An Application of Model Predictive Control to Reactive Motion Planning of Robot Manipulators

Anastasia Mavrommati[1], Carlos Osorio[1], Roberto G. Valenti[1], Akshay Rajhans[1], and Pieter J. Mosterman[1]

***Abstract*— In recent years, a number of trajectory optimization algorithms have been proposed and established for motion planning of robot manipulators in complex, but static, predefined, environments. To enable reactive motion planning under uncertain conditions caused, for example, by moving obstacles, this paper proposes a formulation of the trajectory optimization problem that is tailored for model predictive control. The proposed algorithmic solution leverages off-the-shelf computational tools for nonlinear model predictive control, optimization, and collision checking. In addition, a motion planning paradigm is introduced to allow for online collision-free motion following a joystick command. The approach is validated in the context of an industrial pick-and-place application using MATLAB® and a Kinova® robot manipulator, both in simulation and experimentally.**

I. Introduction

Over the past decades, robotic manipulators, commonly referred to as robot arms, are being employed in industrial environments to perform highly repetitive, mundane maneuvers such as picking, placing, and sorting in static, well-known environments. In most cases, industrial robot manipulators use simple, interpolation-based planning algorithms to plan end-effector motions from point A to point B.

For flexible manufacturing, robot manipulators are required to work in cluttered, ever changing environments while collaborating with and avoiding collisions with human workers and other robotic entities. In addition, new warehouse automation paradigms combine robotic manipulators with mobile robots for expanding capabilities giving rise to situations where the workspace of the robot arm cannot be statically defined [1], [2].

To accommodate contemporary needs for dynamic collision avoidance, new algorithmic solutions have emerged but have yet to be widely adopted in professional settings. Sampling-based planners have become the key in this effort, mostly because of their versatility, speed, and generalizability. With the introduction of the Robot Operating System (ROS) [3], motion planning libraries based on sampling-based techniques, such as the Open Motion Planning Library (OMPL) [4], became readily available through packages such as MoveIt! [5]. This development has been conducive to the adoption of sampling-based planning methodologies in industrial settings, albeit with a long way to go.

On the other side of sampling-based path planning methods stand trajectory optimization approaches (model-free or model-based). Recently, algorithms such as CHOMP [6], TrajOpt [7], and STOMP [8] have revived interest in optimization techniques for motion planning. In addition, deployment tools (e.g., code generation in MATLAB® [9]) have allowed fast implementations of optimization algorithms.

Fig. 1. The experimental setup where a Kinova® Gen3 robotic arm performs a pick-and-place task in a warehouse environment. The Robotics System Toolbox™ [12] Support Package for Manipulators is used to interface with and control the robot arm using MATLAB® [9]. Stateflow® [13] for MATLAB® is used to schedule the pick-and-place tasks. The corresponding simulation environment is shown in Fig. 6. A video of the experiment results is provided in `https://github.com/stacymav/nonlinearmpc-content`.

Although a number of trajectory optimization algorithms for robotic systems (including robot arms) have been proposed, as mentioned above, model predictive control of robot manipulators for real-time optimization in dynamically-varying environments has not received measurable attention from the research and industry community. Model predictive control (MPC) in the context of trajectory optimization refers to optimizing a trajectory every t_s seconds using the current, updated knowledge of the robot and the world as input (e.g., current robot state and current obstacles state) [10]. Only a subset of the inputs of the calculated trajectory is applied to the system and the rest are discarded. Recently, model-based reinforcement learning has rekindled the interest on model predictive control approaches because it allows for constant re-planning as the model is refined based on new data [11].

The algorithm described in this paper proposes a problem formulation, that is, definition of the cost function and nonlinear constraints, that makes trajectory optimization suitable to be employed in MPC paradigms. In particular, compared to single trajectory optimization for full motion planning,

[1]Anastasia Mavrommati, Carlos Osorio, Roberto G. Valenti, Akshay Rajhans, and Pieter J. Mosterman are with The MathWorks, Inc., 3 Apple Hill Dr., Natick, MA 01760, {amavromm, cosorio, rvalenti, arajhans, pmosterm}@mathworks.com

trajectory optimization during a single MPC step does not require that the (temporally) terminal state of the trajectory be constrained to match the target state. Instead, the target state is encoded in the cost function, which promotes shorter time horizons for faster calculations. In addition, although trajectory optimization algorithms do not require the use of models (dynamic or kinematic), the proposed algorithm generates trajectories that satisfy a double integrator model as a proxy for constraints in curvature and displacement lengths from one time step to the next. More sophisticated models can also be used. In the proposed solution, MPC is used in supervisory control mode in order to generate desired position, velocity, and acceleration joint state that can be tracked using low-level controllers (e.g., Proportional Integral Derivative, PID, controllers). As a result, pre-definition of a reference trajectory is not required for planning. Only the final target pose is provided to the algorithm that outputs reference collision-free joint-space waypoints to track in real time.

The algorithm is validated in simulation in four different use cases: a) tracking static pose target with static obstacles, b) tracking static pose target with dynamic obstacles, c) tracking moving pose target with static obstacles, and d) tracking moving pose target with dynamic obstacles. The algorithm is tested in an experimental pick-and-place warehouse scenario in which a Kinova® arm is used to put objects on shelves. The examples aim to showcase the versatility of the approach and its applicability to a range of motion planning scenarios that are common in industrial settings.

