Point-to-Point Kinematic Control of the 3-Joint Robotic Arm in the Presence of Obstacles

Xinlei Zhang, xzhan245@ncsu.edu
Jason Huang, jhuang16@ncsu.edu
Instructor: Prof. Donggun Lee
Department of Mechanical and Aerospace Engineering, North Carolina State University

I. Introduction

Point-to-point motion control of a multi-joint robotic arm is a well-solved problem and has been widely deployed in industry which can be further broken down to three separate subtasks [1-2]. The three subtasks are path planning, trajectory planning and trajectory following or tracking. Since the current actuators (motors) used in robotic arms can directly receive the angular velocity commands and reach the target velocity with their own embedded low-level controller, kinematic control is feasible for the motion control of robotic arms, avoiding the dynamical modeling process and simplifying the control effort. Hence, only joint velocity commands need to be planned.

As for the path and trajectory planning, avoiding potential collision between the end-effector and obstacles is essential in real-world robotic applications, which require sophisticated control of the robotic arm to consider the end-effector. Directly adjusting the joint angles of the robotic arm is one intuitive way to move the end-effector, but the relationship between the joint angles and end-effector is complex and nonlinear which is compounded with how one joint angular position affects the position of the joints down the line. To directly control the end-effector, the transformation from the joint angles to end-effector should be derived and combined into the control design.

In this project, we utilized the optimal control methods learned from this course to solve the point-to-point motion control problem for the PUMA 560 robotic arm, where path and trajectory planning are addressed simultaneously to compute the optimal joint velocity commands while minimizing energy consumption, considering joint angle and velocity limits and avoiding collision between end-effector and obstacles. The methods can be extended to other robotic arms. This report also aims to help readers understand the provided code and reproduce the results. The full materials for this project can be found in: GitHub Repo.

II. Preliminaries

A. CasADi

CasADi was the main tool used in this project for solving the optimization problem, which is an open-source tool for non-linear optimization and algorithmic differentiation, supporting Octave/MATLAB, Python and C++ [3].

B. Robotic Arm Kinematics

Given how most robotic arms are controlled via joint angles, but the position of the end-effector is desired, we need to be able to convert joint angles into the corresponding position coordinates the end-effector is at. Since only the position of the end-effector is concerned in this project, the last three joints of PUMA 560 were neglected, which constitutes a spherical joint that only change the orientation of the end-effector. Breaking down the 3-joint robotic arm simplified from PUMA 560, the arrangement can be represented as the following kinematic diagram.

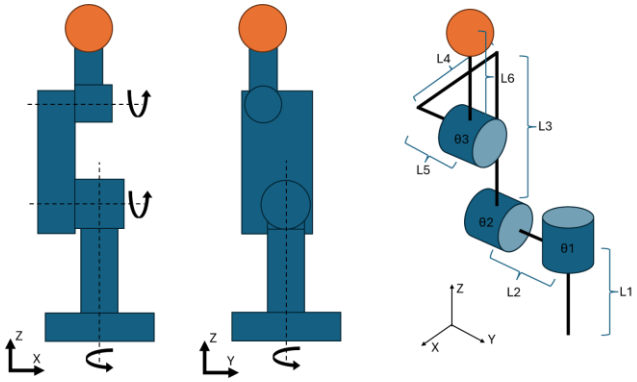

Fig. 1. Kinematic diagram of the 3-joint robotic arm. The left subfigure shows 2 DoFs and the middle subfigure shows 1 DoF. The right subfigure shows the link structure of the robotic arm and depicts its zero configuration.

From the kinematic diagram in Fig. 1, the transformation function which converts the joint angles $\theta = [\theta_1, \theta_2, \theta_3]^T$ to the position of the end-effector $T(\theta)$ can be written as,

$$T(\theta) = T_z(L_1)R_z(\theta_1)T_y(-L_2)R_y(-\theta_2)T_z(L_3) \\ T_x(L_4)T_y(L_5)R_y(-\theta_3)T_z(L_6) \quad (1)$$

where T_x, T_y, T_z, R_y and R_z are elementary transformations for translation and rotation along the axes of world frame, given in Eq. (2).

