ROB498 Final Project: Double Pendulum Trajectory Optimization

Joseph Kennedy, Kamil Nocon
{josephpk, nocon}@umich.edu
April 21, 2023

***Abstract*—This paper explores different trajectory optimization techniques for the double pendulum upswing task. Differential Dynamic Programming (DDP) and Linear Quadratic Regulator (iLQR) are implemented and their performance is compared to Model Predictive Path Integral (MPPI).**

I. Introduction

In the field of robotics, there is often a complex physical system that must be controlled. These systems can vary from a seven degree of freedom robot arm, to a quadcopter, to a humanoid robot. These systems are almost always nonlinear, so linear methods such as full state feedback will not work. An example of a highly nonlinear system is the inverted double pendulum, as shown in Figure 1. This system consists of a cart that can move linearly along a track, a pendulum link that is attached to the cart, and a second pendulum link that is connected to the first link. A linear control force can be applied to the cart, but the joints of the two pendulums cannot be controlled and spin freely. The goal of the controller is to transition the system from a starting state (where the pendulum arms are hanging straight down) to a goal state (where the pendulum arms stand straight up). The state space of this model has six variables: the position of the cart, the angle of the first joint, the angle of the second joint, the linear velocity of the cart, the angular velocity of the first joint, and the angular velocity of the second joint. Thus, this is a difficult system to control, as one control input must drive six states to a desired value. A state is represented as:

Model predictive control (MPC) is one framework used to design controllers for these highly nonlinear systems. It seeks to find a control action that minimizes a cost function over a control sequence for a finite period of time, or horizon. Given the starting state of the system, an MPC controller will calculate a sequence of control actions over the horizon time, or a trajectory. The cost function is often determined by the current state, the current action, and the goal. After computing this trajectory, the MPC controller will apply just the first action to the system, then it will observe the state of the system. Because the model in which MPC calculated the trajectory of the system may be an approximation, or because of noise in the system, the state the system arrives in may not be the same state that the MPC controller predicted. Thus, the controller will take the state of the system and run another iteration to compute the next trajectory. In this manner, the controller will seek to drive the system to the goal state.

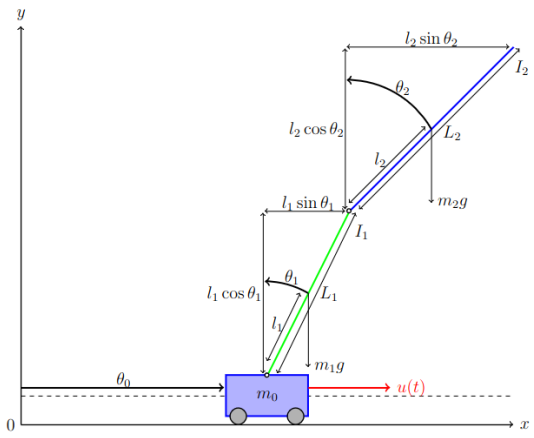

Fig. 1. The inverted double pendulum, as described by [1]

There are several variants of MPC controllers [2], such as iLQR, DDP, and MPPI. Each of these methods requires a motion model, or the equations of motion to get the next state given the current state and action, and the cost function, or the equation to calculate some penalty given the current state and action. DDP, or Differential Dynamic Programming, is a second-order shooting method that achieves quadratic convergence under certain assumptions. DDP requires both the first and second-order derivatives of the motion model and cost function. The second-order derivatives of the cost function are often the most expensive to compute [3]. iLQR, or Iterative Linear Quadratic Regulator, is very similar to DDP, but only the first-order derivatives of the motion model are used. This reduces the computation time of the algorithm [4].

MPPI [5], or Model Predictive Path Integral control, is another MPC method that is similar to DDP and iLQR. However, this model is a sampling-based optimizer and does not require any derivatives. Given an initial trajectory (either initialized to be all zeros, random actions, or some guess of the correct trajectory), this trajectory is copied K times, and small perturbations are applied to each action in each of these trajectories. Then the dynamics are rolled out for the K trajectories, and the resulting cost of each trajectory is calculated. Finally, the K trajectories are weighted according to the inverse of their cost, and the weighted average is calculated. This resulting trajectory is returned, and the controller will execute the first action. However, due to an uncertain model or noise in the system, the resulting state after executing the action may not be where the controller anticipated, so the controller will calculate a new trajectory given this new state.

