

Trajectory Generation with Load Constraints for Robotic Manipulators

David L. Bacher[†] and John R. Cooper[‡]
NASA Langley Research Center, Hampton, VA, 23666

Javier Puig-Navarro[§]
Analytical Mechanics Associates, Hampton, VA, 23666

Future large spacecraft will utilize robotic manipulators for in-space servicing, assembly, and manufacturing. Due to launch mass constraints, such manipulators will be designed to be as lightweight as possible. Trajectory generation algorithms will need to factor in load constraints to avoid overexerting and damaging manipulators. This paper investigates an approach that combines optimal Rapidly-exploring Random Trees, spline interpolation, and Model Predictive Control to generate a manipulator trajectory which respects load constraints.

I. Introduction

The contribution of this paper is a sample-based trajectory generator with a model predictive control (MPC) inner loop. The method is inspired by robotic in-space servicing, assembly, and manufacturing (ISAM) missions where robotic manipulators, designed with launch mass constraints in mind, will be lightweight and must operate within load constraints to avoid overexerting or damaging themselves or their payloads. Unlike in assembly lines, where robots perform a single task repeatedly, robots performing in-space assembly may be required to perform a variety of different tasks. For example, a single robotic manipulator assembling a habitat on the surface of the moon may be required to lift objects of varying sizes and weights. Heavier objects may restrict its range of motion or speed due to torque limits on the motors. A trajectory generation algorithm that adheres to load constraints would allow a robot to calculate a safe way to lift an object at the edge of or even outside its operational envelope.

Current ISAM efforts at NASA Langley Research Center (LaRC) use a time-expansion method by Kuntz and Stilman [1] to handle load constraints, which is included in the ROS2 MoveIt motion planning package. The path of the robot is planned by the sample-based planner RRT* [2], an optimizing variant of the Rapidly-exploring Random Trees (RRT) algorithm [3], and a polynomial spline curve is fit through the sampled points to create a continuous path. The robot moves along the resulting path at the maximum speed that avoids violating load constraints. The time expansion method results in inefficiencies because the speed of the trajectory is limited by whichever joint is operating closest to its load limit. This paper was motivated by a desire to devise an improved method that explicitly considers load constraints in the path planning algorithm.

MPC is an optimal control method in which a dynamic system is controlled by solving a constrained optimization problem over a finite time horizon. MPC is traditionally used as a real-time controller, but it can also be used as a motion planner. Performing a dynamic simulation of a manipulator controlled by an MPC controller results in a planned trajectory. If load is included as a constraint in the MPC optimizer, the load constraints will not be violated in the model. Since the optimization problem is solved at every time step, the model will never deviate far from the real robot, thus load constraints are reliably adhered to.

If the constraints are non-convex, MPC can get trapped in local minima. In a cluttered environment such as a spacecraft in the midst of being assembled or serviced, the collision constraint is non-convex. Sample-based planners are robust to local minima because they uniformly sample the state space. For this reason, a sample-based planner is used to generate a global path. In theory, any sample-based planner could be used. This paper uses a variant of RRT* with an adaptive step size which was developed at LaRC by Cooper et al. [4]. The method is similar to the basic RRT*, but with a step size that is inversely proportional to a clutter metric. The clutter metric is based on the density of objects in the vicinity of the parent node [5]. The step size is made large in sparse regions and small in cluttered regions. MPC is then used to plan the motion between the states generated by RRT*. In effect, MPC does the majority of the planning

[†] Student Trainee (Engineering), Autonomous Integrated Systems Research Branch

[‡] Research Engineer in Navigation, Guidance, and Control Systems, Autonomous Integrate Systems Research Branch

[§] Senior Aerospace Engineer, Analytical Mechanics Associates

in sparse regions, while RRT* does the majority of the planning in cluttered regions where MPC is susceptible to local minima.

II. Algorithm Description

A. Global Planning: RRT* with Adaptive Step Size

The global path planning is performed using RRT*. The algorithm builds a tree of nodes, each of which represents a manipulator joint state configuration. Each run through the main loop, the joint space is uniformly sampled, and a new node is created by taking a step of size δ in the direction of the sampled state. If the new state is valid (satisfies all constraints), a node is added to the tree and connected to a parent node chosen as the closest node in the tree. The optimizing variant also rewires the tree within the vicinity of new nodes to minimize a cost function. RRT* is probabilistically complete and converges asymptotically to a path that connects the start state, q_{start} and the goal state, q_{goal} with minimal path cost [2]. In this case, path length is used as the cost function.

Let $q_{start} \in Q$ and $q_{goal} \in Q$ be the start and goal state of some desired motion of a robotic manipulator. Let $C(q, q_{new})$ be the cost of transitioning from state q to state q_{new} . Let δ be the RRT* step size. To increase the speed of convergence, there is a probability p_{goal} of sampling the goal state instead of a random state.

