Model Predictive Control for Collision-Free Manipulation in Constrained Environments

Tyler Toner

I. Introduction

Robotic manipulators are used widely in industry, particularly for repetitive operations. As manufacturing moves towards smaller batches, industrial robots must be rapidly reprogrammed for new tasks. If the robot, its cell, and its goals are well-modeled in a CAD environment, programming can be done entirely offline. On the contrary, when the current manufacturing environment is reconfigured or dynamic, much of the programming labor must be repeated. This motivates the need for automated motion planning even for industrial robots that are typically programmed manually.

A key capability that scales across many tasks is the ability to move the robot's end effector to a desired pose without collision with the environment. When an environment model is not available, vision sensor feedback can be used to gather important information, such as the location of task-specific objects and nearby obstacles to avoid. This information can be captured before task execution to build a temporary environment model, or processed continuously inside a feedback loop. See Figure 1 for an example. The Kuka LBR iiwa robot is attempting to reach a goal inside a CNC machine for a pick or place operation. For vision feedback, a Toyota HSR robot provides sensor feedback while the iiwa performs the task.

Visual servo (VS) is a classic approach for controlling robot motion with vision feedback that is flexible with respect to sensor type and placement. Position-based visual servo (PBVS) is more common with stereo cameras or RGB-D sensors that directly provide estimates of a goal pose (position and orientation) [1]. In addition to joint motion for end-effector control, obstacle avoidance of the manipulator arm itself must be considered. The artificial potential field method defines repulsive virtual forces as gradients of potential functions generated by obstacles of various shapes [2]. In [3], an explicit velocity control to push arm points away from obstacles is achieved simultaneously with end-effector control using carefully-selected Jacobian nullspace projections. A formal framework for combining multiple controllers in both task-space and joint-space in a geometrically-consistent way is presented in [4]. More recently, obstacles have been represented directly as point clouds or depth maps [5] in the avoidance control loop, enabled by fast, GPU-based algorithms for distance computations [6]. To work well with other task objectives and constraints, the avoidance controllers must be carefully tuned, which is typically done manually in an offline manner.

The author is in the Department of Mechanical Engineering at the University of Michigan, Ann Arbor, MI: twtoner@umich.edu.

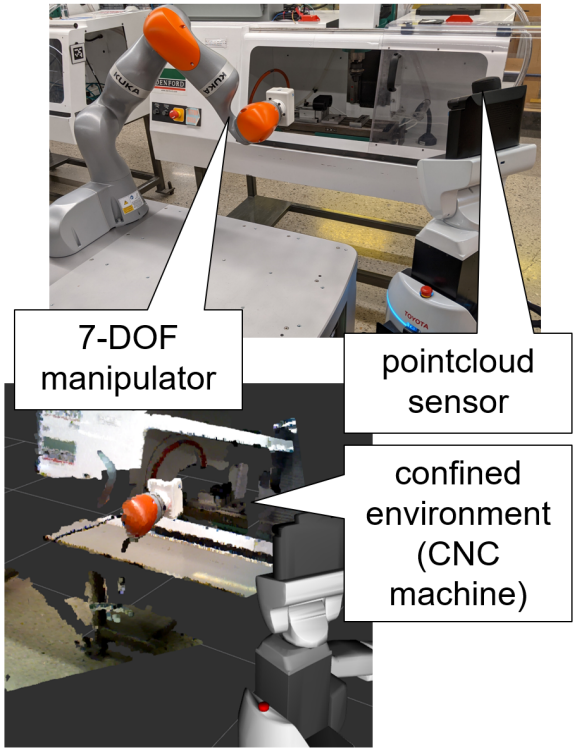

Fig. 1. Example problem: Manipulator goal reaching in a confined environment (CNC machine) given only sensor data (pointcloud) for goal pose estimation and obstacle location measurement.

The author's prior work [7] sought to address some of these limitations through iterative feedback tuning (IFT) on a nonlinear feedback controller. However, such an approach can be brittle in practice, as it relies on real-time availability of Jacobian pseudoinverses, which can be numerically unstable. Moreover, hard constraints on joint position and velocity cannot be guaranteed. These challenges motivate the formulation of the confined manipulator reaching problem as a model predictive control (MPC) problem.

II. Preliminaries and Problem Setup

A. Robot kinematics

For any configuration q in the joint space $Q \subset \mathbb{R}^n$, the resulting pose (p_j, R_j) of the jth frame is given by the forward kinematic map $\mathcal{F}_j : Q \to \mathbb{R}^3 \times SO(3)$, where p_j is a translation vector and R_j is a rotation matrix from the rotation group $SO(3)$. The map $\mathcal{F}_j^p : q \to \mathbb{R}^3$ gives only the translation. Hereafter we drop the subscript j for

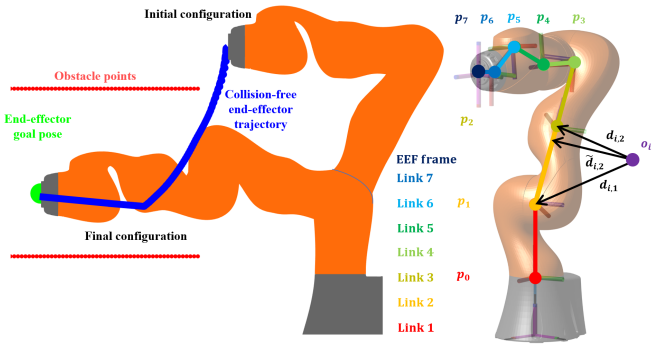

Fig. 2. Left: Problem schematic of obtaining a safe trajectory from an initial configuration to some final configuration that reaches the goal pose and avoids all obstacles with the end-effector and arm. Right: Examples of collision metrics (3) and (2): $d_{i,1}$, $d_{i,2}$ and $\tilde{d}_{i,2}$ on obstacle point o_i. p_0 through p_6 represent the positions of the seven joints on the Kuka LBR iiwa, and p_7 is its end-effector (EEF) frame position. Task steps are abstracted as end-effector goal poses, so end-of-arm tooling is neglected.

clarity. Note that the pose $(p, R) \in SE(3)$ where $SE(3)$ is the Special Euclidean group. Often, this is represented by a homoegeneous transformation matrix (with slight abuse of set notation):

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \in SE(3) \subset \mathbb{R}^{4\times 4} \tag{1}$$

Note that we will use T and (p, R) notation interchangeably as appropriate.