This paper describes step by step how to formulate and solve a motion planning problem for manipulators in dynamic environments using readily-available software tools for nonlinear MPC, global optimization, and collision checking between mesh geometries. The outline of the paper is as follows: Section II aims to summarize related work in the field of trajectory optimization and MPC for robot manipulators. The theoretical background is provided in Section III, and the proposed problem formulation is described in detail in Section IV. Simulation and experiment results are shown in Section V, followed by a discussion on user choices for formulating the trajectory optimization problem for MPC algorithms in Section VI.

II. Related Work

Trajectory optimization has been commonly employed to generate energy-efficient trajectories in settings where energy consumption has a high impact on performance, such as for industrial robots [14]. Time-optimal trajectories are also of interest when the robot's workspace and mobility is expanded, as in the case of mobile manipulators [2].

Lately, there has been significant focus on establishing trajectory optimization techniques for collision-free planning in cluttered environments with static obstacles. CHOMP [6], STOMP [8], and TrajOpt [7] are all commonly-used algorithmic trajectory optimization solutions. TrajOpt handles collisions by encoding them as constraints in the optimization problem definition and relies on the computation of signed distances using convex-convex collision checking (an approach that is adapted in this paper, as well). Instead, both CHOMP and STOMP encode the collision avoidance objective in the cost function and employ the Euclidean distance transform that is precomputed on a voxel grid. All three algorithms assume that the goal of the planned trajectory is fixed and that the obstacles are static. They are originally designed to be model-free with the potential to add model information if required, such as in the case of non-holonomic robots. The problem of non-differentiability of collision avoidance constraints has been addressed in other work [15]. Several approaches for collision-free trajectory optimization of robotic manipulators with a focus on industrial applications have been proposed [16] [17]. The trajectory optimization solutions mentioned in this paragraph are not particularly suited for dynamic trajectory generation via model predictive control without modifications because the trajectory goal is encoded as terminal constraint in the problem definition.

For a robot manipulator to efficiently avoid dynamic obstacles, most available techniques propose reactive online modification of an offline trajectory every time a future collision is expected [18]. The generation of optimal robot manipulator trajectories for dynamic and/or reactive obstacle avoidance via model predictive control has not been studied as extensively. Most proposed methods are specifically tailored to a setting of interest, such as a manufacturing process [19] and a space operation [20], or to a robot of interest, such as the KUKA YouBot [21]. In work that is most closely related to the solution presented in this paper [19], the trajectory goal is also encoded in the cost function but as an error on the robot joint positions instead of a direct error on the target pose. See Section VI for a discussion on the cost function selection and the associated implications.

III. Background

We consider a robot manipulator with n revolute joints $q = [q_1, ..., q_n]$. For this study, the Kinova® Gen3 robotic arm with $n = 7$ joints is considered (Fig. 1). The 6-DoF (Degrees of Freedom) end-effector pose at configuration q is given as $P(q)$ and is calculated using the forward kinematic equations for open-chain manipulators. Here $P(q) = [p_x, p_y, p_z, \phi, \theta, \psi]$, where p_x, p_y, p_z denote the Cartesian coordinates on the x, y, and z axis, and ϕ, θ, ψ are the X-Y-Z Euler angles.

A. Model

The state of the robot manipulator is given by a concatenation of the robot joint positions and velocities such that $x = [q, \dot{q}] \in \mathcal{X} \subseteq \mathbb{R}^{2n}$. We use a double integrator to model the motion of each robot joint as follows:

$$\dot{x} = \begin{pmatrix} \mathbb{0}_{n\times n} & \mathbb{1}_{n\times n} \\ \mathbb{0}_{n\times n} & \mathbb{0}_{n\times n} \end{pmatrix} x + \begin{pmatrix} \mathbb{0}_{n\times n} \\ \mathbb{1}_{n\times n} \end{pmatrix} u, \tag{1}$$

where the inputs $u \in \mathbb{R}^n$ are joint accelerations $\ddot{q}$. Modeling the joints as double integrators guarantees smooth motions

while maintaining model linearity. It is certainly an option to include the full nonlinear robot dynamics, however since we are only using model predictive control in supervisory mode, this will affect the computational efficiency of the algorithm without significant impact on the resulting robot motion. Admittedly, including the robot dynamics would alleviate the need for a low-level controller to track resulting joint state (see Algorithm 1).

B. Trajectory Optimization

A model-based trajectory optimization problem $\mathcal{P}(J, t_0, x_0, T)$ is formulated as follows:

$$\begin{aligned}&\mathcal{P}(J, t_0, x_0, T): \\ &\text{Compute } u_{opt}(t)|_{t_0}^{t_0+T}, x_{opt}(t)|_{t_0}^{t_0+T} \\ &\text{that minimize cost } J(x(t), u(t))|_{t_0}^{t_0+T} \\ &\text{subject to} \\ &\text{system of equations } \dot{x} = f(x, u) \text{ with } x(t_0) = x_0, \\ &\text{and constraints } g(x(t)) \leq 0 \ \forall t \in [t_0, t_0 + T],\end{aligned} \tag{2}$$

where t_0 is the current time, T is the time horizon, $x(t), u(t)$ are the system state and inputs, respectively, at time t, and $J(x(t), u(t))|_{t_0}^{t_0+T}$ is the cost function defined over the state and input trajectories.

