Stabilization of a 2DOF Robot Arm

1st Ifham Abdul Latheef Ahmed, 5362616 *ALU Freiburg*
2nd Tarek Abou Chakra, 5450275 *ALU Freiburg*

***Abstract*—Stabilizing a 2DOF robot arm is one of the benchmark problems in control systems and many different methods have been utilized to solve it. In this report, we attempt solutions with a numerical approach rather than a classical control approach. Specifically, we attempt an Open Loop solution with Direct Multiple Shooting and a Closed Loop solution with Model Predictive Control (MPC). Our simulations show that the MPC controller performs much better than the Open Loop Solution and produces control signals much more intelligently while taking a shorter amount of time to produce each control signal.**

I. Introduction

The aim of this project is to a simulate a controller that is capable of stabilizing a 2DOF robot arm. Essentially, the two pivots of the arm are controlled by two motors whose outputs are chosen after solving an Optimal Control Problem (OCP). The system is further constrained by placing two circular objects to further limit the freedom of the robotic arms and make the choice of controls more restricted. The choice of the project was influenced by the highly nonlinear nature of the system as well as its use as a benchmark in classical control systems. Two approaches were attempted to solve this, an Open Loop and a Closed Loop Approach. The Open Loop approach utilized the Direct Multiple Shooting method while for the closed loop approach, Model Predictive Control (MPC) was implemented. We present our dynamic system model in Section (II). After which, the approaches used are described in Section (III). In Section (IV), we present the experimental framework. Finally, we present our results in Section (V) followed by the conclusion in Section (VI).

II. Model Dynamics

In this section, we present the method used to derive our system's Ordinary Differential Equations (ODEs). The system consists of two arms of lengths l_1 and l_2 with masses m_1 and m_2 respectively. The moment of inertias of each of the arms about their respective pivots are $I_1 = \frac{1}{3}m_1 l_1^2$ and $I_2 = \frac{1}{3}m_2 l_2^2$. We assume that the mass of both the arms are distributed evenly along the length. As such, the center of mass of each arm is located at exactly the center i.e at distances $lc_1 = l_1/2$ and $lc_2 = l_2/2$. g is the acceleration due to gravity. For simplicity, the force of gravity is assumed to only act at the center of each of the arms. The angle θ_1 is chosen to be the angle between inner arm and the negative y axis. The angle θ_2 is chosen to be the angle between the two arms. Two torques, τ_1 and τ_2 are the two control inputs applied at the respective pivots. The figure (1) illustrates the entire system.

A. Euler-Lagrange Equations

Due to the difficulty of deriving the equations using Newtonian mechanics, we opted to obtain them using Lagrange

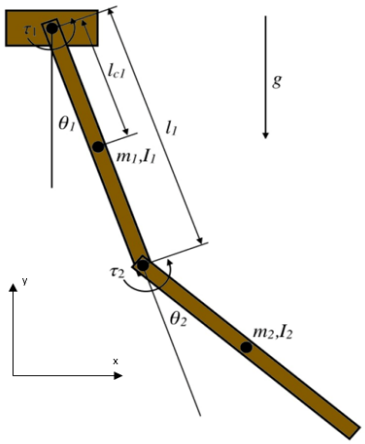


Fig. 1: System Ilustration

mechanics [1]. Lagrangian mechanics uses that principle of conservation of energy to obtain a system of ODEs. The Euler-Lagrange Formulation is as follows, where KE and PE refer to the Kinetic Energy and Potential Energy of the system respectively:

$$\mathcal{L} = KE - PE \tag{1}$$

The definitions of Kinetic and Potential Energy are as follows:

$$\begin{aligned} KE &= \frac{1}{2}mv^2 \\ PE &= mgh \end{aligned} \tag{2}$$

From the definitions above, the cartesian coordinates of the center of masses are given as follows:

$$\begin{aligned} x_1 &= lc_1 \sin(\theta_1) \\ y_1 &= lc_1 \cos(\theta_1) \\ x_2 &= l_1 \sin(\theta_1) + lc_2 \sin(\theta_1 + \theta_2) \\ y_2 &= l_1 \cos(\theta_1) + lc_2 \cos(\theta_1 + \theta_2) \end{aligned} \tag{3}$$