Differentiating $\mathcal{F}$ with respect to q gives the Jacobian, $J(q) \in \mathcal{R}^{6\times n}$, which maps a vector of joint velocities $\dot{q}$ to the resulting linear and angular velocities, $\mathbb{R}^6 \ni V = \begin{bmatrix} v^\top \omega^\top \end{bmatrix}^\top = J(q)\dot{q}$, of the end-effector or other frame. Respectively $\mathbb{R}^3 \ni v = J^v(q)\dot{q}$ where $J^v(q)$ is the Jacobian of $\mathcal{F}^p(q)$.

B. Obstacle distance metrics

For avoidance, it is necessary to track the distances between obstacles and points on the robot. In [5], several control points were manually defined on the robot arm for obstacle distance checking, while in [6], a mesh of the robot's geometry was used to check hundreds or thousands of points. For computational efficiency, we introduce two simplified metrics for approximating a robot's proximity to collision. First, let

$$d_{ij} = \begin{cases} p_j - o_i & h_{ij} \geq 1 \\ p_{j-1} - o_i & h_{ij} \leq 0 \\ \underbrace{\left(\frac{p_{j,j-1}p_{j,j-1}^\top}{p_{j,j-1}^\top p_{j,j-1}} - I_3\right)}_{P_j}(o_i - p_{j-1}) & 0 < h_{ij} < 1 \end{cases} \tag{2}$$

where $p_{j,j-1} = p_j - p_{j-1}$ is the vector pointing from joint $j-1$ to joint j and $h_{ij} = (o_i - p_{j-1}) \cdot p_{j,j-1} / (p_{j,j-1} \cdot p_{j,j-1})$ is the coefficient that scales the vector $p_{j,j-1}$ to the projection of obstacle $o_i \in \mathbb{R}^3$ onto $p_{j,j-1}$. If $h_{ij} \notin [0, 1]$, then the projection falls on a point physically outside the link, so d_{ij} is set to point to the nearest joint. P_j is a matrix to simplify the projection operation that will be used later.

We also introduce a simpler distance metric; let

$$\tilde{d}_{ij} = p_j - o_i \tag{3}$$

be the vector pointing from obstacle point o_i to the position p_j of joint j. Examples of d_{ij} and $\tilde{d}_{ij}$ are shown in Figure 2. These metrics were developed independently in the author's prior work [7], but (2) is similar to the distance between line-swept spheres introduced in [8] and exploited for obstacle avoidance in MPC in [9].

Generally $||d_{ij}||$ is a more accurate representation of obstacle proximity than $||\tilde{d}_{ij}||$, but the latter is used to simplify analysis. We define the relation

$$\min_{i,j} ||d_{ij}|| < r_j \tag{4}$$

to indicate collision, where r_j is a prespecified safety radius for link j, which can be used to account for link geometry as well as additional tolerance.

C. Problem setup

To analyze the problem of collision-free goal reaching in the presence of obstacles as an MPC problem, we make a few simplifying assumptions.

Assumption 1: There exists a low-level controller to command appropriate joint torques given a joint velocity command $\dot{q}_{cmd}$. Moreover, this controller is sufficiently high bandwidth that its transient response can be neglected for commands within the joint velocity limits. Thus, we assume a **kinematic model** at the MPC level:

$$\begin{aligned} q_{k+1} &= q_k + \Delta_t u_k \\ &= Aq_k + Bu_k \end{aligned} \tag{5}$$

for discrete time steps $k \in \mathbb{Z}^+$, control input $u_k = \dot{q}_{cmd}$, and states being joint positions q_k. Note that the system matrices are $A = I_n$ and $B = \Delta_t I_n$ so the system is linear and clearly controllable.

Assumption 2: For the problems under consideration, there exists an input trajectory $\{u_0, ..., u_p - 1\}$ that generates a state trajectory $\{q_1, ..., q_p\}$ from the initial condition q_0 and some finite $p > 0$ such that either $q_p = q_g$ for a joint goal q_g or $\mathcal{F}(q_p) = T_g$ for a pose goal T_g. In other words, we assume **feasibility**.

Assumption 3: The goal (p_g, R_g) and the obstacle set $O = \{o_i\}_{i=1}^{n_o}$ are static, or are changing sufficiently slowly that they can be **assumed constant over the MPC horizon**.

Using these assumptions, we will propose two approaches tracking pose goals in joint space and tracking pose goals directly. The goal of this project is to develop an MPC controller that can reach pose goals in the presence of obstacles using only pointcloud data as sensor feedback. First we analyze the obstacle-free case of both formulations, then discuss how the formulations can be adapted to include obstacles. In all cases, built-in MATLAB toolboxes are used for optimization. The MPC time step is $\Delta_t = 0.1s$ and the simulation time step is $\Delta_t = 0.01s$.

III. Joint Goal Tracking without Obstacles

To study the problem of goal tracking with a manipulator, we first analyze the case of tracking a joint configuration in an obstacle-free workspace.

A. Inverse Kinematics

If a goal is given in terms of an end-effector pose (p_g, R_g), a joint configuration goal q_g may exist to realize (p_g, R_g); finding q_g is known as the *inverse kinematics (IK) problem*. Closed-form solutions exist for robots with exactly six joints, and even some with more than six [10]. However, the formulation of IK as an optimization problem solved with numerical methods is more general and has been thoroughly studied [11]. Since IK is outside the scope of this report, we cite the existing solvers used by `inverseKinematics` in the MATLAB Robotics System Toolbox [12].

B. Joint Tracking MPC Problem

To reach some goal q_g, we use the LQ-MPC reference tracking Formulation 1 discussed in Module 5 of the course. This formulation was chosen for its simplicity and its compactness. In particular, as obstacles are added to the constraints in later sections, small state spaces keep the solution speed reasonable, as opposed to the augmented states of Formulations 2-4 in Module 5. We formulate the MPC problem as:

$$\begin{aligned} \min_{u_0,\dots,u_{N-1}} \mathcal{J} &= \sum_{k=1}^{N} \|q_k - q_g\|^2_{Q_q} + \|u_k\|^2_R + \|q_N - q_g\|^2_{P_q} \\ \text{s.t.} \quad q_{k+1} &= Aq_k + Bu_k \\ q_{min} &\le q_k \le q_{max} \\ u_{min} &\le u_k \le u_{max} \\ q_0 &\quad \text{given} \end{aligned} \tag{6}$$

where q_{max}, q_{min}, u_{max}, and u_{min} are constraints on joint position and velocity. As shown in class, this can be condensed and shown to be a strictly convex quadratic program (QP). Importantly, recall that we can use the discrete time state transition formula to define a lifted system form:

$$\underbrace{\begin{bmatrix} q_1 \\ \vdots \\ q_N \end{bmatrix}}_{\mathcal{Q}} = \underbrace{\begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ A^{N-1}B & A^{N-2}B & \dots & B \end{bmatrix}}_{S} \underbrace{\begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{U} + \underbrace{\begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}}_{M} q_0 \tag{7}$$

which gives a straightforward gradient of the state trajectory with respect to the control trajectory:

$$\frac{\partial \mathcal{Q}}{\partial U} = S. \tag{8}$$

Next we summarize some experimental results from applying MPC to the manipulator tracking problem in joint space.