This continuous-time problem can be formulated as a discrete optimization problem. In this case, equality constraints are used to encode the (discretized) system equations. Accordingly, inequality constraints encode state limits and/or restrictions about the environment (e.g., obstacles to avoid). Similar to TrajOpt [7], to solve a single trajectory optimization problem $\mathcal{P}$, the well-known Sequential Quadratic Programming (SQP) method is employed which solves a Quadratic Programming (QP) subproblem at each iteration [22]. The function `fmincon` from Optimization Toolbox™ [23] available in MATLAB® [9] offers a generic implementation of this method and is used throughout.

C. Model Predictive Control

Model Predictive Control (as defined in this paper) solves a trajectory optimization problem $\mathcal{P}$ every t_s seconds with updated initial state and inequality constraints. The function `nlmpc` from Model Predictive Control Toolbox™ [24] available in MATLAB® [9] is used to perform nonlinear MPC.

IV. Problem Formulation

This section introduces a formulation of the trajectory optimization problem $\mathcal{P}$ in (2) that can be employed for model predictive control of robot manipulators to plan collision-free paths. This formulation includes the cost function definition, the constraints, and their respective Jacobians.

A. Cost Function Definition

The objective in this work is to plan a trajectory from initial end-effector pose P_A to target end-effector pose P_B without intersecting with obstacles in the environment while minimizing effort. As opposed to the majority of trajectory optimization solutions, the model predictive control paradigm allows including the target pose in the cost function instead of setting it as equality constraint. In particular, the cost is formulated on the task space instead of the joint space, to include the 6-DoF end-effector pose error directly by applying nonlinear forward kinematics on the robot state. Specifically, the cost function takes the following form:

$$\begin{aligned}&J = \int_{t_0}^{t_f} L(x(t), u(t))dt + K(t_f, x(t_f)) \\ &\text{with running cost} \\ &L = (P_B - P(q_t))^T Q (P_B - P(q_t)) + u(t)^T R u(t) \\ &\text{and terminal cost} \\ &K = (P_B - P(q_{t_f}))^T S_1 (P_B - P(q_{t_f})) + \dot{q}_{t_f}^T S_2 \dot{q}_{t_f}\end{aligned} \tag{3}$$

where q_t and q_{t_f} are the robot joint configurations at times t and t_f, respectively, t_f is the final time such that $t_f = t_0 + T$ with T being the time horizon of the optimization problem, that is, how far in the future the algorithm can see. The weight matrices $Q, S_1 \in \mathbb{R}^{6\times6}$ are symmetric, positive semi-definite and define the importance of each of the 6 target pose dimensions (translation and rotation) based on the requirements of the use case. The weight matrix $S_2 \in \mathbb{R}^{n\times n}$ determines the weight on the terminal joint velocities, while the matrix $R \in \mathbb{R}^{n\times n}$ regulates the cost on joint accelerations. See the results in Section V how different values of the weight matrices can be used for different use cases. For a discussion on other choices of cost functions as well as their advantages and disadvantages, see Section VI-A.

B. Constraints

Inequality constraints in (2) are employed for two purposes in the proposed problem formulation: i) to impose bounds on state and inputs, and ii) to avoid collision with obstacles. The

Algorithm 1: Model Predictive Control Algorithm

```
Initialize: current time t_0, time horizon T, time step t_s,
  current state x_0 = [q_0, q̇_0], current pose target P_B^(t_0),
  current obstacle constraints g^(t_0), cost weight matrices
  Q, R, S_1, S_2;
while Termination Condition do
    Solve P(J, t_0, x_0, T) to compute x_opt(t)|_{t_0}^{t_0+T};
    x_ref ← x_opt(t_0 + t_s);
    Use low-level controller to track x_ref ;
    t_0 ← t_0 + t_s;
    x_0 ← get_current_state() ;
    if dynamic obstacles then
        Obtain obstacles location at t_0 ;
        Update obstacles O_j in constraint g^(t_0) in (4) ;
    end
    if moving pose target then
        P_B^(t_0) ← update(P_B);
        Update cost function J in (3);
    end
end
```

former are linear constraints, while the latter are nonlinear constraints, and they are both described in more detail in the following subsections.

1) Linear inequality constraints: Upper and lower bounds are imposed on the robot state x (joint positions and velocities) and inputs u (joint accelerations) as inequality constraints to problem $\mathcal{P}$, such that $x_{min} < x < x_{max}$ and $u_{min} < u < u_{max}$. Imposing constraints on input acceleration is an indirect way to generate curvature-constrained trajectories.

2) Nonlinear inequality constraints: Trajectories that avoid collisions with the world are planned by specifying appropriate inequality constraints. The world is represented as a set of obstacles, with each obstacle being a distinct convex mesh. Meshes can be primitives (box, cylinder, or sphere) or customized imported convex meshes.

To generate collision-free trajectories, a method from related work [7] is adopted to add discrete optimization constraints on the minimum distance between the robot links and the convex representation of obstacles using convex-convex collision checking. Inequality constraints require that every robot body is at a safe distance from every obstacle (mesh) in the scene by setting the following constraint in the optimization problem (2):

$$\begin{aligned} g_{i,j}(x(t)) &= -(dist(\mathcal{L}_i, \mathcal{O}_j) - SD)|_t \\ \text{for } i &= 1, ..., k \text{ and } j = 1, ..., m \end{aligned} \tag{4}$$

where k and m are the total number of robot links and convex obstacles respectively, $\mathcal{L}_i$ and $\mathcal{O}_j$ are the i^{th} robot link and j^{th} obstacle, respectively, and SD is the user-defined safety distance required between meshes. Collision checking and minimum distance calculation on convex meshes are applied to calculate this constraint. An implementation of such an approach is available in Robotics System Toolbox™ [12] in MATLAB® [9]. Detailing the collision checking methodology is out of scope for this paper.

