disturbance is low and does not occur often, the system is still able to react well by having time to go back to the goal each time it is skewed. Below shows two plots when the disturbance action is 0.1 and probability of occurrence are low (0.005 and 0.001). Each time it is skewed, it does not deviate too much from the goal and is able to work its way towards 0 at a quick enough rate before the next disturbance.

low disturbance (action=0.1)

p=0.005

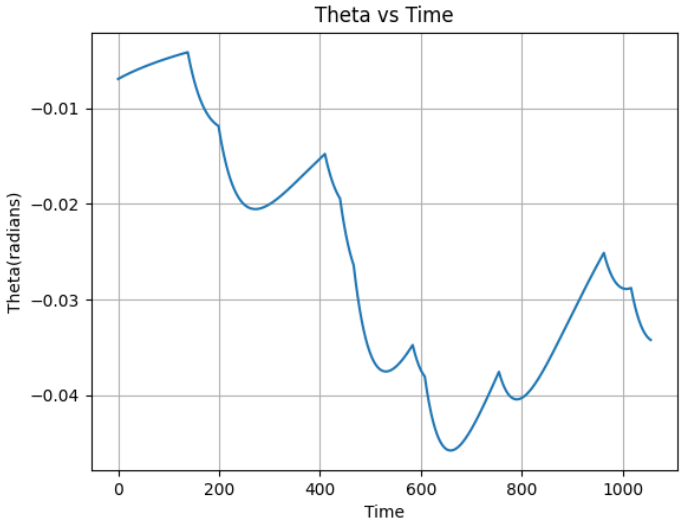


p=0.001



Below is a plot when both the disturbance and its probability of occurrence is high. This causes much more failure as the system keeps being pushed in the wrong direction and furthermore does not have enough time to correct itself. As we can see, the angle of the pendulum steadily decreases, in the opposite direction of the goal.

high disturbance (action=10) with high probability (p=0.01)



One remark I made was, if the PID controller gains were adjusted, specifically Ki, the system would react better to these disturbances. If Ki was higher, the system would move more quickly towards the goal each time there is a disturbance, because the action would be more highly proportional to the integral of the error. It is in such cases where the value of the integral gain comes into play. When everything is more "ideal", with no noise or disturbance, this gain is not as valuable. This is why, at the beginning, I struggled with defining Ki. I was unable to see its utility. With no disturbances, the PD-controller was working very well and I did not see much improvement in adding the integral portion.

For experimental purposes, I tried changing Ki to see what happens. In the trials presented above, Ki was very low (0.008). When I increased it to Ki=1, with the same high disturbance and high probability as the trial above (action=10 with p=0.01), the system performed much better. It was able to last longer, correcting itself more at a faster rate each time there was a disturbance.

high disturbance (action=10) with high probability (p=0.01), higher integral gain (Ki=1)