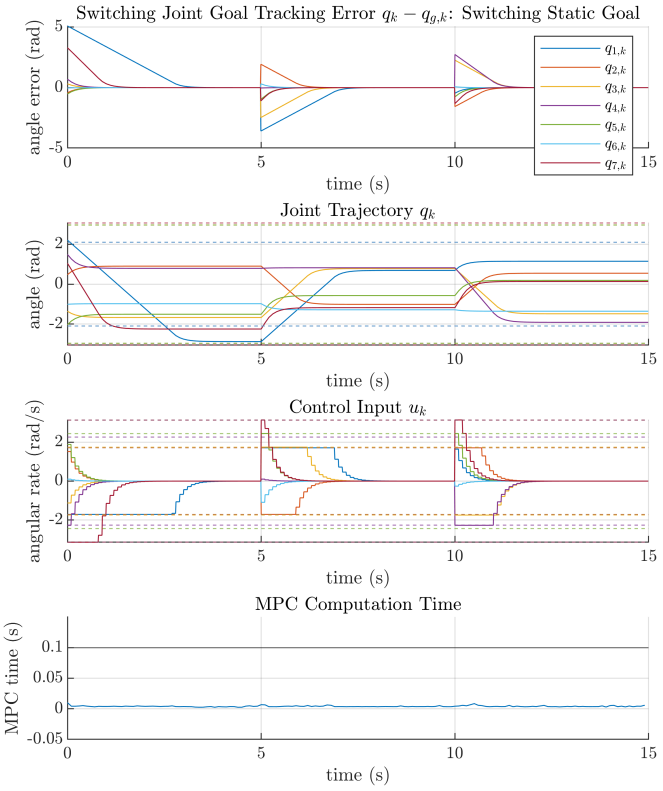

Fig. 3. Goal joint configuration tracking without obstacles.

C. Performance

The following parameters were chosen to balance performance with computation time:

- $R = I_7$
- $Q_q = 10 \cdot I_7$
- $P_q = Q_q$
- $N = 10$.

P was chosen to equal Q since stability with the kinematic system (5) is not a concern in the case without obstacles. The ratio between Q and R was chosen so that the choice of N was evident. In practice, it seemed that Q could be set to any large multiple of R, and performance would only improve, even with small a small horizon N. The effect of N on tracking and computation time is discussed later. The condensed MPC problem was solved using `quadprog` from MATLAB's Optimization Toolbox [13], with parameters:

$$\begin{aligned} H &= 2S^T \bar{Q} S + R \\ f &= 2S^T \bar{Q} M (q_0 - q_g) \end{aligned} \tag{9}$$

where q_0 represents the initial condition of the MPC problem, or in other words the value of q_k when MPC is called at time k; likewise, the goal value of q_g was allowed to change at each iteration.

1) Switching static goals: To show convergence to general static goals, an initial configuration was chosen randomly from $\mathcal{Q}$, as well as three goal configurations spaced equally in time. The tracking performance on switching goals is shown in Figure 3. Note that errors converge rapidly in joint

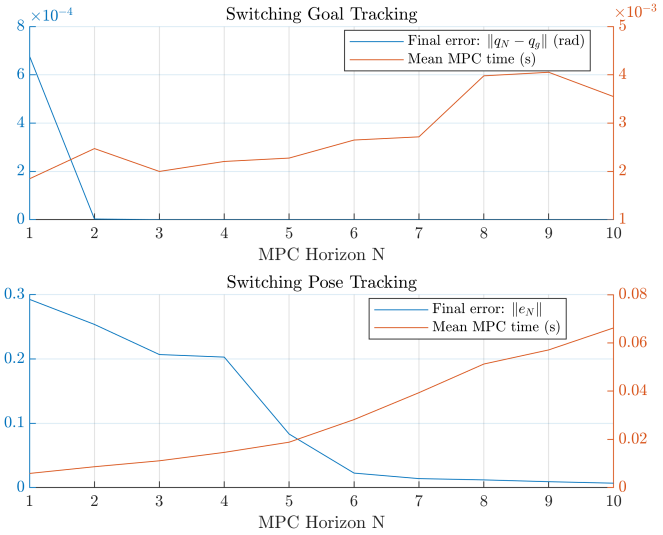

Fig. 4. Error convergence and MPC time for a range of horizon values N, for both joint tracking and direct pose tracking.

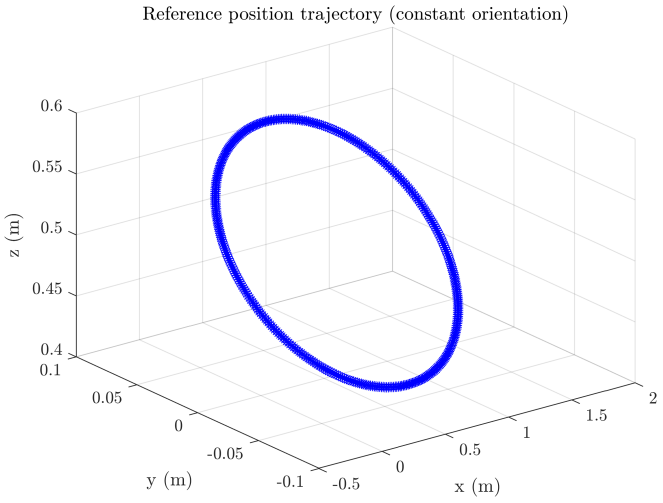

Fig. 5. Reference position trajectory to be tracked.

space, but neither joint nor control constraints are exceeded. Moreover, the MPC computation time is trivial, well below $\Delta_t = 0.1s$.

To understand the effect of the MPC horizon on convergence and MPC computation time, a range of N values from 1 to 10 were simulated, shown in figure 4 (top). Over the range of horizon values, the mean final error drops dramatically without a significant rise in computation time. However, it is clear that even at $N = 1$, the performance is perfectly acceptable for static or switching goals.