Taking the time derivative of the (3) where θ_1 and θ_2 are functions of time, we get the following:

$$\begin{aligned} \dot{x_1} &= lc_1 \cos(\theta_1)\dot{\theta_1} \\ \dot{y_1} &= -lc_1 \sin(\theta_1)\dot{\theta_1} \\ \dot{x_2} &= lc_1 \cos(\theta_1)\dot{\theta_1} + lc_2 \cos(\theta_1 + \theta_2)\dot{\theta_1} \\ &\quad + lc_2 \cos(\theta_1 + \theta_2)\dot{\theta_2} \\ \dot{y_2} &= -lc_1 \sin(\theta_1)\dot{\theta_1} - lc_2 \sin(\theta_1 + \theta_2)\dot{\theta_1} \\ &\quad - lc_2 \sin(\theta_1 + \theta_2)\dot{\theta_2} \end{aligned} \tag{4}$$

Using the equations for Kinetic and Potential Energies (2), we obtain the total Kinetic and potential energies for the system. We calculate these quantities from the center of masses of the arms.

$$\begin{aligned} KE = \frac{1}{2}(m_1 \dot{x_1}^2 + m_1 \dot{y_1}^2 + m_2 \dot{x_2}^2 + m_2 \dot{y_2}^2) \\ PE = m_1 g l c_1 \cos(\theta_1) + \\ m_2 g (l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)) \end{aligned} \tag{5}$$

We now formulate the Langrangian as defined in (1) and form the Euler-Lagrange equation to obtain equations for the torques τ_1 and τ_2 in the following matrix form:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} \frac{d}{dt} \cdot [\frac{\partial \mathcal{L}}{\partial \dot{\theta_1}}] - \frac{\partial \mathcal{L}}{\partial \theta_1} \\ \frac{d}{dt} \cdot [\frac{\partial \mathcal{L}}{\partial \dot{\theta_2}}] - \frac{\partial \mathcal{L}}{\partial \theta_2} \end{bmatrix} \tag{6}$$

After performing the partial and time derivatives and simplifying, we obtain a system of ordinary differential equations (ODEs) in matrix form as follows:

$$\mathbf{I} \cdot \begin{bmatrix} \ddot{\theta_1} \\ \ddot{\theta_2} \end{bmatrix} = \mathbf{B} \tag{7}$$

where $\mathbf{I}$ is known as the Inertia Matrix and $\mathbf{B}$ is the matrix specifying the torques along with the effects of the masses, angular velocities and gravity. The Intertia Matrix $\mathbf{I}$ is determined to be as follows:

$$\begin{bmatrix} I_1 + I_2 + m_2 l_1^2 + 2 m_2 l_1 l c_2 \cos(\theta_2) & I_2 + m_2 l_1 l c_2 \cos(\theta_2) \\ I_2 + m_2 l_1 l c_2 \cos(\theta_2) & I_2 \end{bmatrix} \tag{8}$$

The matrix $\mathbf{B}$ is determined to be as follows:

$$\begin{bmatrix} \tau_1 + 2 m_1 l_1 l c_2 \sin(\theta_2) \frac{d\theta_1}{dt} \frac{d\theta_2}{dt} + m_2 l_1 l c_2 \sin(\theta_2) (\frac{d\theta_2}{dt})^2 - \\ (m_1 l c_1 + m_2 l_1) g \sin(\theta_1) - m_2 g l_2 \sin(\theta_1 + \theta_2) \\ \tau_2 - m_2 l_1 l c_2 \sin(\theta_2) (\frac{d\theta_1}{d})^2 - \\ m_2 g l_2 \sin(\theta_1 + \theta_2) \end{bmatrix} \tag{9}$$

The system ODEs can be determined by taking the inverse of the Inertia Matrix $\mathbf{I}$ and multiplying both sides of (7) with it to obtain the equations with the angular accelerations, $\ddot{\theta_1}$ and $\ddot{\theta_2}$, as the subject. This is shown as follows:

$$\begin{bmatrix} \ddot{\theta_1} \\ \ddot{\theta_2} \end{bmatrix} = \mathbf{I^{-1}} \cdot \mathbf{B} \tag{10}$$