II. Implementation

A. Equations of Motion

For any MPC controller, the equations of motion of the system must be known. The accuracy of these equations will greatly determine the success of the MPC controller [2]. The equations of motion for the inverted double pendulum can be calculated using Lagrange equations [1]. It is assumed that the center of mass for each pendulum link is at its center and that the moments of inertia for each pendulum link with respect to its center of mass is

$$I = \frac{1}{12}ml \tag{1}$$

where m is the mass of the pendulum link and l is its length.

The state $\mathbf{x}$ can be represented as:

$$\mathbf{x} = \begin{bmatrix} x \\ \theta_1 \\ \theta_2 \\ \dot{x} \\ \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix} \tag{2}$$

where x is the position of the cart, θ_1 is the angle of the first joint, θ_2 is the angle of the second joint, $\dot{x}$ is the velocity of the cart, $\dot{\theta_1}$ is the angular velocity of the first joint, $\dot{\theta_2}$ is the angular velocity of the second joint. The input u is the force applied to the cart. m_0 is the mass of the cart, m_1 is the mass of link 1, m_2 is the mass of link 2, l_1 is the length of link 1, l_2 is the length of link 2 and g is the acceleration due to gravity. The system can be converted to a first-order system described by:

$$\dot{x} = A(x)x + B(x)u + L(x) \tag{3}$$

where $A(x)$, $B(x)$, and $L(x)$ are defined as

$$A(x) = \begin{bmatrix} 0 & 0 \\ 0 & -D^{-1}(x)C(x) \end{bmatrix} \tag{4}$$

$$B(x) = \begin{bmatrix} 0 \\ D^{-1}(x)H \end{bmatrix} \tag{5}$$

$$L(x) = \begin{bmatrix} 0 \\ -D^{-1}(x)G(x) \end{bmatrix} \tag{6}$$

and $D(x)$, $C(x)$, $G(x)$ and H are defined as

$$D(x) = \begin{bmatrix} D1 & D2 & D3 \end{bmatrix} \tag{7}$$

$$D1 = \begin{bmatrix} m_0 + m_1 + m_2 \\ (0.5m_1 + m_2)l_1 cos(\theta_1) \\ 0.5m_2 l_2 cos(\theta_2) \end{bmatrix} \tag{8}$$

$$D2 = \begin{bmatrix} (0.5m_1 + m_2)l_1 cos(\theta_1) \\ (\frac{1}{3}m_1 + m_2)l_1^2 \\ 0.5m_2 l_1 l_2 cos(\theta_1 - \theta_2) \end{bmatrix} \tag{9}$$

$$D3 = \begin{bmatrix} 0.5m_2 l_2 cos(\theta_2) \\ 0.5m_2 l_1 l_2 cos(\theta_1 - \theta_2) \\ \frac{1}{3}m_2 L_2^2 \end{bmatrix} \tag{10}$$

$$C(x) = \begin{bmatrix} C1 & C2 \end{bmatrix} \tag{11}$$

$$C1 = \begin{bmatrix} 0 & -(0.5m_1 + m_2)l_1 sin(\theta_1)\dot{\theta_1} \\ 0 & 0 \\ 0 & -0.5m_2 l_1 l_2 sin(\theta_1 - \theta_2)\dot{\theta_1} \end{bmatrix} \tag{12}$$

$$C2 = \begin{bmatrix} -0.5m_2 l_2 sin(\theta_2)\dot{\theta_2} \\ 0.5m_2 l_1 l_2 sin(\theta_1 - \theta_2)\dot{\theta_2} \\ 0 \end{bmatrix} \tag{13}$$

$$G(x) = \begin{bmatrix} 0 \\ -0.5(m_1 + m_2)l_1 g sin(\theta_1) \\ -0.5m_2 l_2 g sin(\theta_2) \end{bmatrix} \tag{14}$$

$$H = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{15}$$

B. Cost Function

The goal of the trajectory optimization is to minimize the total trajectory cost J, which is the sum of the running cost ℓ for each state, action pair, and the final cost ℓ_f of the final state. The cost is taken

for all actions $U = \{u_0, u_1, ..., u_{N-1}\}$ and all states $X = \{x_0, x_1, ..., u_N\}$, where N is the trajectory length:

$$J(X, U) = \sum_{n=0}^{N-1} \ell(x_i, u_n) + \ell_f(x_N) \tag{16}$$

For optimal performance of the trajectory optimization algorithms used, a quadratic cost function was used for both the running cost and the final cost with respect to a reference state x_r:

$$\ell(x_n, u_n) = (x - x_r)^T S(x - x_r) + u^T R u \tag{17}$$

$$\ell_f(x_N) = (x - x_r)^T S_f(x - x_r) \tag{18}$$

The matrices S and R are diagonal matrices with each term chosen to weigh each variable in the state space and action space. To balance the double pendulum in the upright position, the weights were chosen to heavily penalize the error in θ_1 and θ_2 to ensure the links stay to maintain the vertical position. The error in the x position was weighted somewhat heavily to ensure that the cart did not deviate too much from the center. The velocities were weighted less to allow for quick changes to the system and ensure that it was responsive. The weight of the control input was weighted the least to allow for large control inputs to stabilize the system. S_f was chosen to be a scaled version of S to have high importance of the final state matching the goal state. The tuned values for the upswing task were:

$$S = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix} \tag{19}$$

$$R = \begin{bmatrix} 0.001 \end{bmatrix} \tag{20}$$

$$S_f = 1000 Q_f \tag{21}$$

C. iLQR and DDP

Two similar approaches to trajectory optimization are iterative Linear Quadratic Regulator (iLQR) and Differential Dynamic Programming (DDP). These approaches can be used for non-linear systems such as the double pendulum problem via quadratic approximation of the cost function. The dynamics are approximated as linear (iLQR) or quadratic (DDP). The algorithms then use the approximation to iteratively update the actions of an initial action sequence U to achieve a minimum total cost. For each step in the MPC, the initial action sequence used is the previously calculated action sequence from iLQR or DDP after the first action has been executed. For the first step of the MPC a random sequence of actions is used initially. For each iteration of the DDP or iLQR algorithm, the derivatives of the cost function and dynamics functions were calculated, a backward pass was completed, and finally, a forward pass was completed. This was for a trajectory until the total trajectory cost converged.

1) Derivative Calculations: To perform the linear or quadratic approximations, the derivatives of the cost function and the dynamics were calculated using PyTorch's built-in autograd functions.

2) Backward Pass: The cost-to-go function J_i is the cost over a partial control sequence $U_i = \{u_i, u_1, ..., u_{N-1}\}$ is given by:

$$J_i(x_i, U_i) = \sum_{n=i}^{N-1} \ell(x_n, u_n) + \ell_f(x_N). \tag{22}$$

Define the value function as the optimal cost-to-go as:

$$V(x_i) = \min_{U_i} J_i(x, U_i). \tag{23}$$

The problem can then be described as a minimization over a single control rather than a minimization over an entire sequence:

$$V(x_i) = \min_{u_i} [\ell(x_i, u_i) + V(f(x_i, u_i)]. \tag{24}$$

With $V(x_N) = \ell_f(x_N)$, the optimal cost-to-go at each time step can be calculated recursively, starting at x_N.

Let Q be the variation around an (x, u) pair for the argument of the minimum of 24. Via second-order expansion, this can be approximated as:

$$Q(\delta x, \delta u) \approx \begin{bmatrix} 0 & Q_{xx}^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix} \tag{25}$$

The expansion coefficients are:

$$Q_x = \ell_x + f_x^T V_x' \tag{26}$$

$$Q_u = \ell_u + f_u^T V_x' \tag{27}$$

$$Q_{xx} = \ell_{xx} + f_x^T V_{xx}' f_x + V_x' xx \tag{28}$$

$$Q_{uu} = \ell_{uu} + f_u^T V_{xx}' f_u + V_x' uu \tag{29}$$

$$Q_{ux} = \ell_{ux} + f_u^T V_{xx}' f_x + V_x' ux \tag{30}$$

The equations above are used for DDP since they require the second derivative of the dynamics function. For iLQR the terms in the equations that used the second derivatives of the dynamics function were dropped. Also, note that V_x' and V_{xx}' represent V_x and V_{xx} for the next step in the sequence.

With these terms an open-loop term $k = -Q_{uu}^{-1} Q_{uu}$ and a feed back gain term $k = -Q_{uu}^{-1} Q_{ux}$ are obtained.

V_x and V_{xx} are both initialized as $\ell_x(x_N)$ $\ell_{xx}(x_N)$ a the beginning of the backwards pass. They are then updated for the current step using:

$$V_x = Q_x - Q_{xu} q_{uu}^{-1} Q_u \tag{31}$$

$$V_{xx} = Q_{xx} - Q_{xu}q_{uu}^{-1}Q_{ux} \tag{32}$$

This is then recursively iterated from $n = N - 1$ to $n = 0$ to obtain a K_i and k_i for each time step.