2) Tracking a reference trajectory with IK: Next, a reference end-effector pose trajectory was generated for a constant orientation R_g but time-varying $p_{g,k}$, which forms an ellipse in the workspace (Figure 5) that is traversed with frequency $1/4Hz$. Extending the MPC formulation in the case of a static switching goal, we now introduce the step of solving an IK problem at each step of the MPC. Using the current configuration q_k as an initial guess, an appropriate

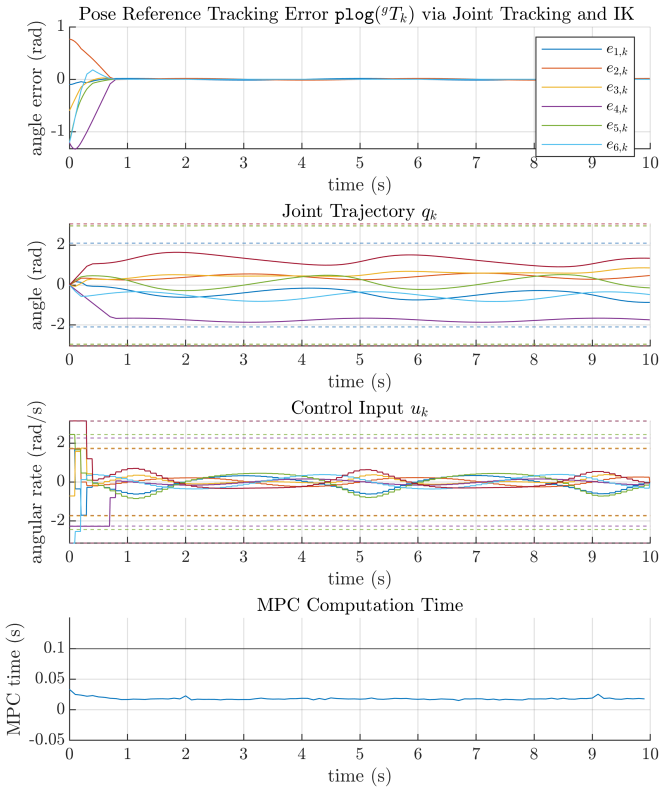

Fig. 6. Joint tracking a pose using in-loop inverse kinematics.

goal $q_{g,k}$ is generated for the time-varying ^gT_k.

The resulting tracking error is represented by $e_k = \texttt{plog}(^gT_k)$ (a metric introduced in the next section, equation 11), whose trajectory is shown in Figure 6. The robot is initialized at $q_0 = 0$ but converges rapidly to the trajectory within 1 second, and tracks accurately over the slowly-varying reference trajectory without violating joint or control constraints. Also note that, even with the IK solver called at each MPC iteration, the computational time is only around $0.01s$, well below the MPC discrete time of $\Delta_t = 0.1s$.

IV. Pose Goal Tracking without Obstacles

Often, robot goals can be better described as poses (p_g, R_g) than explicit joint configurations q_g. Even when IK solvers are available, their solutions non-unique for robots with $n > 6$; furthermore, in the case of moving goals, the IK solver must be called in each instance of the control loop. In this section, we study how pose goals can be tracked directly.

A. *Pose Tracking in* SE(3)

To formulate an MPC problem similar to (6), we must define a pose error metric e_k to penalize in the cost function at each time step; we will refer to this term as $\mathcal{J}_k^e$ to indicate the tracking cost. If only position goals p_g are of importance, we can define $e_k = p_k - p_g$ with tracking cost $\mathcal{J}_k^e = e_k^T Q_e e_k$, which has a straightforward gradient (recall $p_k = \mathcal{F}^p(q_k)$):

$$\begin{aligned} \frac{\partial \mathcal{J}_k^e}{\partial q} &= \frac{\partial \mathcal{F}^p(q_k)}{\partial q}^T \frac{\mathcal{J}_k^e}{\partial e} \\ &= 2J^p(q_k)^T Q_e(\mathcal{F}^p(q_k) - p_g) \end{aligned} \tag{10}$$

However, tracking goals in $SE(3)$ requires a more careful treatment. Instead of driving a vector difference to 0, the objective is instead to drive the transform between the goal and the current pose, ${}^gT_k = (T_g)^{-1}T_k$, to I_4.

A recent technical paper details a number of approaches for optimization in $SE(3)$, including vector representations of $SE(3)$ elements T that have well-defined gradients [14]. One approach is the *pseudo-log* (abbrev. $\texttt{plog}$), which maps matrices T, defined by submatrices R and p as in (1), to vectors in $\mathbb{R}^6$:

$$\texttt{plog}(T) = \begin{bmatrix} p \\ \log(R)^\vee \end{bmatrix} \tag{11}$$

where $\log(R)$ is a skew-symmetric matrix and the notation $\cdot^\vee$ rearranges the nonzero elements of a skew-symmetric matrix into a vector in $\mathbb{R}^3$, and can be interpreted as the vector of total angular rotation, which goes to 0 as R goes to I_3. Note that if we define our error metric as $e_k = \texttt{plog}({}^gT_k)$, then we have a vector that, as $e_k \to 0$, ${}^gT_k \to I_4$.

Furthermore, the gradient of $\texttt{plog}(T)$ is given by:

$$\frac{\partial \texttt{plog}(T)}{\partial T} = \begin{bmatrix} 0_{3\times 9} & I_3 \\ \frac{\partial \log(R)}{\partial R} & 0_{3\times 3} \end{bmatrix} \tag{12}$$

where the gradient of $\log(R)$ is defined in [14]. Finally, for $e_k = \texttt{plog}({}^gT_k)$ and $\mathcal{J}_k^e = e_k^T Q_e e_k$, one can compute the gradient of the cost as:

$$\begin{aligned} \frac{\partial \mathcal{J}_k^e}{\partial q} &= \left(\frac{\partial \texttt{plog}({}^gT_k)}{\partial q}\right)^T \frac{\mathcal{J}_k^e}{\partial e} \\ &= \left(\frac{\partial \texttt{plog}({}^gT_k)}{\partial {}^gT_k}\frac{\partial {}^gT_k}{\partial q}\right)^T \cdot 2Q_e \texttt{plog}({}^gT_k) \end{aligned} \tag{13}$$

where we have closed form expressions for each term except $\frac{\partial {}^gT_k}{\partial q}$. This term is similar to $J(q)$, since it is a Jacobian of the forward kinematic map $\mathcal{F}(q)$, except for the premultiplication by T_g^{-1}. In practice, it was obtained through symbolic differentiation using the MATLAB Symbolic Math toolbox [15], and converted to a function using the `matlabFunction` command. Finally, the overall cost gradient with respect to U can be computed by stacking values of $\frac{\partial \mathcal{J}_k^e}{\partial q}$ to form $\frac{\partial \mathcal{J}^e}{\partial \mathcal{Q}}$, and then:

$$\begin{aligned} \frac{\partial \mathcal{J}^e}{\partial U} &= \left(\frac{\partial \mathcal{Q}}{\partial U}\right)^T \frac{\mathcal{J}^e}{\partial Q}. \\ &= S^T \frac{\mathcal{J}^e}{\partial Q}. \end{aligned} \tag{14}$$