Since handling of collisions is not the focus of this paper, this algorithm implementation does not consider continuous-time collision checking or any other additional collision penalties. These can be added based on preference with no significant change to the concept of the presented algorithm. For additional details, the reader is referred to [7].

C. Jacobian functions

To speed up each iteration run of the optimization algorithm, the Jacobians of the nonlinear cost function and inequality constraint function are derived.

The Jacobians of the nonlinear cost function in (3) are the following:

$$\begin{aligned} \frac{\partial J}{\partial q}\Big|_t &= -(P_B - P(q_t))^T Q \mathcal{J}_a^{ee}(q_t) \\ \frac{\partial J}{\partial q}\Big|_{t_f} &= -(P_B - P(q_{t_f}))^T S_1 \mathcal{J}_a^{ee}(q_{t_f}) \\ \frac{\partial J}{\partial \dot{q}}\Big|_t &= \mathbb{0}, \ \frac{\partial J}{\partial \dot{q}}\Big|_{t_f} = \dot{q}_{t_f}^T S_2, \ \frac{\partial J}{\partial u}\Big|_t = u(t)^T R \end{aligned} \tag{5}$$

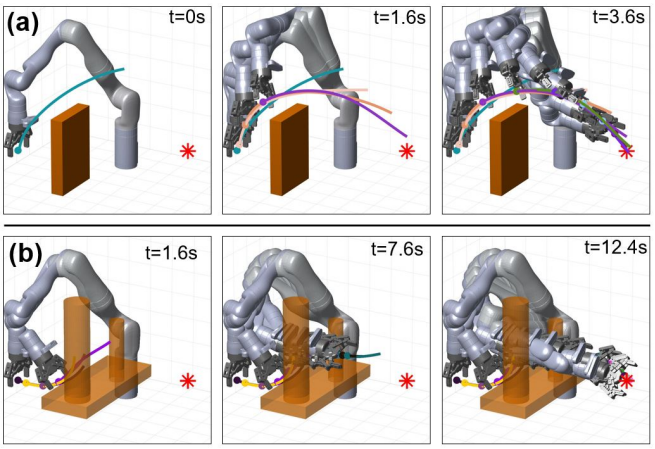

Fig. 2. Two examples, (a) and (b), of the Kinova® robot arm tracking a static target pose shown as a red asterisk while avoiding the orange obstacles. Each open-loop trajectory is shown with different color. The terminal weight matrix S_1 in (3) is defined such that the final arm rotation in the yaw direction does not matter. This use case is common when the robot arm is used for inspection purposes. Note that the open-loop trajectories are computed in joint space but only the corresponding task-space trajectories are shown for clarity.

where $\mathcal{J}_a^{ee}(\cdot)$ is the analytic Jacobian matrix of the robot calculated at the end effector.

The Jacobians of the nonlinear inequality constraint function in (4) are calculated as

$$\begin{aligned} \frac{\partial g_{i,j}}{\partial q}\Big|_t &= \vec{n}_{i,j}^T \, \mathcal{J}^{\mathcal{L}_i}(q_t) \\ \frac{\partial g_{i,j}}{\partial \dot{q}}\Big|_t &= \mathbb{0}, \ \frac{\partial g_{i,j}}{\partial \dot{u}}\Big|_t = \mathbb{0} \end{aligned} \tag{6}$$

where $\mathcal{J}^{\mathcal{L}_i}(q_t)$ are the last 3 rows of the geometric robot Jacobian matrix calculated at the point on the robot link $\mathcal{L}_i$ that is closest to obstacle $\mathcal{O}_j$. In addition, $\vec{n}_{i,j}^T$ is the 3×1 normalized vector in the direction of the smallest translation that puts robot link $\mathcal{L}_i$ and obstacle $\mathcal{O}_j$ in contact. For more information on the derivation, kindly refer to [7].

D. Final algorithm

The complete MPC steps are given in Algorithm 1. The *Termination Condition* is selected according to the use case, e.g., whether the desired target is static or moving. More information about different choices of termination conditions is provided in the Results section that follows.

V. Results

A number of common industrial use cases were selected to test the approach in simulation and experiment. All the subsequent examples assume the existence of sufficient infrastructure for detecting obstacles in real time at a rate that is at a minimum equal to the MPC sampling rate (e.g., related work [19]). The examples were simulated on a 1.9 GHz Intel Core i7 personal laptop running Windows with 16 GB memory. The algorithm was implemented in MATLAB® [9]. Subsequently, MATLAB Coder™ [25] was used to generate C code and speed up the algorithm execution. All the simulations thereafter were run in MATLAB.