III. Methods

A. Open Loop Approach

Open Loop optimal control belongs to a category of control problems where feedback information is excluded from the system. Within this framework, a comprehensive simulation of the system's behavior is conducted throughout the entire time horizon. The solution to an Open Loop problem, which is the entire state and control trajectory, may be computed offline and may remain unchanged during the operation of the system. This category of optimal control problems offers notable benefits in situations where feedback is deemed unnecessary, and the model equations provide a comprehensive description of the system. In such cases, the influence of external variables beyond those explicitly captured by the model equations is negligible. Multiple approaches are employed to tackle these types of optimal control problems, including two prominent methods known as 'Single shooting' and 'Direct Multiple shooting' [2]. In the single shooting approach, also called the sequential approach, only one forward integration step is performed before every optimization problem. On the contrary, in multiple shooting, several forward integration steps can be performed before an optimization problem. As our first method, we implement an open-loop control system on a well-defined model with the application of direct multiple shooting. This method makes the Jacobian and Hessian Matrices block diagonal and thus increases matrix sparsity when compared with Direct Single Shooting.

Objective Function. For our OCP, we consider the two circular objects that may come into contact with the second arm when it is fully extended as hard constraints. To address this, we incorporate these constraints into the objective function by calculating the Euclidean distance between P points located on the second arm, including its start and end points, and the center of the circle. If this distance is smaller than the radius, it indicates that the arm is inside the circle and violates the constraint. Additionally, hard constraints on the maximum angular velocities are imposed in order to prevent them from rotating at excessively high speeds, which could lead to damage in real-life scenarios. To simplify the notation for the final objective function, we specify the following vectors for the input torques and the angular velocities:

$$\begin{aligned} \omega_k &= \begin{bmatrix} \omega_k^{[1]} & \omega_k^{[2]} \end{bmatrix}^T \quad k = 0, \ldots, N-1 \\ u_k &= \begin{bmatrix} \tau_k^{[1]} & \tau_k^{[2]} \end{bmatrix}^T \quad k = 0, \ldots, N-1 \end{aligned} \tag{11}$$

Similarly, we define the same for the center of both the circular objects:

$$C_1 = \begin{bmatrix} X_{\mathrm{C1}} & Y_{\mathrm{C1}} \end{bmatrix}^T \quad C_2 = \begin{bmatrix} X_{\mathrm{C2}} & Y_{\mathrm{C2}} \end{bmatrix}^T \tag{12}$$

We now define P points on the second arm according to the Cartesian coordinate system, knowing that the start point of the first rod is at the origin:

$$\begin{aligned} X_k &= L_1 \cdot sin(\theta_k^{[1]}) \quad k = 0, \ldots, N-1 \\ Y_k &= L_1 \cdot cos(\theta_k^{[1]}) \quad k = 0, \ldots, N-1 \end{aligned} \tag{13}$$

$$\begin{aligned} X_k^{[p]} &= X_0 + L_2 \cdot (p/P) \cdot sin(\theta_k^{[1]} + \theta_k^{[2]}), \\ Y_k^{[p]} &= Y_0 + L_2 \cdot (p/P) \cdot cos(\theta_k^{[1]} + \theta_k^{[2]}), \\ & p = 0, \ldots, P, \\ & k = 0, \ldots, N-1 \end{aligned} \tag{14}$$

where, due to multiple indices, the notation $\theta_k^{[1]}$ refers to θ_1 at timestep k and $X_k^{[p]}$ refers to the x-coordinate of the p^{th} point along the second arm at timestep k. The notation is similar for the other quantities. Equation 13 refers to the tip of the first arm (start of second arm) for all the time indices of the horizon N and equation 14 refers to all the points along

the second arm including the tip of the first arm, also for all the time indices of the horizon. With these definitions, we formulate the objective function for state vector x and input vector u as:

$$\begin{aligned} &\min_{x\in\mathbb{R}^{(N+1)\times n_x}, u\in\mathbb{R}^{N\times n_u}} f(x,u) \\ &\text{s.t} \\ &g(x,u)=0 \\ &h(x,u)\leq 0 \end{aligned} \tag{15}$$

where $f(x,u)$ is defined as follows:

$$\frac{1}{2}(\sum_{i=0}^{N-1}\|u_i\|_2^2+\|x_i-x_f\|_2^2)+\frac{1}{2}\|x_N-x_f\|_2^2 \tag{16}$$

and where x_f is the desired state and $\bar{x}_0$ is preferred initial state. The equality constraints, $g(x,u)$, defined as follows:

$$\begin{bmatrix} x_0-\bar{x}_0 \\ x_{N-10}-x_f \\ \vdots \\ x_N-x_f \\ x_1-F(x_0,u_0) \\ \vdots \\ x_N-F(x_{N-1},u_{N-1}) \end{bmatrix}=0 \tag{17}$$

where $F(x,u)$ is the RK4 integration step performed using the Model Dynamics. We also specify multiple end constraints to force the arm to reach the desired state before the simulation time to show the controller's ability to maintain the state. The inequality constraints $h(x,u)$ are defined as follows:

$$\begin{bmatrix} u_k-u_{\max} \\ u_{\min}-u_k \\ \omega_k-\omega_{\max} \\ \omega_{\min}-\omega_k \\ r_1^2-((X_k^{[p]}-X_{C1})^2+(Y_k^{[p]}-Y_{C1})^2) \\ r_2^2-((X_k^{[p]}-X_{C1})^2+(Y_k^{[p]}-Y_{C1})^2) \end{bmatrix}\leq 0 \quad \begin{matrix} p=0,\ldots,P, \\ k=0,\ldots,N-1 \end{matrix} \tag{18}$$

B. Closed-Loop Approach

The previous approach of Open Loop Control to determine a precomputed control trajectories can suffer from Model-Plant mismatch. As such, the actual final state might not be the same as the desired final state. To account for this, we employ the use of closed loop feedback control. Specifically, we implement Model Predictive Control.

Essentially, we observe the true current state of the system $\tilde{x}_0$ and then solve an open loop OCP on a limited horizon of N with the starting state being $\tilde{x}_0$. After which, we only apply the first control action u_0^* on the system. Following that, we shift the optimization horizon forward by one time step and repeat the procedure. This procedure allows the controller to correct itself continuously, even in the presence of sensor or process noise. By continuously adapting to real-time conditions, MPC enhances the control performance.

Objective Function. With regards to the cost function, it was slightly modified to accommodate the measurements. However, the original hard constraint for the final desired state was removed due to it making the problem infeasible as the final state in the horizon cannot be known before the controls are determined. Instead, the objective function focuses on minimizing the path to the end state throughout the horizon by incorporating it as a soft constraint. Sensor noise, (ϵ_{noise}), was added to the first state of the problem. This only applies to the cost function and not the constraints and is shown as follows:

$$x=\begin{bmatrix}\tilde{x}_0+\epsilon_{noise} & x_1 & \ldots & x_N\end{bmatrix}^T \tag{19}$$

Using the above x, the objective or cost function $f(x,u)$ for the MPC controller is shown below:

$$\frac{1}{2}(\sum_{i=0}^{N-1}\|u_i\|_{\mathbf{R}}^2+\|(x_i-x_f)\|_{\mathbf{Q}}^2)+\frac{1}{2}\|(x_N-x_f)\|_{\mathbf{Q}}^2 \tag{20}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are the weighting matrices. The equality constraints, $g(x,u)$, are also modified as follows:

$$\begin{bmatrix} \tilde{x}_0-\bar{x}_0 \\ x_1-F(\tilde{x}_0,u_0) \\ \vdots \\ x_N-F(x_{N-1},u_{N-1}) \end{bmatrix}=0 \tag{21}$$

The inequality constraints $h(x,u)$ remain the same. However, they are now only defined for the shorter horizon of the MPC controller

IV. Experimental Framework

We now present the parameters chosen for simulation. We first begin by describing the State representation in section (IV-A). We follow that up by describing the chosen values for the system/model parameters, in section (IV-B), such as the mass and length of the arms, etc. After which, in section (IV-C), we define our simulation parameters such as the simulation time, sampling time, etc.