B. Pose Tracking MPC Problem

To track goals directly in $SE(3)$, we define the following MPC problem.

$$\begin{aligned} \min_{u_0,\ldots,u_{N-1}} \mathcal{J} &= \sum_{k=1}^{N} \|e_k\|_{Q_e} + \|u_k\|_R + \|e_k\|_{P_x} \\ \text{s.t.} \quad q_{k+1} &= q_k + \Delta_t u_k \\ e_k &= \texttt{plog}((T_g)^{-1}T_k) \\ T_k &= \mathcal{F}(q_k) \\ u_k &\in \mathcal{Q} \\ q_k &\in \mathcal{U} \end{aligned} \tag{15}$$

Although the cost function is convex in e, it is not convex in q or u, nor is there a linear relationship between $E = \begin{bmatrix} e_1^T & \ldots & e_N^T \end{bmatrix}^T$ and U as in the LQ-MPC formulation. Moreover, the first two constraints are clearly nonlinear; thus this is a nonlinear MPC (NMPC) problem.

If we consider position only and define $X = \begin{bmatrix} p_1^T & \ldots & p_N^T \end{bmatrix}^T$, $X_g = \begin{bmatrix} x_g^T & \ldots & x_g^T \end{bmatrix}^T$, $\bar{J}(\mathcal{Q}) = \texttt{diag}(J^v(q_1), \ldots, J^v(q_N))$, and $\bar{\mathcal{F}}(\mathcal{Q}) = \begin{bmatrix} \mathcal{F}^p(q_1)^T & \ldots & \mathcal{F}^p(q_N)^T) \end{bmatrix}^T$, the cost function can be condensed to the following:

$$\begin{aligned} \mathcal{J} &= (X - X_g)^T \bar{Q}_e (X - X_g) + U^T \bar{R} U \\ &= (\bar{\mathcal{F}}(\mathcal{Q}) - X_g)^T \bar{Q}_e (\bar{\mathcal{F}}(\mathcal{Q}) - X_g) + U^T \bar{R} U \\ &= (\bar{\mathcal{F}}(SU + Mq_0) - X_g)^T \bar{Q}_e (\bar{\mathcal{F}}(SU + Mq_0) - X_g) \\ &\qquad + U^T \bar{R} U \end{aligned} \tag{16}$$

The gradient (dropping constants) becomes:

$$\frac{\partial \mathcal{J}}{\partial U} = 2S\bar{J}(SU + Mq_0)^T Q(\bar{F}(SU + Mq_0) - X_g) + 2\bar{R}U. \tag{17}$$

If there are no control constraints ($\bar{R} = 0$), this gradient is equal to 0 only for $\bar{F}(SU + Mq_0) = X_g$ (since the transposed Jacobian is always at least full column rank); thus it clearly has a unique solution in this case. However, since $\mathcal{F}^p(q)$ has no analytical inverse for $n > 3$ (for position goals only), there is actually a manifold of optimal solutions with dimension $n-3$. This represents null-space motion, or "self-motion", where the joints can move without affecting the end-effector position. This indicates that the cost function may be *convex* but not *strictly convex*. Regardless, where $\bar{R} \neq 0$ and general $SE(3)$ goals are defined, the notion of convexity is less clear. Thus (15) was treated as a general NMPC.

C. Performance

The implementation of (15) was similar to that of (6), except that the nonlinear solver `fmincon` was used. Furthermore, each MPC iteration after the first one was warm-started using the previous solution U^* shifted by n steps and padded with the final value of U^*. In each of the following cases, the first step was completed with the Interior-Point solver for robustness to infeasible initial guesses, then subsequently solved with SQP for faster solutions. The following parameters were chosen:

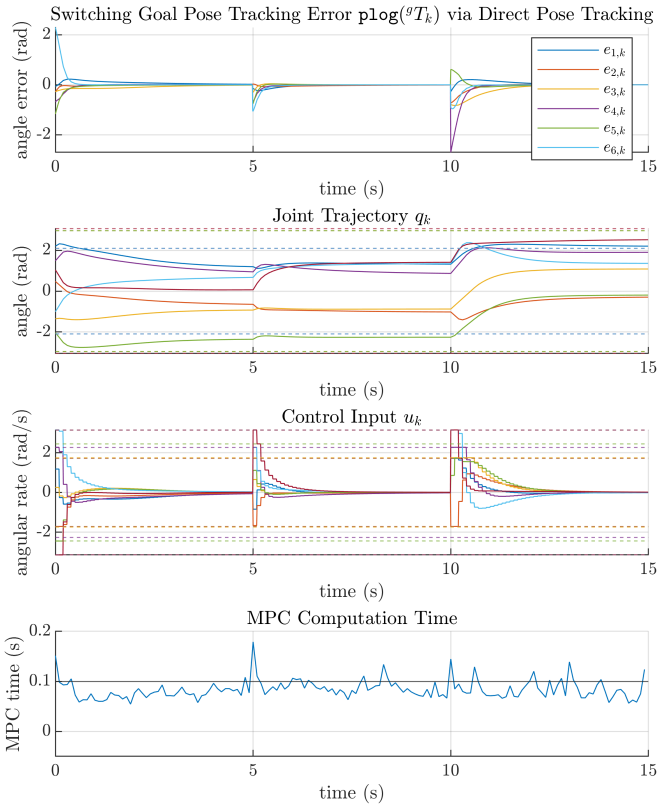

Fig. 7. Tracking randomly chosen static goals that switch at regular intervals.