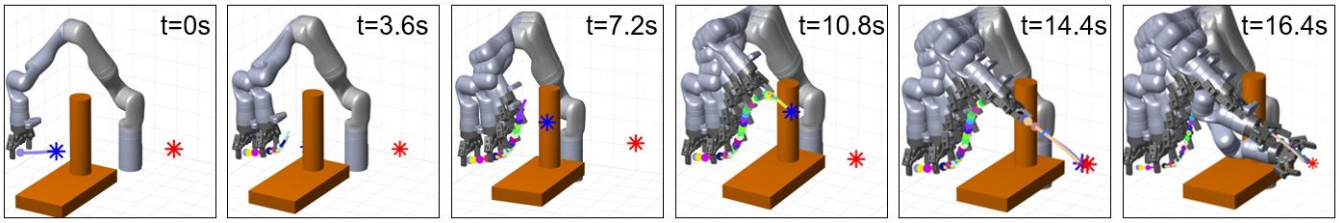

Fig. 3. An example of the Kinova® robot arm tracking a static final target pose while avoiding a cylindrical obstacle that is moving towards the robot base. The final target pose (red asterisk) is the same as in the examples in Fig. 2. The blue asterisk shows the intermediate target pose at each time step that is computed according to (7). The motion of the obstacle is not known a priori, only its current location is communicated to the robot at each time step. Each open-loop trajectory is shown with different color. Note that the open-loop trajectories are computed in joint space but only the corresponding task-space trajectories are shown for clarity.

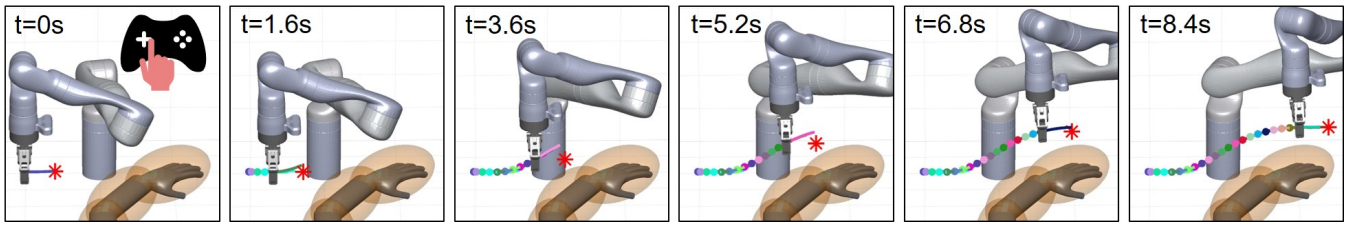

Fig. 4. An example of the Kinova® robot arm following a joystick command to "jog" towards the positive y task-space direction by tracking a moving target pose shown as a red asterisk. The moving target pose at each time step is computed according to (7). Collision with a set of static obstacles (human arm modelled as a set of ellipsoids) is avoided. Compared to the example in Fig. 3, the final target pose is not fixed but rather changes according to the obstacles the robot must avoid. Each open-loop trajectory is shown with different color. Note that the open-loop trajectories are computed in joint space but only the corresponding task-space trajectories are shown for clarity.

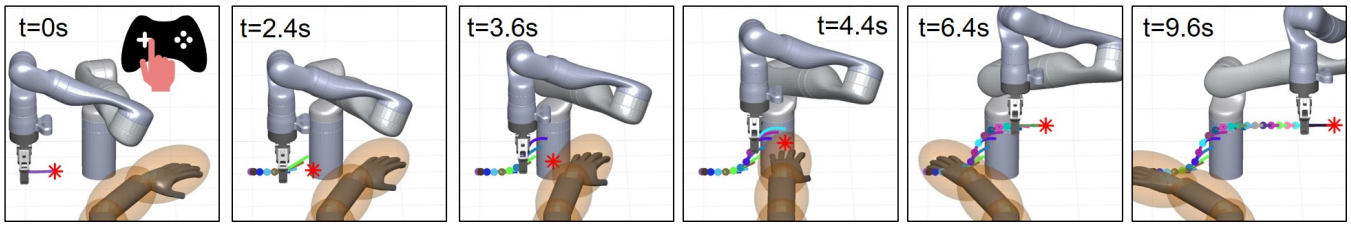

Fig. 5. An example of the Kinova® robot arm following a joystick command to "jog" towards the positive y task-space direction by tracking a moving target pose shown as a red asterisk. The moving target pose at each time step is computed according to (7). Collision with a set of moving obstacles (human arm modelled as a set of ellipsoids) is avoided. Each open-loop trajectory is shown with different color. Note that the open-loop trajectories are computed in joint space but only the corresponding task-space trajectories are shown for clarity.

In all the simulation examples, the following parameters have been used unless otherwise noted: time step $t_s = 0.4s$, time horizon $T = 1.6s$, weight matrices $Q = diag([0.1, 0.1, 0.1, 0, 0, 0])$, $S_1 = diag([10, 10, 10, 1, 1, 1])$, $S_2 = diag([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01])$, and $R = diag([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])$. Numerical optimization of open-loop problem (2) using SQP terminates after a maximum of 5 iterations (unless convergence has been reached at fewer iterations) to promote speed of algorithm execution.

A. Static target

The robot is instructed to reach a static target pose P_B to complete a high-level task (e.g., pick or place an object). The motion is complete when the static target is reached within a desired tolerance. The *Termination Condition* in Algorithm 1 is set to $diff\,[P_B^{(t_0)} - P(q(t_0))] < \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ is a vector of tolerances, that is, $\boldsymbol{\epsilon} \in \mathbb{R}^6$, and *diff* a function that computes the difference between two poses by appropriately handling angle wrapping.