It should be noted that there exists a joint angle offset between the joint angles in Eq. (1) and the PUMA 560 robot object in Robotics Toolbox for MATLAB, which is addressed in the MATLAB script file *forward_kinematics_3D.m*.

C. Robotics Toolbox for MATLAB

The PUMA 560 robot object (SerialLink) provided in the Robotics Toolbox for MATLAB developed by Peter Corke [1] is utilized for visualizing the PUMA 560 robotic arm in MATLAB.

$$T_x(L) = \begin{bmatrix} \text{diag}\left(\begin{bmatrix}1\\1\\1\end{bmatrix}\right) & \begin{bmatrix}L\\0\\0\end{bmatrix} \\ [0 \quad 0 \quad 0] & 1 \end{bmatrix}, T_y(L) = \begin{bmatrix} \text{diag}(\begin{bmatrix}1\\1\\1\end{bmatrix}) & \begin{bmatrix}0\\L\\0\end{bmatrix} \\ [0 \quad 0 \quad 0] & 1 \end{bmatrix}$$

$$T_z(L) = \begin{bmatrix} \text{diag}\left(\begin{bmatrix}1\\1\\1\end{bmatrix}\right) & \begin{bmatrix}0\\0\\L\end{bmatrix} \\ [0 \quad 0 \quad 0] & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \begin{bmatrix}\cos(\theta) & 0 & \sin(\theta)\\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta)\end{bmatrix} & \begin{bmatrix}0\\0\\0\end{bmatrix} \\ [0 \quad 0 \quad 0] & 1 \end{bmatrix} \tag{2}$$

$$R_z(\theta) = \begin{bmatrix} \begin{bmatrix}\cos(\theta) & -\sin(\theta) & 0\\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1\end{bmatrix} & \begin{bmatrix}0\\0\\0\end{bmatrix} \\ [0 \quad 0 \quad 0] & 1 \end{bmatrix}$$

III. Optimal Control Problem Formulation

A. Problem Statement

To formally define the problem, a sequence of joint velocity commands, $\{\alpha_0, \dots, \alpha_K\}$, are desired, which will drive the end-effector of the robotic arm to the target position in finite time while considering the joint angle and velocity limits, minimizing energy consumption and avoiding known static obstacles. The robotic arm and obstacles are shown in Fig. 2.

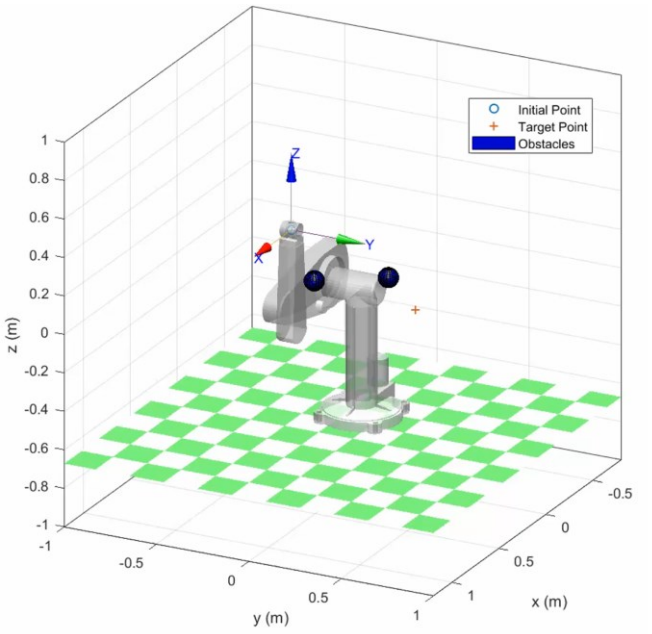

Fig. 2. Problem illustration. Robotic arm PUMA 560 is in its initial configuration. The cross marker depicts the target point, and these two blue spheres represent the obstacle regions.