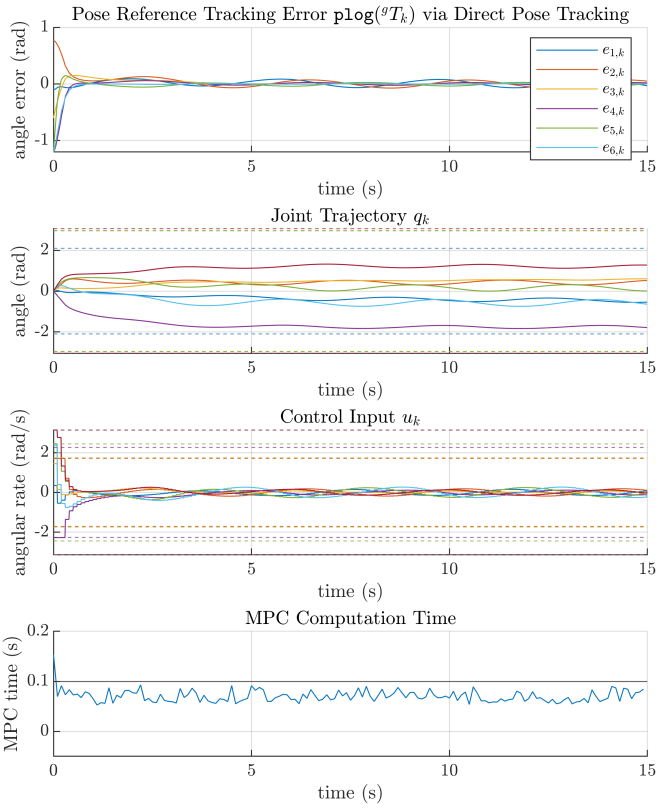

Fig. 8. Following a reference trajectory.

- $R = I_7$
- $Q_e = 30 \cdot I_6$
- $P_e = Q_e$
- $N = 10$.

where the only change from (6) is the dimension of Q_e, P_e and the multiplier 30, which appeared to lead to similar convergence rates with the same value of N as in the joint-tracking case (6). Next, the same two cases of switching static goals and a slowly-varying reference trajectory are repeated from section III.

1) Switching static goals: An random initial joint configuration and three random goal poses were generated and spaced equally in time. The tracking results are shown in Figure 7. Note the similar convergence properties in tracking as the joint goal switching case in Figure 3. One difference is that in the previous case, the joint trajectories are stationary after goal convergence (this is expected since the goals are in joint space); however, here, the joint trajectories continue to evolve gradually after the pose error has vanished. Moreover, notice the significantly higher MPC computation time. This is initially counterintuitive considering there is no IK solver being called in this case. This increased time is explained by the fact that the problem is now NMPC rather than LQ-MPC, thus the faster quadratic programming solvers cannot be exploited.

When profiling the MATLAB script, it appears that the most costly operations are those to compute the `plog` (11) and the cost gradient (14). This remains the case even when the MATLAB Coder is used to write these functions to C-code.

Similar to the joint tracking case, the effect of the MPC horizon N on computation time and tracking performance was studied: see Figure 4 (bottom). We see that, unlike the joint tracking case, the pose tracking case benefits greatly from longer horizons, but this more significantly affects the computation time, showing roughly $O(N)$ growth. The largest practical horizon is around $N = 10$, as larger horizons bring the mean MPC computation time above the MPC discrete time $\Delta_t = 0.1s$.

2) Tracking a reference trajectory: To study reference trajectory following for the pose tracking formulation, the same reference as in section III was tracked: Figure 5. The tracking results for $N = 10$ are shown in Figure 8. From the figure it is clear that both the tracking error and MPC computation time are significantly larger than in the joint tracking case.

V. Cost Functions for Obstacle Avoidance

Note: Some of this section is reproduced from the author's prior research [7], as it is currently unpublished and thus cannot be referenced.

When obstacles are introduced into the workspace, both the joint tracking and pose tracking problems become NMPC problems through the introduction of nonlinear constraints. However, the incorporation of obstacle avoidance into the cost function itself reduces the likelihood of the constraints

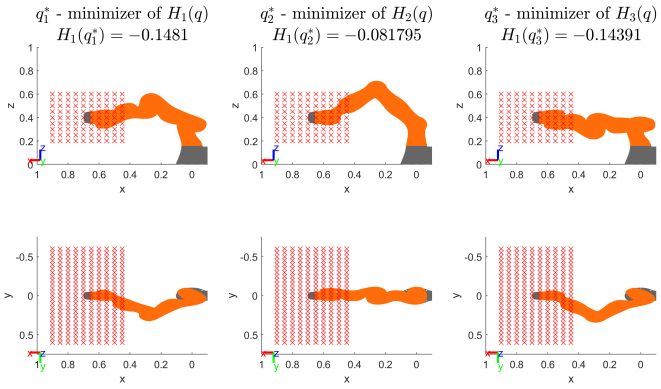

Fig. 9. Comparison of global minimizers of $H_1(q)$, $H_2(q)$, and $H_3(q)$ for a fixed end-effector pose and fixed obstacle locations. Note that the obstacle field (red markers) is a box that is open on the side facing the robot, and in all cases, collision is avoided. Top row: side views; bottom row: top views. $H_1(\cdot)$ is used to compare the performance of the minimizers q_1^*, q_2^*, and q_3^* (lower is better). q_3^* has nearly the same cost as q_1^* while q_2^* has a much higher cost, indicating $H_3(q)$ is a better proxy for $H_1(q)$ than $H_2(q)$ is.

being active, speeding up computation time and leading to safer overall trajectories.

For avoidance, we want to ensure that the minimum distance between the robot and all obstacles does not fall below some safety radius r, defined in (4). We can express this objective as a cost function,

$$H_1(q) = -\min_{i,j} ||d_{ij}(q)||, \tag{18}$$

which is minimized by maximizing the minimum obstacle distance. The minimum in (18) can be approximated by a smooth, differentiable function like α-softmax [16] to obtain $\nabla_q H_1(q)$, which is made more tractable by replacing d_{ij} (2) with $\tilde{d}_{ij}$ (3). However, in practice, we found that the existence of several local minima in $H_1(q)$ leads to a performance that is heavily dependent on the obstacle configuration and is generally not acceptable.

Instead, we can consider an alternative cost function with acceptable local minima that serves as a proxy for $H_1(q)$. For example, we could attempt to maximize the *sum* of all obstacle distances:

$$H_2(q) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n_o} ||\tilde{d}_{ij}(q)||^2, \tag{19}$$

where the gradient $\nabla_q H(q) = -\sum_{j=1}^{n}\sum_{i=1}^{n_o}(J_j^v(q))^T\tilde{d}_{ij}(q)$ is easily computed. While this formulation is intuitively appealing and has fewer local minima, even the global minimum does not yield acceptable performance. To lessen the contribution of faraway obstacles, we replace $||\tilde{d}_{ij}(q)||^2$ with $\exp(-b||\tilde{d}_{ij}(q)||^2)$ for some $b > 0$ that controls the rate of exponential decay, yielding:

$$H_3(q) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n_o}\exp(-b||\tilde{d}_{ij}(q)||^2). \tag{20}$$

To visualize the difference in performance between $H_2(q)$ and $H_3(q)$, see Figure 9; the minimizer of $H_3(q)$ is very similar to that of $H_1(q)$, while $H_2(q)$ is significantly worse. For clarity, we will hereafter refer to $H_3(q)$ in (20) simply as $H(q)$.