A. State Representation

We first describe how the system state space is represented with respect to the model dynamics described in section (II) and the Cartesian coordinate system. The state and control vector, $x\in\mathbb{R}^4$ and $u\in\mathbb{R}^2$, of our system are defined as follows:

$$x=\begin{bmatrix}\theta_1 & \theta_2 & \omega_1 & \omega_2\end{bmatrix}^T u=\begin{bmatrix}\tau_1 & \tau_2\end{bmatrix}^T \tag{22}$$

Taking the time derivative of the state vector, we obtain:

$$\dot{x}=\begin{bmatrix}\omega_1 & \omega_2 & \ddot{\theta}_1 & \ddot{\theta}_2\end{bmatrix}^T \tag{23}$$

where $\ddot{\theta}_1=\dot{\omega}_1$ and $\ddot{\theta}_2=\dot{\omega}_2$. Here, the last two elements of $\dot{x}$ are equal to the system ODEs derived in section 10 . The first two elements of $\dot{x}$ are the same as the last two elements of x. Based on illustration 1, we choose the initial state of the system, $\bar{x}_0$, to be when the robot arm is stationary and fully

extended on the negative y-axis. In vector form, we illustrate it as follows:

$$\bar{x}_0 = \begin{bmatrix}\theta_1 & \theta_2 & \omega_1 & \omega_2\end{bmatrix}^T = \begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}^T \tag{24}$$

The desired or reference state, x_f, is when the robot arm is vertically inverted and stationary on the positive y-axis. In vector form, this state is illustrated as follows:

$$x_f = \begin{bmatrix}\theta_1 & \theta_2 & \omega_1 & \omega_2\end{bmatrix}^T = \begin{bmatrix}\pi & 0 & 0 & 0\end{bmatrix}^T \tag{25}$$

The values for θ_1 and θ_2 are defined with θ_1 measured anti-clockwise from the negative y-axis to the first arm. Similarly, the angle θ_2 is measured anti-clockwise from the inner or first arm to the second arm.

B. System Parameters

In this section, we present our choices for the model parameters. These choices have been taken from [3]. The units are all defined as SI units [4]. As such, the calculated values will be in terms of SI units as well. The parameters are listed in table I below:

Arm	Length (m)	Center of Mass (m)	Mass (kg)
1	$l_1 = 1$	$lc_1 = 0.5$	$m_1 = 1$
2	$l_2 = 1$	$lc_2 = 0.5$	$m_2 = 1$

TABLE I: Model Parameters

while $g = 9.81\ \frac{\mathrm{m}}{\mathrm{s}^2}$ is a constant.

C. Simulation Parameters

In this section, we describe and justify our choices for the parameters used in our simulation and controllers. For all our experiments, we simulate for a time $T_{\text{sim}} = 10$ s. For the open loop approach, we use a timestep of $h = 0.1$ s for the RK4 discretization. This value is chosen to reduce the approximation error of the ODEs. For the closed loop MPC approach, where we have a noisy state feedback, we assume that the sampling time of the sensor is $T_s = 0.1$ s. For simplicity, we also keep timestep for the RK4 discretization the same. The horizon length is chosen to be $N = 2$ s. Using a longer horizon would allow the controller to predict further into the future at the cost of a larger computation time. However, due to the sampling time bound, we use the above horizon length to better approximate reality. We also limit the input torque magnitude to $10\ \mathrm{Nm}$ while also limiting the magnitude of the angular speeds to $2.5\ \frac{\mathrm{rad}}{\mathrm{s}}$. These are motivated by the fact that in an actual system, there will be operational and structural limits. The limits are shown as follows for both the control inputs and angular velocities:

$$\tau_{1,2} \leq |10\ \mathrm{Nm}| \quad \omega_{1,2} \leq |2.5\ \frac{\mathrm{rad}}{\mathrm{s}}| \tag{26}$$