3) Forward Pass: Once the backward pass is completed, a forward pass can be used to compute a new trajectory:

$$\hat{x}(0) = x(0) \tag{33}$$

$$\hat{u}(n) = u(n) + \alpha k(n) + K(n)(\hat{x}(n) - x(n)) \tag{34}$$

$$\hat{x}(n+1) = f(\hat{x}, \hat{u}(n)) \tag{35}$$

The α term is a backtracking line search parameter. For a non-linear system, the new trajectory may stray from the approximation, so the open-loop parameter can be scaled to ensure the cost decreases. To do so, a range of alphas was used where $0 < \alpha \leq 1$. Starting with $\alpha = 1$ complete the forward pass and compute the total trajectory cost. If the total trajectory cost increases, decrease α. Once the total trajectory cost decreases, accept the trajectory.

4) Regularization: To ensure Q_{uu} is positive definite, a regularization term μ is used. The equations for Q_{uu} and Q_{ux} are then:

$$Q_{uu} = \ell_{uu} + f_u^T(V'_{xx} - \mu I)f_u + V'_x u u \tag{36}$$

$$Q_{ux} = \ell_{ux} + f_u^T(V'_{xx} - \mu I)f_x + V'_x u x \tag{37}$$

If a backward pass fails and Q_{uu} is not positive definite, increase μ and retry the backward pass.

III. Results

Figure 2 shows the outcome of a test run using the DDP controller. The image shows snapshots of the simulated double pendulum every 0.5 seconds. The images are ordered left to right, then top to bottom. The DDP controller is able to swing up the pendulum consistently within the 150 steps that the test takes (since each time step is 0.05 seconds, the total runtime is 7.5 seconds). It is hard to tell from the image, but the cart is often not exactly at the center position when the test ends, because inputs must constantly be applied to the cart to keep the arm balanced.

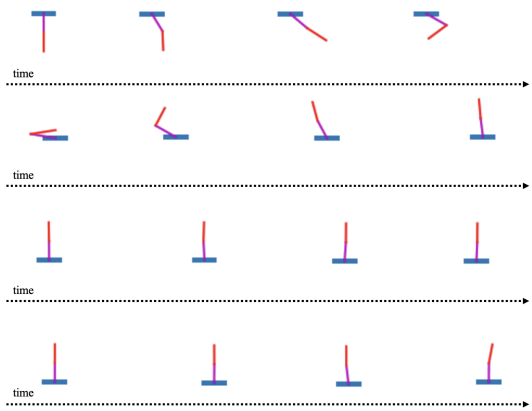

Fig. 2. DDP: horizon=30, steps=150

Figure 3 shows the mean square error of all the states with respect to the goal state. The error in all the states is driven to 0. Note that because the cart starts at rest, the error of the cart, joint 1, and joint 2 will all increase as forces are applied to the cart to move the joints to the correct position. Then these errors will drive back down to 0 as the cart stabilized with the links straight up. This is why the costs on the joint angles must be higher than the cost on the joint angle velocities, or else the system will be incentivized to never move.

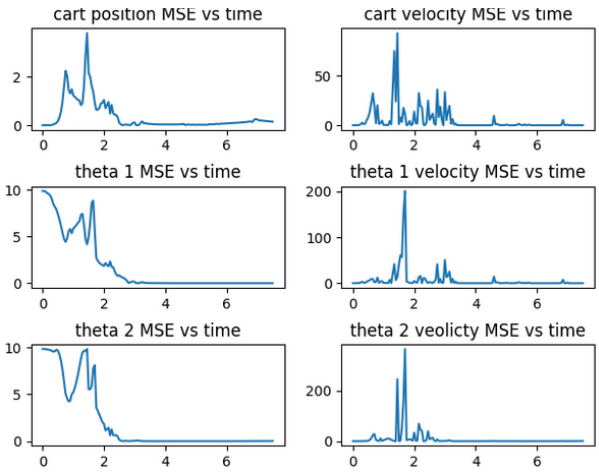

Fig. 3. DDP: The Mean Square Error of the states over time

Figure 4 shows the outcome of a test run using the iLQR controller using the same cost function, the same horizon, and the same number of steps as the DDP controller. Notice that this controller takes longer to stabilize the links in the upright position, and there were some tests where the system failed to stabilize.