VI. Joint Goal Tracking with Obstacles

A. Joint Tracking MPC Problem

We modify the joint tracking MPC problem (6) as follows:

$$\begin{aligned}\min_{u_0,\ldots,u_{N-1}} \mathcal{J} = &\sum_{k=1}^{N} \|q_k - q_g\|_{Q_q}^2 + \|u_k\|_R^2 + k_a H(q_k) \\ &+ \|q_N - q_g\|_{P_q}^2 \\ \text{s.t.}\quad & q_{k+1} = Aq_k + Bu_k \\ & q_{min} \le q_k \le q_{max} \\ & u_{min} \le u_k \le u_{max} \\ & r_j < \|d_{ij}(q_k)\| \\ & q_0 \quad \text{given}\end{aligned} \tag{21}$$

where $k_a > 0$ is the gain on the obstacle avoidance term. Clearly this is a general NMPC problem. The addition of $H(q_k)$ ensures $\mathcal{J}$ is no longer quadratic, and the additional hard constraint is clearly nonlinear (though it is differentiable).

B. Performance

For brevity, the results of this MPC implementation are omitted. Briefly stated, the problem was initially solved as a long-horizon optimal control problem (OCP) as a baseline. However, it was clear that even with long horizons, the goal configuration could not be reached from general initial conditions due to being trapped in local minima by the presence of obstacles. Better performance was achieved by the pose tracking formulation (discussed in the next section).

VII. Pose Goal Tracking with Obstacles

A. Pose Tracking MPC Problem

We modify the pose tracking MPC problem (15) as follows:

$$\begin{aligned}\min_{u_0,\ldots,u_{N-1}} \mathcal{J} = &\sum_{k=1}^{N} \|e_k\|_{Q_e}^2 + \|u_k\|_R^2 + k_a H(q_k) \\ &+ \|q_N - q_g\|_{P_q}^2 \\ \text{s.t.}\quad & q_{k+1} = Aq_k + Bu_k \\ & q_{min} \le q_k \le q_{max} \\ & u_{min} \le u_k \le u_{max} \\ & r_j < \|d_{ij}(q_k)\| \\ & q_0 \quad \text{given}\end{aligned} \tag{22}$$

which is similar to the modification made in (21).

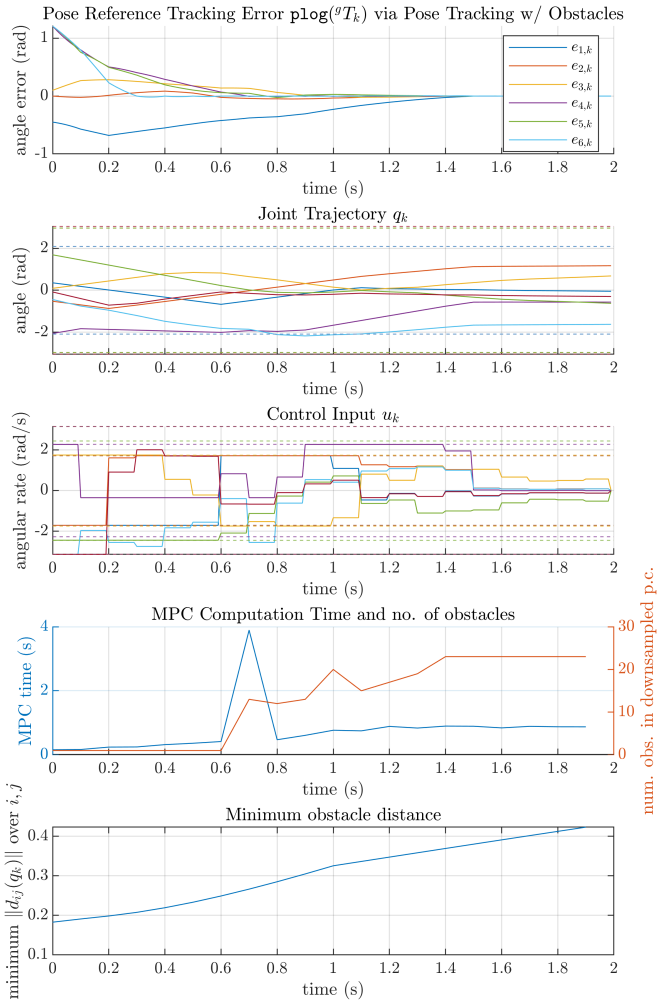

Fig. 10. NMPC performance on pose tracking for static goal in presence of obstacles.

B. Performance

The following parameters were chosen through trial and error:

- $R = I_7$
- $Q_e = 1 \times 10^7 \cdot I_6$
- $P_e = Q_e$
- $N = 3$.
- $k_a = 1000$.
- $b = 15$

The main challenge in tuning was selecting a balance between Q_e and k_a. The gain k_a must be large enough to avoid obstacles; however, this acts as a disturbance, so the magnitude of Q_e must be increased to reject it and maintain good convergence. The value of $b = 15$ was chosen so that the squared exponential in the cost function (20) decays to around 5% at a maximum distance of $0.2m$, limiting the influence of faraway obstacles. This is similar to the concept of [3], who term this type of metric a "sphere of influence". The modest MPC horizon of $N = 3$ was chosen because any larger values led to unreasonably long computation times. Still, even at $N = 3$, the MPC performs below real time.

To improve computational performance, the total observed obstacle set $O = \{o_i\}_{i=1}^{n_o}$ was uniformly downsampled to around 5% of its original value to O_{ds}^{glo} ("glo" for "global"), which was used in the obstacle avoidance term $H(q)$. This allows the gradient of $H(q)$ to guide the solution to an overall configuration that tends to move the arm away from obstacles. Additionally, at each MPC iteration, the original set O was downsampled to O_{ds}, which instead considered only obstacles closer than some threshold; we chose $1.5 \times r_j$ where r_j is the minimum safety distance for joint j. The idea is that during the MPC solution, the nonlinear constraints consider the local O_{ds} to ensure safety, while the cost function avoidance term $H(q)$ considers the sparser but more global O_{ds}^{glo} to guide the overall configuration away from obstacles.