B. Optimization Formulation

According to the problem statement, this problem can be formulated with the optimization framework as follows,

$$\begin{aligned} &\inf_{\alpha_0,\dots,\alpha_K,} \mathcal{L}(X_0, \dots, X_K, \alpha_0, \dots, \alpha_K) \\ &\boldsymbol{s.t.}\, X_{k+1} = \mathrm{F}(X_k, \alpha_k), \qquad \text{and} \\ &\begin{bmatrix} g_1(X_k) \\ g_2(\alpha_k) \\ C(\mathrm{T}(X_k)) \end{bmatrix} \leq \mathbf{0}, \qquad \text{for } k = 0, \dots, K \end{aligned} \tag{3}$$

where $X_k \in \mathbb{R}^3$ is the column vector including three joint angles, $\alpha_k \in \mathbb{R}^3$ contains the angular velocities of these three joints, $\mathcal{L}(\cdot): (\mathbb{R}^3)^k \circ (\mathbb{R}^3)^k \mapsto \mathbb{R}$ defines the objective function of the problem, $g_1(\cdot)$ and $g_2(\cdot)$ are state and control boundary functions respectively, $C(\cdot)$ is the obstacle characteristic function, and $\mathrm{F}(\cdot)$ is the discrete dynamical equation. The specific form of the objective function (also called cost function in control scenario) varies within different methods used to solve the optimization problem.

In this problem, $\mathrm{F}(\cdot)$ is assumed as a constant angular velocity model,

$$X_{k+1} = \mathrm{F}(X_k, \alpha_k) = X_k + \Delta t \alpha_k \tag{4}$$

State and control boundary functions are,

$$g_1(X_k) = \begin{bmatrix} X_k - X_{max} \\ X_{min} - X_k \end{bmatrix}, g_2(\alpha_k) = \begin{bmatrix} \alpha_k - \alpha_{max} \\ \alpha_{min} - \alpha_k \end{bmatrix} \tag{5}$$

where the boundary limits are dependent on the physical limitations of the actuators used in the robotic arm, and these values for PUMA 560 are obtained from [1], given in Table X as well.

$C(\cdot)$ detects if the end-effector of the PUMA 560 collides with obstacles, which is simplified in this problem as determining if the center position of the end-effector lies within the two spherical regions. $C(\cdot)$ can be written as,

$$C\big(\mathrm{T}(X_k)\big) = \begin{bmatrix} \text{sigmoid}(d_k^1) \\ \text{sigmoid}(d_k^2) \end{bmatrix} \tag{6}$$

where the superscript i in d_k^i denotes the i-th obstacle, and

$$d_k^i = (r^i)^2 - \left\| \mathrm{T}(X_k) - p^i \right\|_2^2 \tag{7}$$

with r^i and p^i are the radius and center position vector of the i-th obstacle respectively, and $\|\cdot\|_2$ computes the L-2 norm of the vector.

IV. Methods

This section explains the two methods utilized to solve the proposed optimization problem step by step, i.e., direct optimization and model predictive control. Their results will be presented, compared, and discussed in the next section.

A. Direct Optimization with the Multiple-Shooting Method

Direct optimization method is straightforward to solve the optimization problem, but it needs a specific time horizon K to run any optimization algorithm, e.g., interior-point optimizer (IPOPT) which is used in this project. Hence, the time horizon K is a hyperparameter which will significantly influence

optimization results on computation complexity and the existence of solutions.

Time horizon K can be tuned by adjusting the terminal time T with their relation as,

$$K = \text{floor}\left(\frac{T}{\Delta t}\right) + 1 \tag{8}$$

where Δt is the temporal discretization step.

After specifying the time horizon K, the objective function $\mathcal{L}(\cdot)$ can be defined as,

$$\mathcal{L} = \sum_{k=0}^{K-1} \Delta t L(\mathrm{T}(X_K), \alpha_K) + g(\mathrm{T}(X_K), \alpha_K) \tag{9}$$

where $\mathcal{L}$ is divided into two parts,

$$L(\mathrm{T}(X_K), \alpha_K) = (\mathrm{T}(X_K) - P^t)^{\mathrm{T}} Q (\mathrm{T}(X_K) - P^t) + \alpha_k^{\mathrm{T}} R \alpha_K \tag{10}$$