1) Static obstacles: The robot is instructed to reach a target pose P_B without enforcing a specific yaw angle at the end effector. This use case is applicable in scenarios where the robot is used for inspection purposes with an onboard camera, and is achieved by setting the weight matrix $S_1 = diag([10, 10, 10, 1, 1, 0])$. Fig. 2 shows the evolution of the resulting closed-loop trajectory as well as the intermediate open-loop trajectories. Notice how the intermediate trajectories do not reach the final target but only serve to drive the robot towards it. This will be useful in the next use case where the obstacles are moving.

2) Dynamic obstacles: When the robot manipulator is known to operate in an environment with dynamic obstacles, the target pose $P_B^{(t_i)}$ at each time step t_i is set to an intermediate pose in the direction of the final target pose P_B, till the robot is sufficiently close to the final target.

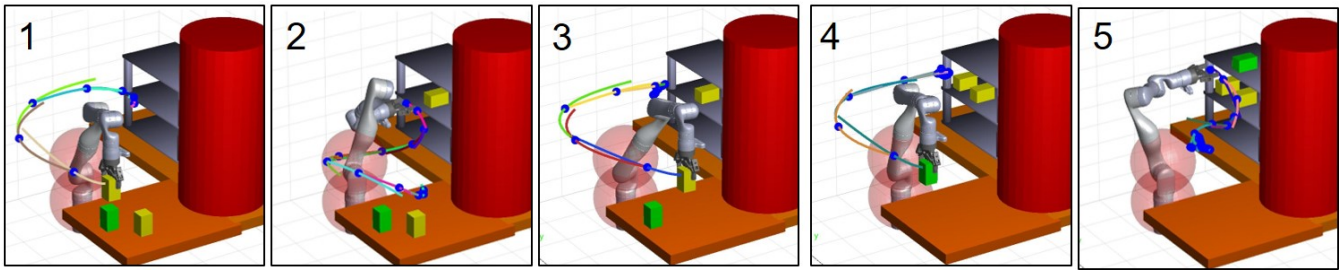

Fig. 6. The Kinova® robot arm is controlled to pick up three objects and place them on the shelves according to their color (yellow to middle shelf, and green to top shelf), while avoiding obstacles in the scene. Each open-loop trajectory is shown with a different color. Blue dots indicate the closed-loop trajectory points that the robot actually visited. Note that the open-loop trajectories are computed in joint space but only the corresponding task-space trajectories are shown for clarity.

Specifically:

$$P_B^{(t_i)} = P_A^{(t_i)} + speed \cdot targetDirection \cdot t_s \qquad (7)$$

where $targetDirection = diff(P_A^{(i)}, P_B)$ with $P_A^{(i)}$ the end effector pose at time step t_i and *diff* a function that computes the difference between two poses by taking angle wrapping into account. Updating the target pose at each time step to an intermediate target pose, instead of setting it to the final desired pose directly, ensures that the robot will proceed cautiously to its final location, always "looking out" for new obstacles obstructing its path to the goal. The variable $speed$ affects how fast the robot is expected to move to the final target pose. Setting the $speed$ to a low number allows the robot to proceed more cautiously to the target pose, with higher probability of effectively avoiding dynamic obstacles. The motion of the obstacle is not known to the planning algorithm, only the current obstacle location is shared at each time step. Fig. 3 shows how the robot moves cautiously while avoiding a cylindrical obstacle that is moving towards the robot base. Initially, the robot attempts to move over the cylinder, but as the cylinder moves out of the way, the robot ends up finding a collision-free path around the side of the obstacle.

B. Moving target: joystick-controlled motion

Similar to how dynamic obstacles are handled, the robot is assigned a new target pose $P_B^{(t_i)}$ at every time step t_i, according to the direction of task-space motion dP instructed by a joystick (e.g., pressing the "right arrow" will instruct the end effector to move to the positive direction of the global y axis without changing its task space orientation[1]). The new target $P_B^{(t_i)}$ is calculated relative to the current end effector pose $P_A^{(t_i)}$, without any knowledge of the obstacles in the scene, based on equation (7) with $targetDirection = dP$.

1) Static obstacles: This use case is tested in a scenario where the robot is instructed to follow a task-space joystick command while avoiding the static arm of an operator. To enable collision avoidance, the human arm is modelled as a set of ellipsoid meshes. The evolution of motion is shown in Fig. 4. Note how the final target pose is not static, as in Fig. 3, but varies according to the obstacles the robot must avoid while completing the motion. To see a video of the resulting motion, visit `https://github.com/stacymav/nonlinearmpc-content`.

2) Moving obstacles: This use case is tested in a scenario where the robot is instructed to follow a task-space joystick command while avoiding the moving arm of an operator. The motion of the robot is shown in Fig. 5 and can be compared to the one performed when the human arm was not moving (which is shown in Fig. 4). Notice how the the manipulator ends up reacting to the updated human arm locations at each time step by moving upwards much faster. The intermediate target pose is always set to the direction of the instructed motion (here towards the global y direction) even if it coincides with an obstacle. Since the target pose is not encoded as a hard terminal constraint, the algorithm simply outputs, at each time step, an open-loop trajectory that moves the robot close enough to the target while avoiding the current obstacles. To see a video of the resulting motion, visit `https://github.com/stacymav/nonlinearmpc-content`.