The problem setup is shown in Figure 11, and the resulting trajectory is shown. The results are shown in Figure 10. Notice that even when the pose error converges around $1.4s$, the joints continue to move to maximize the overall obstacle distance (shown in the bottom plot).

Also notice that the MPC time far exceeds $\Delta_t = 0.1s$, and appears to be correlated to the number of obstacles in O_{ds}, the locally downsampled pointcloud. This makes sense, since the more obstacles observed, the more complex the cost function and constraint computations must be.

VIII. Conclusions

In this report, we described the problem of online obstacle avoidance for a manipulator reaching goals in confined environments, where collision information is only available in the form of a pointcloud data stream.

We first considered the case without obstacles for both joint tracking and end-effector pose tracking. We found that, despite the inclusion of an inverse kinematics solver at each MPC step, the LQ-MPC joint tracking controller was far more computationally efficient than the equivalent NMPC pose tracking controller.

However, the pose tracking controller proved to be far more versatile for reaching pose goals in the presence of obstacles. By carefully designing a new cost function for our NMPC that explicitly drives the overall robot arm away from obstacles, we minimize the frequency of hard constraint activation, leading to a smoother and safer solution. Furthermore, as shown in Figure 10, the controller continues to increase the minimum obstacle distance even after the pose error has converged and even when constraints are not active. Meanwhile, all controllers this report avoid the numerical issues present in traditional obstacle avoidance feedback controllers that require Jacobian inversion and careful null-space treatment.

The primary remaining challenge is further computational improvement of the final NMPC pose tracking controller with obstacles. The MPC computation time in simulation is far higher than the designed $0.1s$ interval. Gains could be made by explicit parallelization using GPUs, which were exploited in [6]. Additionally, it remains to implement this controller on the real robot, or a nonlinear model that

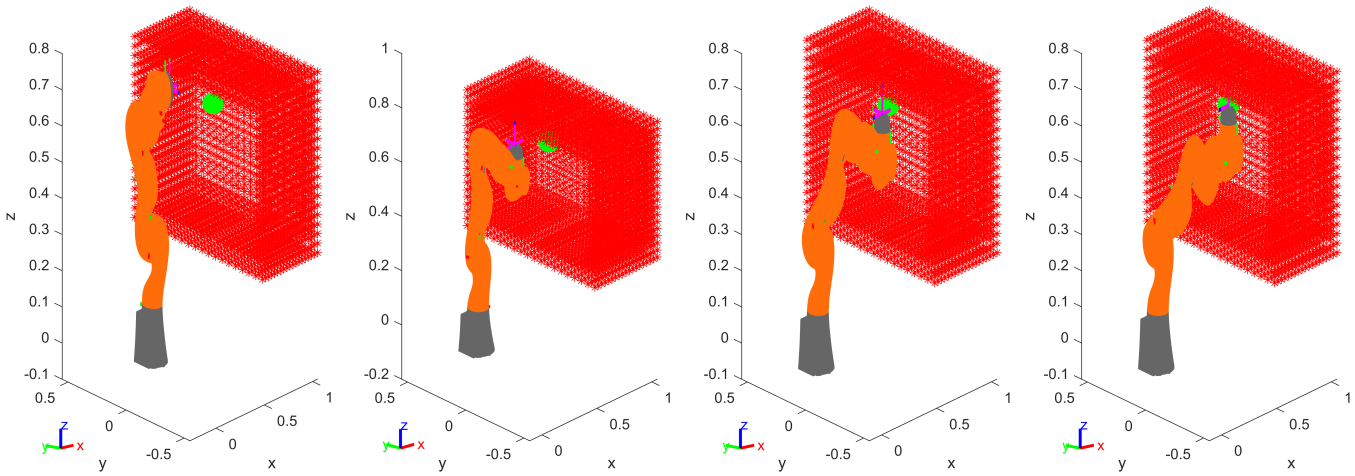

Fig. 11. Time flow from left to right. Safe reaching generated by the pose tracking NMPC. The red markers are points from environmental geometry and the green marker is a goal end-effector position. The modified MPC problem satisfies the minimum distance metric (2) for safety radius $r = 0.1$ m, and the robot reaches the goal pose without collision.

includes low-level controller dynamics. These goals will be the next steps in the development of this work.

ACKNOWLEDGMENT

This work was completed as a final project for the AEROSP 740: Model Predictive Control course at the University of Michigan.

REFERENCES

[1] F. Chaumette and S. Hutchinson, "Visual Servo Control Part I: Basic Approaches," *IEEE Robotics and Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[2] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 500–505, 1985.

[3] A. A. Maciejewski and C. A. Klein, "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 109–117, 1985.

[4] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," *arXiv*, 2018.

[5] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 338–345, 2012.

[6] K. B. Kaldestad, S. Haddadin, R. Belder, G. Hovland, and D. A. Anisi, "Collision avoidance with potential fields based on parallel processing of 3D-point cloud data on the GPU," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3250–3257, IEEE, 2014.

[7] T. Toner, D. M. Tilbury, and K. Barton, "Probabilistically Safe Mobile Manipulation in an Unmodeled Environment with Automated Feedback Tuning." 2022.

[8] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast Proximity Queries with Swept Sphere Volumes," *technical report of Department of Computer Science, UNC Chapel Hill*, pp. 1–32, 1999.

[9] M. Krämer, C. Rösmann, F. Hoffmann, and T. Bertram, "Model predictive control of a collaborative manipulator considering dynamic obstacles," *Optimal Control Applications and Methods*, vol. 41, no. 4, pp. 1211–1232, 2020.

[10] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, "Development of Human Support Robot as the research platform of a domestic mobile manipulator," *ROBOMECH Journal*, vol. 6, no. 1, 2019.

[11] S. R. Buss, "Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods," 2009.

[12] MathWorks, *Robotics System Toolbox*. Natick, Massachusetts, United States, 2021.

[13] MathWorks, *Optimization Toolbox*. Natick, Massachusetts, United States, 2021.

[14] J. L. Blanco-Claraco, "A tutorial on SE(3) transformation parameterizations and on-manifold optimization," no. 3, 2021.

[15] MathWorks, *Symbolic Math Toolbox*. Natick, Massachusetts, United States, 2021.

[16] M. Lange, D. Zühlke, O. Holz, and T. Villmann, "Applicationsof lp-norms and their smooth approximations for gradient based learning vector quantization," in *22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2014 - Proceedings*, no. April, pp. 271–276, 2014.