P^t is the target position, Q and R are tunable parameters for solution performance, and

$$g(\mathrm{T}(X_K), \alpha_K) = (\mathrm{T}(X_K) - P^t)^{\mathrm{T}} Q_K (\mathrm{T}(X_K) - P^t) \tag{11}$$

$L(\mathrm{T}(X_K), \alpha_K)$ is the stage cost including the squared distance to the target position and the squared angular velocity which is proportional to the energy cost. $g(\mathrm{T}(X_K), \alpha_K)$ is the terminal cost that reflects the final distance to the target position. Q_K is also a tunable parameter, and the values of these tunable parameters are given below.

$$Q = R = \text{diag}([1,1,1]), Q_K = \text{diag}([10,10,10]) \tag{12}$$

With the forward Euler method for discretization and multiple-shooting formulation for optimization, the optimization problem can be expanded as,

$$\begin{aligned} \min_{\substack{\alpha_0,\dots,\alpha_{K-1} \\ X_1,\dots,X_K}} & \sum_{k=0}^{K-1} \Delta t L(\mathrm{T}(X_K), \alpha_K) + g(\mathrm{T}(X_K), \alpha_K) \\ \boldsymbol{s.t.}\ & X_{k+1} = \mathrm{F}(X_k, \alpha_k), k = 0, \dots, K-1 \\ & \begin{bmatrix} g_1(X_k) \\ g_2(\alpha_k) \\ C(\mathrm{T}(X_k)) \end{bmatrix} \leq \mathbf{0}, k = 1, \dots, K \end{aligned} \tag{13}$$

B. Model Predictive Control

Although the time horizon K can be manually pre-defined after some tunning efforts, the time cost for running the direct optimization algorithm is still unacceptable for real-time robotic applications. Model predictive control (MPC) utilizes Bellman optimality principle to split the whole optimization problem into sub-problems with receding horizons, which increases the control frequency effectively [4]. Even though the receding horizon H is also a tunable hyperparameter, tuning this parameter has a clearer practical meaning than directly determining the termination time in direct optimization. The MPC formulation of Eq. (3) based on Eq. (13) with the receding horizon H is given below.

$$\begin{aligned} \min_{\substack{\alpha_{k|k},\dots,\alpha_{k+H-1|k}, \\ X_{k|k},\dots,X_{k+H-1|k}}} & \sum_{s=k}^{k+H-1} \Delta t L(\mathrm{T}(X_{s|k}), \alpha_{s|k}) \\ & + g(\mathrm{T}(X_{k+H|k}), \alpha_{k+H|k}) + V(X_{k+H|k}) \\ \boldsymbol{s.t.}\ & X_{s+1|k} = \mathrm{F}(X_{s|k}, \alpha_{s|k}), s = k, \dots, k+H-1 \\ & \begin{bmatrix} g_1(X_{s|k}) \\ g_2(\alpha_{s|k}) \\ C(\mathrm{T}(X_{s|k})) \end{bmatrix} \leq \mathbf{0}, s = k, \dots, K \end{aligned} \tag{14}$$

where $V(\cdot)$ is the cost-to-go value function. Candidate functions can be used to approximate $V(\cdot)$, but the quality of candidate functions will greatly impact the optimality of the solution. $V(\cdot)$ is simply set as 0 in our problem.

MPC is an iterative algorithm, and a terminating condition is required to run MPC. The distance from the end-effector to the target point is used as the threshold ϵ,

$$\epsilon = \|(\mathrm{T}(X_k) - P^t)^{\mathrm{T}} (\mathrm{T}(X_k) - P^t)\|_2 \tag{15}$$

V. Simulation Results

These two methods are implemented in MATLAB with the open-source tool CasADi, and the results are visualized with the PUMA 560 robot object provided in the Robotics Toolbox for MATLAB. The MATLAB script is running on the Intel® Core™ i7-10750H CPU @ 2.60 GHZ.

A. Direct Optimization with the Multiple-Shooting Method

The terminal time T is set as 5 s and the corresponding time horizon K equals 26 after fine-tuning. $\Delta t = 0.2$ s for all the simulation. The final trajectory of the end-effector with the optimized control is shown in Fig. 3.