C. Experiment results: Pick-and-place warehouse scenario

The model predictive control algorithm was applied to plan collision-free paths for a typical warehouse scenario in which a robotic arm is instructed to pick up the objects on a bench and place them on shelves. The resulting trajectories from pure simulation are shown in Fig. 6. The simulation results were validated in experiment using the setup shown in Fig. 1. The experiment results are shown in Fig. 7. A video of the experiment results is provided at `https://github.com/stacymav/nonlinearmpc-content`.

VI. Considerations - Discussion

This section discusses the optimization design and the user choices that can affect the performance of a model predictive control algorithm employed to generate collision-free trajectories.

A. Cost function

As mentioned earlier, our objective is to plan a trajectory from initial end-effector pose P_A to target end-effector pose P_B without intersecting with obstacles in the environment

[1]In the realm of industrial robots, this operation is known as "jogging the robot" using a handheld controller, referred to as teaching pendant.

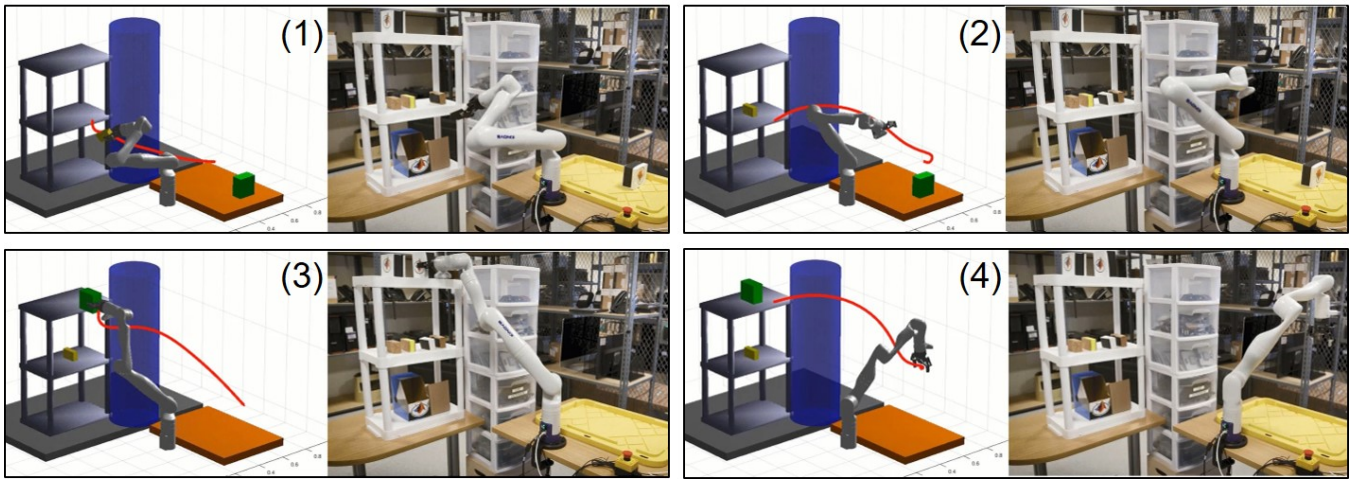

Fig. 7. The Kinova® robot arm is controlled to pick up three objects and place them on the shelves according to their color (yellow to middle shelf, and green to top shelf), while avoiding obstacles in the scene. The experimental setup is explained in Fig. 1. A video of the experiment results is provided in `https://github.com/stacymav/nonlinearmpc-content`.

while minimizing effort. In addition to the cost function definition proposed in (3), there are multiple ways one can formulate the cost function to achieve this single objective.

One approach is to formulate the cost around desired performance (i.e., joint effort, or sum of joint angle displacements [7], [26]) irrespective of the target pose P_B. This requires that the target pose is encoded in the terminal constraint, instead, most often the final target pose is constrained to be inside a target zone for added tolerance. For pure trajectory optimization without the receding horizon of model predictive control, this choice is preferred [6], [7] because it guarantees reaching the target pose at desired tolerance. In MPC, however, this would impose unnecessary limitations to the solutions of a single optimization run: we do not need the robot to reach the desired pose in a single run, we just need it to move closer to the target compared to the previous run. If we simply guarantee that the cost decreases at each run, the robot will reach its target eventually.

A second approach is to include the target pose objective in the cost function. The most straightforward way to achieve this is to formulate the cost on the joint space, i.e., on the joint positions error [21]. Target 6-DoF pose is mapped to target n-dimensional joint configuration via inverse kinematics. This renders the cost function quadratic with respect to the state x (i.e., robot joint positions q) but comes with limitations arising from the kinematic redundancy of the manipulators under study. In short, there are no guarantees that the target joint configuration (derived as an inverse kinematic solution) satisfies the problem constraints, and there is no direct method to prioritize a subset of the 6 target pose dimensions.

B. Time horizon

In trajectory optimization, the time horizon determines—in simple terms—how much time one allows the robot to reach its target destination. Therefore, it has a significant impact on the algorithm performance. Instructing the robot to move a $1m$ distance in $2s$ while minimizing energy consumption will result in a different motion from instructing it to move the same distance in $10s$. For some choices of the time horizon, the robot cannot complete the instructed motion within the time limits because of actuation constraints. When obstacles are present, it is more challenging to estimate the time the robot might need to reach a destination because there is no straightforward way of knowing how the robot will try to go around the obstacle.