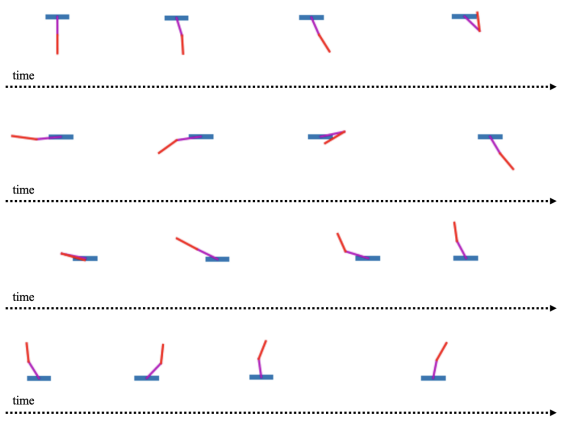

Fig. 4. iLQR: horizon=30, steps=150

Figure 5 shows the mean square error of all the states with respect to the goal state for iLQR. Notice that it does not converge as quickly as DDP.

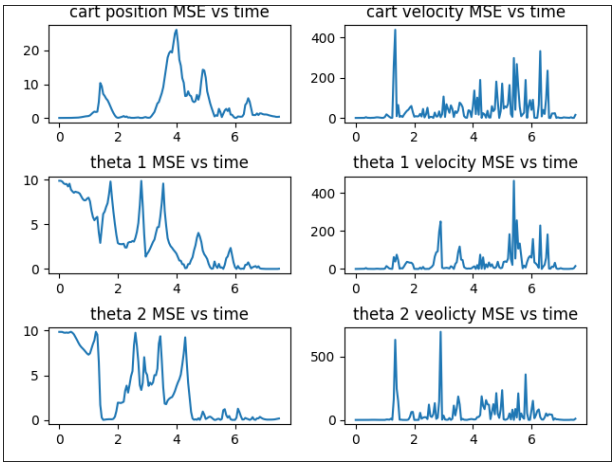

Fig. 5. iLQR: The Mean Square Error of the states over time

Figure 6 shows the outcome of a test run using the MPPI controller with the horizon set to 30, the number of steps set to 100, and a slightly different cost function where the Q matrix is tuned more to suit MPPI. MPPI is able to swing the pendulum up in a smoother manner than DDP or iLQR, and typically it stabilizes quicker.

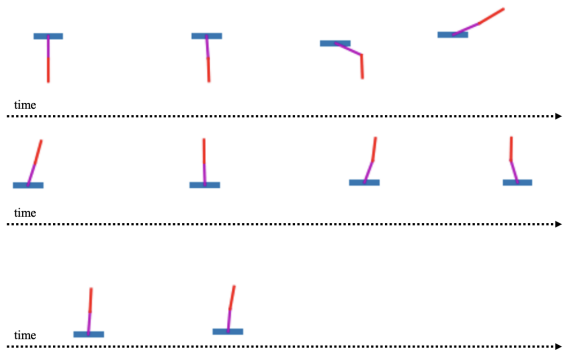

Fig. 6. MPPI: horizon=30, steps=100

Figure 7 shows the mean square error of all the states with respect to the goal state for MPPI. Notice that the error converges just as fast, if not faster than DDP.

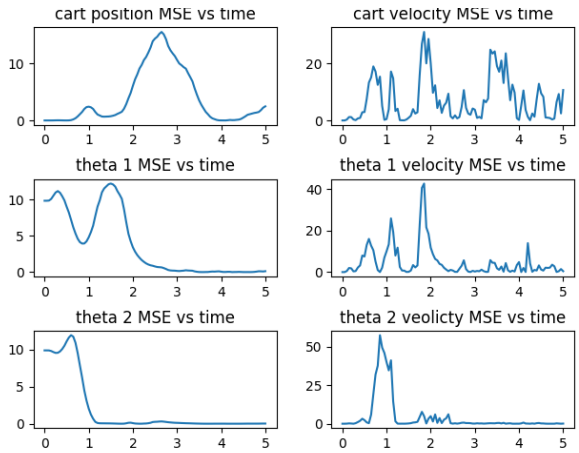

Fig. 7. MPPI: The Mean Square Error of the states over time

Each controller was able to reach the desired state. From these tests, it appears that MPPI produces the smoothest transition to the upright position. DDP was able to reach the desired the fastest. The iLQR controller took the longest to reach the desired state and had the worst performance with the final state not being as stable, as seen from the rapid changes in the angular velocities of the joints.

Each controller was also tested with the initial condition near the goal condition. The results are shown in Figures 8, 9, and 10.