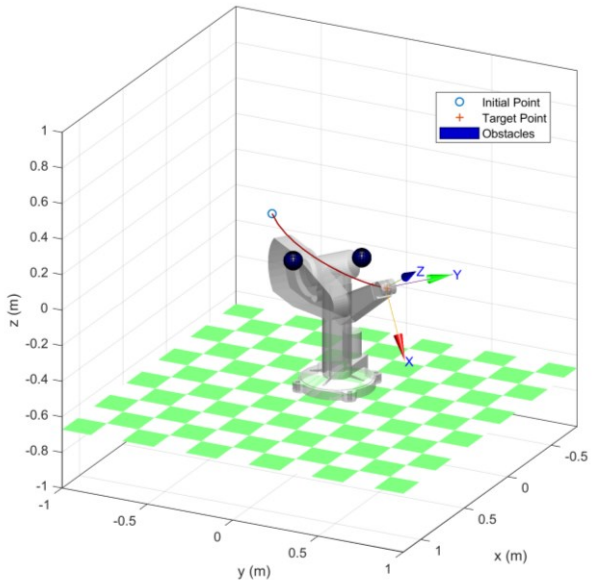

Fig. 3. Position trajectory of the end-effector with direct optimization.

The distance between the end-effector and the target point is shown in Fig. 4, where the optimization algorithm takes 5.2 s to find the optimal solution first, and then the optimized commands are executed by the robotic arm, which takes 5 s as pre-defined by the terminal time T. As running the IPOPT (Interior Point Optimizer) algorithm will take 5.2 s, this method cannot be used for real-world applications.

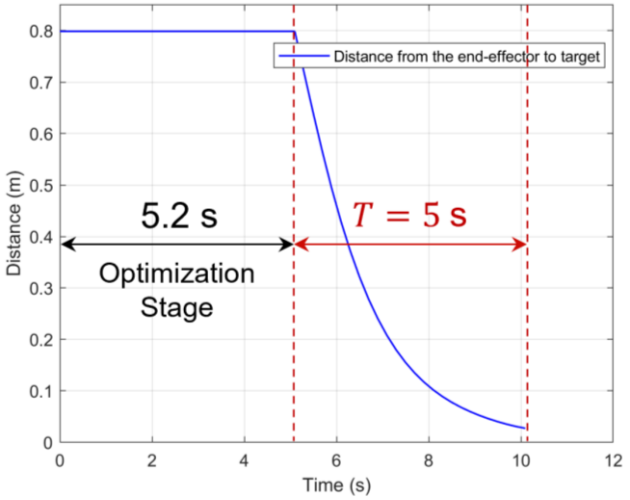

Fig. 4. Distance from end-effector to the target point over time of direct optimization.

The optimized control sequence and state response are depicted in Fig. 7 and Fig. 8 with those of MPC, respectively.

B. Model Predictive Control

Small value of the receding horizon H will reduce the optimization computation but it may lead to failure in tasks, e.g., collision with the obstacles, while the large value of H will increase the computation load. $H = 10$ is found to be the best solution and the results of this case are given in this section from in Fig. 5, 6, 7, 8. The results for other values, i.e., $H = 5,15,20$ can be found in the repo.