Algorithm 1 RRT* with adaptive step size

```

1: function RRT*( $q_{start}, q_{goal}, p_{goal}$ )
2:   initialize search tree  $T$  with  $q_{start}$ 
3:    $success \leftarrow False$ 
4:   while time limit not exceeded do
5:      $dice \leftarrow$  sample from  $U(0, 1)$ 
6:     if  $dice < p_{goal}$  then
7:        $q_{samp} \leftarrow q_{goal}$ 
8:     else
9:        $q_{samp} \leftarrow$  sample from  $Q$ 
10:     $q_{parent} \leftarrow \arg \min_{q \in Q_{near}} C(q, q_{new}, T)$ 
11:     $q_{new} \leftarrow q_{parent} + \delta \frac{q_{samp} - q_{parent}}{\|q_{samp} - q_{parent}\|}$ 
12:    if  $q_{new}$  is a valid pose then
13:       $Q_{near} \leftarrow$  nodes in  $T$  with distance to  $q_{new} < \delta_{near}$ 
14:      add  $q_{new}$  to  $T$  with an edge from  $q_{parent}$  to  $q_{new}$ 
15:      for  $q \in Q_{near}$  do
16:        if  $C(q_{new}, q, T) < C(q_{parent}, q, T)$  then
17:           $q_{parent} \leftarrow q_{new}$ 
18:      if  $q_{new}$  is in  $Q_{goal}$  then
19:         $success \leftarrow True$ 
20:         $q_{end} \leftarrow q_{new}$ 
21:    if  $success = False$  then
22:       $q_{end} \leftarrow$  node in  $T$  with minimum distance to  $q_{goal}$ 
  return success and the motion from  $q_{start}$  to  $q_{goal}$ 

```

The result of the RRT* algorithm is a discontinuous series of states connecting q_{start} to q_{goal} . MPC will be used to allocate time to the path while avoiding violation of constraints. Before that happens, the discontinuities are removed with spline interpolation. If this is not done, the manipulator will have to slow down and stop at each discontinuity.

B. Local Planning: Model Predictive Control

MPC requires a dynamic model of the system in the form:

$$\dot{x}(t) = f(x(t), u(t)) \quad (1)$$

where x is the system state, u is the control input, and t is time. At each time step, the optimal control problem is solved over a finite time horizon, resulting in a trajectory of inputs that minimize the cost function

$$c(x(t_0), u(t_0)) = \sum_{t=t_0}^{t_h} (x(t)^T Q x(t) + u(t)^T R u(t)) \quad (2)$$

where Q and R are weight matrices, subject to constraints in the form:

$$g(x(t), u(t)) \leq 0 \quad (3)$$

$$h(x(t), u(t)) = 0 \quad (4)$$

The general form of the equation of motion for a serial link robotic manipulator with N links is:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) + J(q)^T f = \tau \quad (5)$$

where

- q = $N \times 1$ vector of joint states
- τ = $N \times 1$ vector of joint torques
- f = 6×1 vector of external forces and moments applied to the end effector
- $M(q)$ = $N \times N$ symmetric moment of inertia matrix
- $C(q, \dot{q})$ = $N \times N$ matrix representing centrifugal and Coriolis effects
- $G(q)$ = $N \times 1$ vector of gravitational torques on each joint
- $J(q)$ = $6 \times N$ Jacobian matrix

The $M(q)$, $C(q, \dot{q})$, and $G(q)$ matrices are calculated algorithmically at each time step using the methods described in Walker and Orin [6]. Dynamic simulations performed in this paper are based on the Denavit-Hartenberg parameters [7]. Implementations of these algorithms came from the Python Robotics Toolbox by Corke and Haviland [8].

The manipulator tracks the prescribed path using an MPC controller to follow a virtual leader, which moves in 1D along the path. Recall that the path is a polynomial spline interpolation between the RRT* generated waypoints. The path can therefore be represented by a polynomial function $q = f(s)$, which maps a normalized parameter s onto joint space. s represents the distance traveled along the path as a fraction of the total path length.

An update law for the virtual leader is defined such that it slows down if it gets too far ahead of the manipulator and as it approaches the end of the path. The speed of the virtual leader in 1D path space is:

$$\dot{s}_t = \alpha(1 - s)^2 - \beta||q_t - q||^2 \quad (6)$$

where

- s_t = position of the virtual leader in 1D path space
- q_t = position of the virtual leader in ND joint space
- α, β = scalar weights for tuning virtual leader motion

At each time step, the MPC algorithm integrates the equations of motion forward from the current time step t_0 to the end of the time horizon t_h , and selects a series of joint torques which minimize the cost function below. The cost function penalizes deviation from the virtual leader's state to drive the manipulator towards the virtual leader's position. It also penalizes joint torques to minimize the control effort. Q and R are matrices of parameters to tune the effect of each term of the cost function on the resulting system behavior.