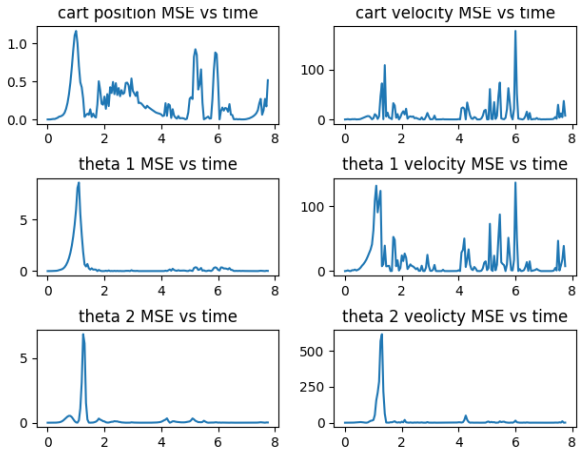

Fig. 8. DDP: The Mean Square Error of the states over time, initial state near goal state

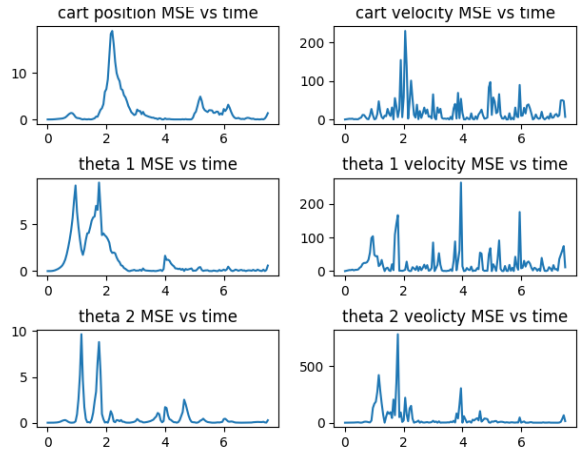

Fig. 9. iLQR: The Mean Square Error of the states over time, initial state near goal state

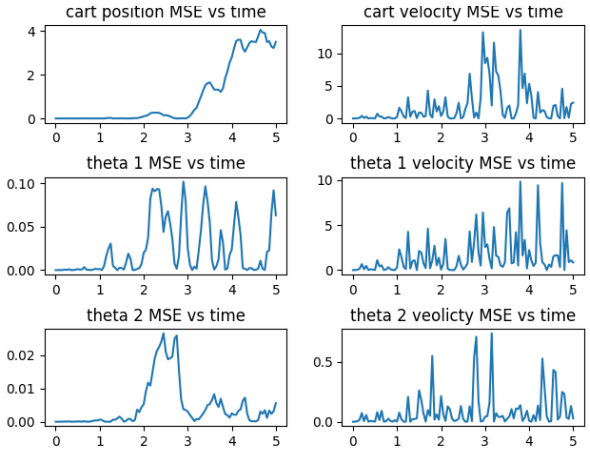

Fig. 10. MPPI: The Mean Square Error of the states over time, initial state near goal state

From the results, it can be seen that the best performing controller was the MPPI controller (Note: the y-axis on the graphs scales to the maximum value). Interestingly, the DPP and the iLQR controllers both dropped the pendulum initially as seen in the large error in θ_1 and θ_2. However, both controllers were able to recover and stabilize the pendulum in the upright position, with the DDP controller recovering faster.

IV. Conclusion

Model Predictive controllers can be used to stabilize the chaotic double pendulum system. DDP, iLQR, and MPPI controllers were used to find a trajectory that minimized the cost function over a horizon. Each controller was able to achieve the desired state of maintaining the double pendulum in the upright position, however, MPPI accomplished this task more smoothly and consistently. DDP performed the second best and iLQR performed the worst. The DDP and iLQR controllers used approximations of the cost and dynamics functions to update the trajectories while MPPI uses a sample-based method. For future work, we would like to test iLQR and DDP on learned dynamics. Since iLQR and DDP use derivatives of the dynamics for nonlinear systems, they would also be able to use learned dynamics from a neural network to approximate the dynamics and optimize the trajectories.

References

[1] I. J. Crowe-Wright, "Control theory: The double pendulum inverted on a cart," 2018.

[2] W. Edwards, G. Tang, G. Mamakoukas, T. Murphey, and K. Hauser, "Automatic tuning for data-driven model predictive control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7379–7385.

[3] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1168–1175.

[4] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4906–4913.

[5] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1433–1440.