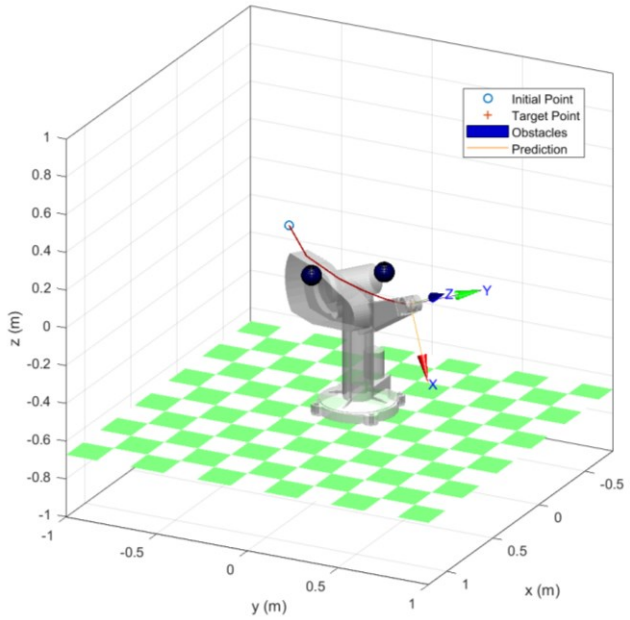

Fig. 5. Position trajectory of the end-effector with MPC.

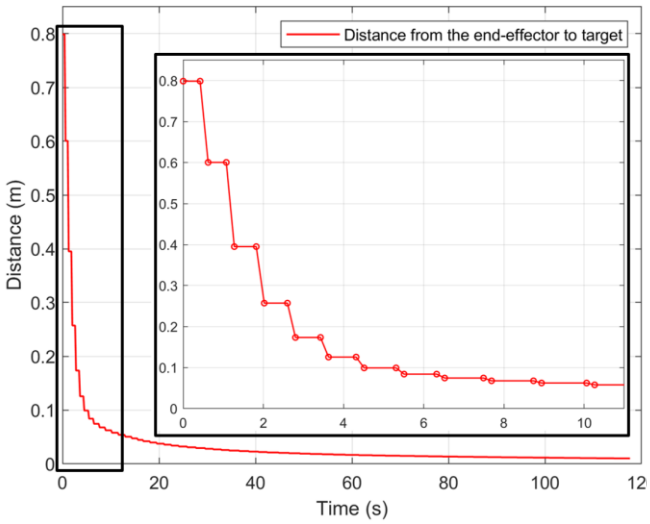

Fig. 6. Distance from end-effector to the target point over time of MPC.

C. Results Comparison of two methods

Fig. 7 and Fig. 8 show the joint angles (state) and angular velocities (control) of two methods.

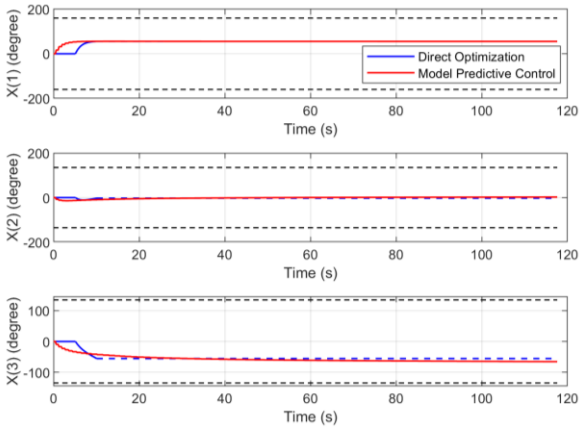

Fig. 7. Comparison between direct optimization and MPC in terms of the joint angle (state) response.

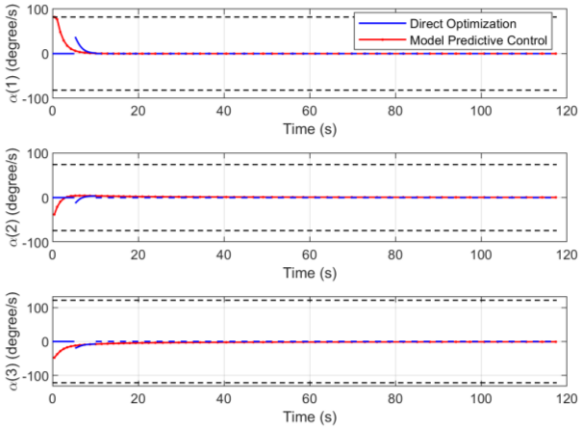

Fig. 8. Comparison between direct optimization and MPC in terms of the joint angular velocity (control input).

As shown in Fig. 9, the time cost for optimization in each iteration of MPC is much shorter than that in the direct optimization, which gives MPC faster response. But due to the value setting of the threshold ϵ, MPC takes much longer to terminate. However, as shown in Fig. 10, MPC reaches the target point closer than direct optimization after around 30 s in all three dimensions.