For the open loop approach, we do not employ the use of weighting matrices in the objective functions due to the presence of an end constraint. We have observed that the weights are unnecessary to obtain an optimal solution. For the closed loop MPC approach however, weighting matrices were specified due to the fact that an end constraint cannot be added to the OCP. The weighting matrix $\mathbf{Q}$ ensures that the objective function heavily penalizes the difference between the states in the horizon and the final desired state. We choose to penalize the angular positions more heavily than the angular velocities of the arms as reaching and maintaining the desired state is not feasible in one horizon. As such, to allow for continuity, we do not greatly penalize the angular velocities. We also weight the control variables, with a matrix $\mathbf{R}$, to ensure that the least possible torque magnitudes are used. However, due to the lack of an end constraint, the weighting cannot be too large. After some testing and tuning, we found that the following weighting matrices were the most suitable:

$$\mathbf{Q} = \begin{bmatrix}10^3 & 0 & 0 & 0\\ 0 & 10^2 & 0 & 0\\ 0 & 0 & 10^{-1} & 0\\ 0 & 0 & 0 & 10^{-1}\end{bmatrix} \quad \mathbf{R} = \begin{bmatrix}10^{-2} & 0\\ 0 & 10^{-2}\end{bmatrix} \tag{27}$$

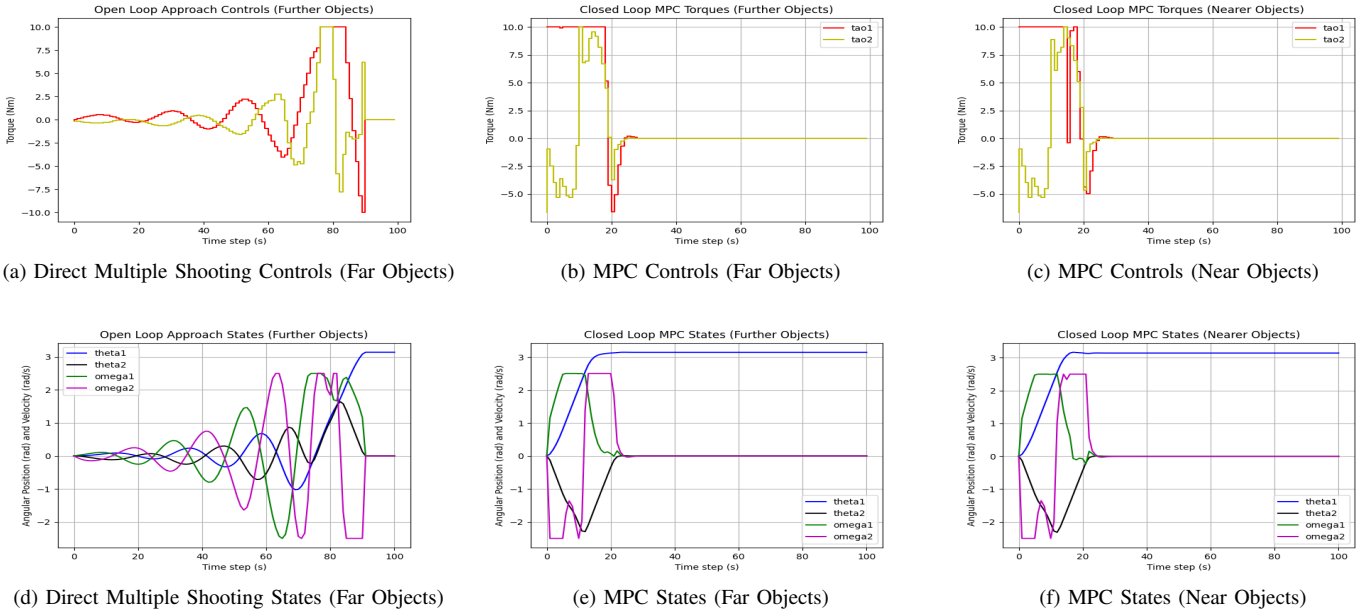
For simplicity, the sensor noise is assumed to be additive Gaussian and is also assumed to be independently and identically distributed. The noise mean is $\mu = 0$ with a standard deviation of $\sigma = 0.25$. The same noise model is assumed for all the measured states. In all our initial simulations, both the circular objects, of radius $r = 0.5$ m are placed at the coordinates $(1.75, 1.75)$ and $(1.75, -1.75)$ respectively. Finally, the number of points on the second arm in the constraints is set to $P = 5$

V. Results and Discussion

Here, we present the results obtained from our simulations. The simulations and the optimization problems were formulated and solved using the Casadi framework [5] with the IPOPT solver [6] while the visualizations were performed using the matplotlib library on python [7].