In model predictive control, the choice of time horizon has a different impact, possibly less critical. Because of its iterative nature, as we explained earlier in the construction of the cost function, we are able to include the target destination requirement in the cost function instead of in the constraint. This means that regardless of what the time horizon is, we do not expect the robot to reach its destination within this time horizon in any case. What we do expect purely for stability purposes is that the cost will be reduced—or at least will not increase—at every optimization run [27].

To promote a lower computational time as required by the iterative nature of the optimization algorithm, a shorter time horizon can be used. This is only possible because there is no constraint on the final state of the optimizes trajectory—the target robot pose is included in the cost function instead (see (3)).

C. Initial guess of optimization problem

The initial estimate of the first MPC iteration is set to the current joint state q with zero joint velocities and accelerations (i.e., the robot is stationary at its current configuration). Subsequent iterations use the suboptimal trajectory of the previous iteration as the initial estimate. The problem of insufficient convergence due to initial guess has been addressed by researchers, with a number of solutions proposed, including recently with an application of model-based reinforcement learning [28].

D. Execution time

The execution time for a single run of the open-loop trajectory optimization problem $\mathcal{P}$ in (2) depends largely on the user choice of underlying utilities, such as optimization solver and collision checking algorithm. The examples presented in this paper were implemented with generic, not problem-specific tools, resulting in an average of 0.6 seconds of computation time per single open-loop solution.

VII. Summary

This paper proposes the application of MPC algorithms to reactive motion planning of robot manipulators by introducing a suitable formulation of the open-loop trajectory optimization problem. The proposed algorithm can be implemented by leveraging off-the-shelf tools to enable industrial applications. The approach is verified in both simulation and experiment. We demonstrate how collision with randomly moving obstacles can be more efficiently avoided by incorporating intermediate target poses for each open-loop optimization problem at every time step. Future work will focus on verifying and proving convergence of the proposed algorithm in diverse situations, including number and shape of obstacles, and time step and time horizon selections.

References

[1] E. Venator, G. S. Lee, and W. Newman, "Hardware and software architecture of abby: An industrial mobile manipulator," in *2013 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2013, pp. 324–329.

[2] S. Thakar, L. Fang, B. Shah, and S. Gupta, "Towards time-optimal trajectory planning for pick-and-transport operation with a mobile manipulator," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2018, pp. 981–987.

[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[4] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[5] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[6] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 489–494.

[7] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: science and systems*, vol. 9, no. 1. Citeseer, 2013, pp. 1–10.

[8] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[9] MathWorks®, *MATLAB®*. Release R2020b, September 2020.

[10] G. Pannocchia, "Handbook of model predictive control [bookshelf]," *IEEE Control Systems Magazine*, vol. 40, no. 5, pp. 96–99, 2020.

[11] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[12] MathWorks®, *Robotics System Toolbox™*. Release R2020b, September 2020.

[13] ——, *Stateflow®*. Release R2020b, September 2020.

[14] C. Hansen, J. Öltjen, D. Meike, and T. Ortmaier, "Enhanced approach for energy-efficient trajectory generation of industrial robots," in *2012 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2012, pp. 1–7.

[15] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology*, 2020.

[16] S. Björkenstam, D. Gleeson, R. Bohlin, J. S. Carlson, and B. Lennartson, "Energy efficient and collision free motion of industrial robots using optimal control," in *2013 IEEE international conference on automation science and engineering (CASE)*. IEEE, 2013, pp. 510–515.

[17] H. Ding, G. Reißig, D. Groß, and O. Stursberg, "Mixed-integer programming for optimal path planning of robotic manipulators," in *2011 IEEE International Conference on Automation Science and Engineering*. IEEE, 2011, pp. 133–138.

[18] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[19] D. Han, H. Nie, J. Chen, and M. Chen, "Dynamic obstacle avoidance for manipulators using distance calculation and discrete detection," *Robotics and Computer-Integrated Manufacturing*, vol. 49, pp. 98–104, 2018.

[20] M. Wang, J. Luo, and U. Walter, "A non-linear model predictive controller with obstacle avoidance for a space robot," *Advances in Space Research*, vol. 57, no. 8, pp. 1737–1746, 2016.

[21] G. B. Avanzini, A. M. Zanchettin, and P. Rocco, "Reactive constrained model predictive control for redundant mobile manipulators," in *Intelligent Autonomous Systems 13*. Springer, 2016, pp. 1301–1314.

[22] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[23] MathWorks®, *Optimization Toolbox™*. Release R2020b, September 2020.

[24] ——, *Model Predictive Control Toolbox™*. Release R2020b, September 2020.

[25] ——, *MATLAB Coder™*. Release R2020b, September 2020.

[26] S. Diao, X. Chen, L. Wu, M. Yang, and J. Liu, "The optimal collision avoidance trajectory planning of redundant manipulators in the process of grinding ceramic billet surface," *Mathematical Problems in Engineering*, vol. 2017, 2017.

[27] A. Mavrommati, E. Tzorakoleftherakis, I. Abraham, and T. D. Murphey, "Real-time area coverage and target localization using receding-horizon ergodic exploration," *IEEE Transactions on Robotics*, vol. 34, no. 1, pp. 62–80, 2017.

[28] H. Bharadhwaj, K. Xie, and F. Shkurti, "Model-predictive control via cross-entropy and gradient-based optimization," in *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, ser. Proceedings of Machine Learning Research, vol. 120. PMLR, 2020, pp. 277–286.