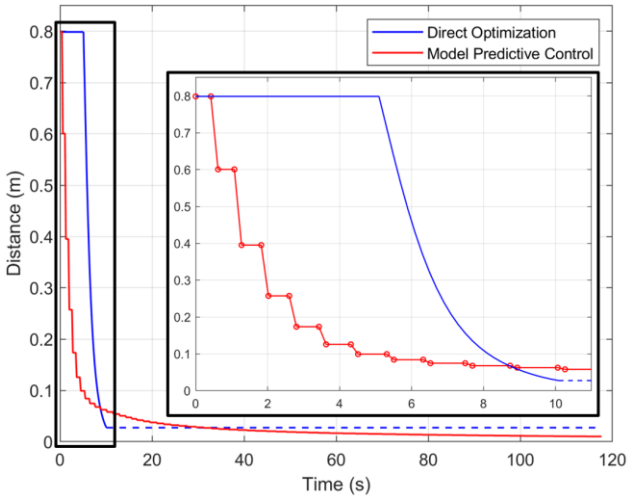

Fig. 9. Comparison between direct optimization and MPC in terms of the distance from the end-effector to the target point.

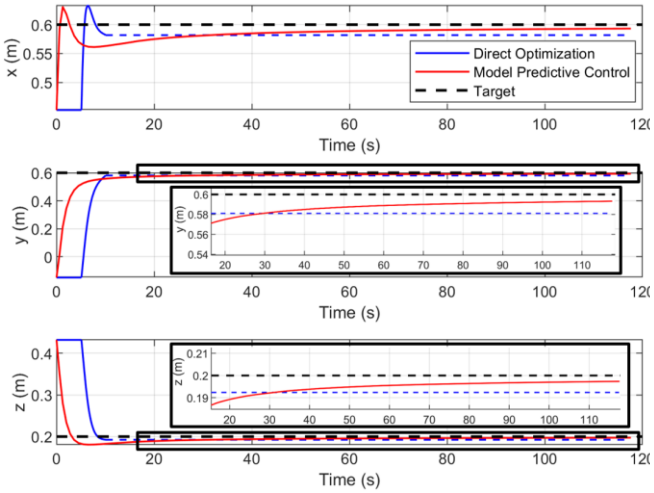

Fig. 10. Comparison between direct optimization and MPC in terms of the position response of the end-effector.

From Fig. 11, it can be noticed that the trajectory from direct optimization is smoother than that from MPC. Other constraints regarding the smoothness of the end-effector position trajectory should be added into the optimization formulation.

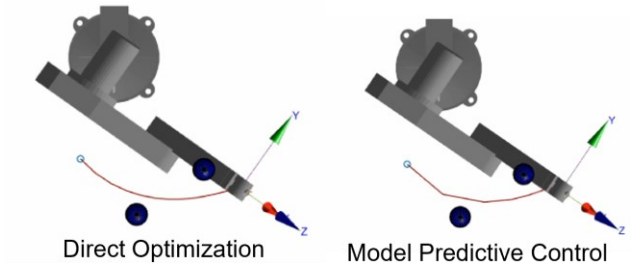

Fig. 11. Comparison between direct optimization and MPC in terms of the position trajectory of the end-effector.

VI. Conclusion

In this project, the point-to-point kinematic control of the 3-joint robotic arm is formulated within the optimization framework and addressed using direct optimization method, which is also treated as an optimal control problem and solved using MPC. Both methods implemented were able to successfully find the solution. Comparative analysis in terms of time, cost, and solution quality of both methods were also done.

References

[1] Corke, P. (2011). *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer.

[2] Lynch, K. M. (2017). *Modern Robotics*. Cambridge University Press.

[3] Andersson, J. A., Gillis, J., Horn, G., Rawlings, J. B., & Diehl, M. (2019). CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, *11*, 1-36.

[4] Borrelli, F., Bemporad, A., & Morari, M. (2017). *Predictive control for linear and hybrid systems*. Cambridge University Press.