Results for the Open Loop Approach. As can be seen from figures (2a) and (2d), the open loop approach reaches the desired state while avoiding the objects. This approach uses the fact that the objects are far enough initially to swing the arms repeatedly until enough momentum is gained. After which, a final control is applied to get to the desired state. This is evidenced by the sinusoidal nature of the plots. Additionally, the angular speeds only reach the limit towards the end during the final push by the controller. The OCP also contains a large number of variables, due to it optimizing for the entire simulation time, and as such takes much longer to obtain an optimal solution, although it can be done offline. In our simulation, it required 51.6 s although this is dependant on the hardware.

Results for the closed Loop MPC. From figures (2b) and (2e), it can be seen that the MPC controller reaches and maintains the desired state much more quickly. This can be attributed to the fact that the controller does not have access to a larger horizon. As a result, the controller attempts to reach the desired state purely using the torques and does not utilize a swinging motion. As such, this causes the torques and angular velocities to clip early on. The alternative can



(a) Direct Multiple Shooting Controls (Far Objects)

(b) MPC Controls (Far Objects)

(c) MPC Controls (Near Objects)

(d) Direct Multiple Shooting States (Far Objects)

(e) MPC States (Far Objects)

(f) MPC States (Near Objects)

Fig. 2: Results of all Experiments

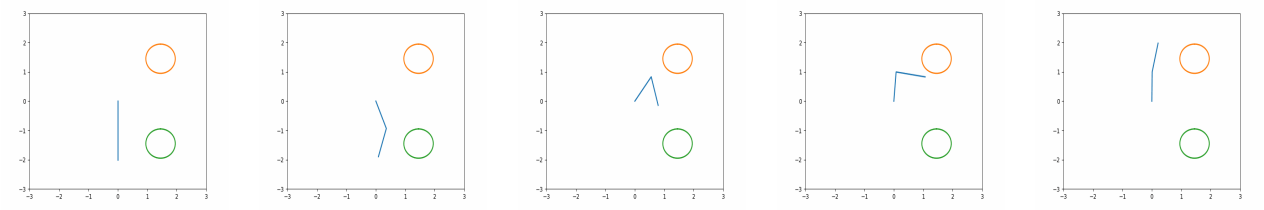


Fig. 3: Snapshots of the MPC Simulation with Nearer Objects

be enabled by increasing the horizon length at the cost of a larger computation time, which in our case is not desired due to the sensor sampling time bound. The MPC controller is also able to give an optimal solution if the objects are moved closer to the arms. In figures (2c) and (2f), the results for nearer objects are shown. Specifically, the circular objects were moved to the respective coordinates: $(1.45, 1.45)$ and $(1.45, -1.45)$ keeping the radius the same. From these results, it can be concluded that the MPC controller is still able to reach the desired state without much difficulty albeit with slightly different control trajectories to avoid the objects. In figure (3), we present some frames of our simulation with the MPC Controller for nearer objects. Our MPC controllers required, on average, 0.14 s and 0.17 s respectively. As this was just a simulation, we do not expect the sampling time bound to be respected firmly. However, it an actual controller, the the algorithm must terminate within the bound.

VI. Conclusion

We have presented two approaches to solve the stabilization problem of a 2DOF Robot Arm. Specifically, an open Loop and a Closed Loop MPC method. Our results have demonstrated that the MPC controller is much more intelligent in its choice of controls due to state feedback while also maintaining the feasibility of the OCP in every timestep. In contrast, the open loop solution approaches infeasibility if the objects are brought closer or if the simulation time of the horizon is increased. Additionally, the MPC controller is able to generate the optimal control much quicker than the Open Loop method while also being able to account for noisy measurements which is not possible with the latter.

References

[1] H. Goldstein, C. P. Poole, and J. L. Safko, *Classical mechanics. 3rd.* Addison Wesley, 2002.

[2] S. Gros and M. Diehl, "Numerical optimal control (draft)," 2019.

[3] N. M. Ghaleb and A. A. Aly, "Modeling and control of 2-dof robot arm," *International Journal of Emerging Engineering Research and Technology*, vol. 6, no. 11, pp. 24–31, 2018.

[4] BIPM, *Le Système international d'unités / The International System of Units ('The SI Brochure')*, 9th ed. Bureau international des poids et mesures, 2019.

[5] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[6] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, pp. 25–57, 2006.

[7] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.