$$c(q, \tau) = \sum_{t=t_0}^{t_h} (q_t - q)^T Q (q_t - q) + \tau^T R \tau \quad (7)$$

subject to the following constraints:

$$q_{min} - q \leq 0 \quad (8)$$

$$q - q_{max} \leq 0 \quad (9)$$

$$|\dot{q}| - \dot{q}_{max} \leq 0 \quad (10)$$

$$|\tau| - \tau_{max} \leq 0 \quad (11)$$

$$-d_{min}(q) \leq 0 \quad (12)$$

Equations (8) and (9) prevent the manipulator from exceeding mechanical joint angle limits. Equation (10) prevents the robot from exceeding maximum joint speeds. Equation (11) prevents the manipulator from exceeding joint torque limits, thus explicitly considering load constraints during the trajectory generation process. Equation (12) prevents collisions. d_{min} represents the minimum distance between the manipulator and all objects in the scene. This value must be greater than zero to prevent collisions. Collision checking was done using the built in mesh collision checking algorithms in the robotics toolbox by Corke and Haviland [8], which is based on PyBullet [9].

Implementation of the MPC algorithm came from the Python do-mpc toolbox by Fiedler et al. [10].

III. Results

The RRT*-MPC trajectory generation algorithm was tested on a Universal Robots UR3 robotic manipulator with six joints. The algorithm was tasked with planning the same motion in two environments; one in which there were no obstacles to avoid apart from the floor, and one in which there was an obstacle in the path of the robot. The obstacle used was a TriTruss, a structural element used in ISAM projects at LaRC. The corner of the TriTruss base was placed 420 mm from the center of the robot's base, oriented as shown in figure 1. Parameters of the planned motion and parameters passed to the trajectory planner are listed in tables 1 through 3. Once the trajectories were generated, points from the trajectory were passed to the UR3 at intervals of dt , the same time step used by the MPC controller during motion planning. The UR3 tracked the trajectory using its own built-in joint position controller.

Table 1 Trajectory Parameters

Parameter	Value
q_{start}	$[-1.57, -1.83, -2.28, -.59, 1.23]$ rad
q_{goal}	$[1.57, -1.83, -2.28, -.59, 1.23]$ rad
q_{start}	$[-1.57, -1.83, -2.28, -.59, 1.23]$ rad

Table 2 MPC Parameters

Parameter	Value
dt	.05 s
h	20
Q	$100\mathbb{I}_{12 \times 12}$
R	$\mathbb{I}_{6 \times 6}$
α	1
β	1

Table 3 Constraints

Parameter	Value
τ_{max}	$[56, 56, 28, 12, 12, 12]$ Nm
q_{max}, q_{min}	$\pm 2\pi$ rad for each joint
$\dot{q}_{max}$	π rad/s

Figure 2 shows example generated trajectories. The 6D state space trajectory is represented as the 3D path of the end effector through the work space by multiplying by the manipulator Jacobian $J(q)$.

Fig. 1 Robot and environment used in testing

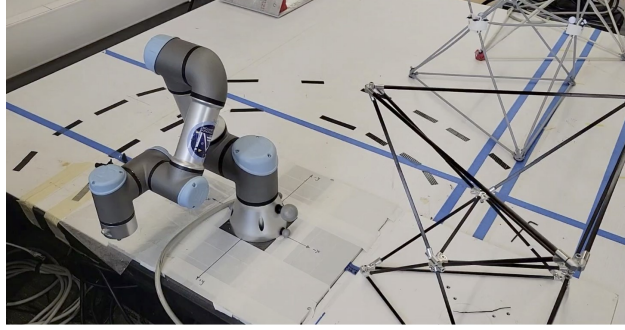
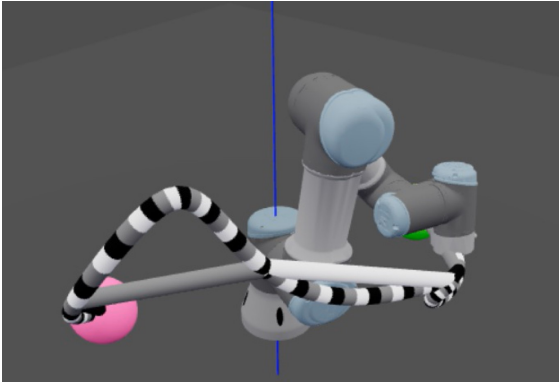


Fig. 2 Sample trajectories



(a) Empty environment



(b) Environment with an obstacle

To verify that load constraints were being considered explicitly, the torques commanded by the MPC algorithm during the trajectory generation step were compared to the torques experienced by the joints during trajectory execution. Figure 3 shows these comparisons for each joint. Joints are numbered 0 to 5 in order from the base of the robot to the end effector. Joint torques were not measured directly, as the UR3 did not have torque sensors on its joints. The joint positions, velocities, and accelerations were measured, and the joint torques were calculated using equation 5.

IV. Conclusion

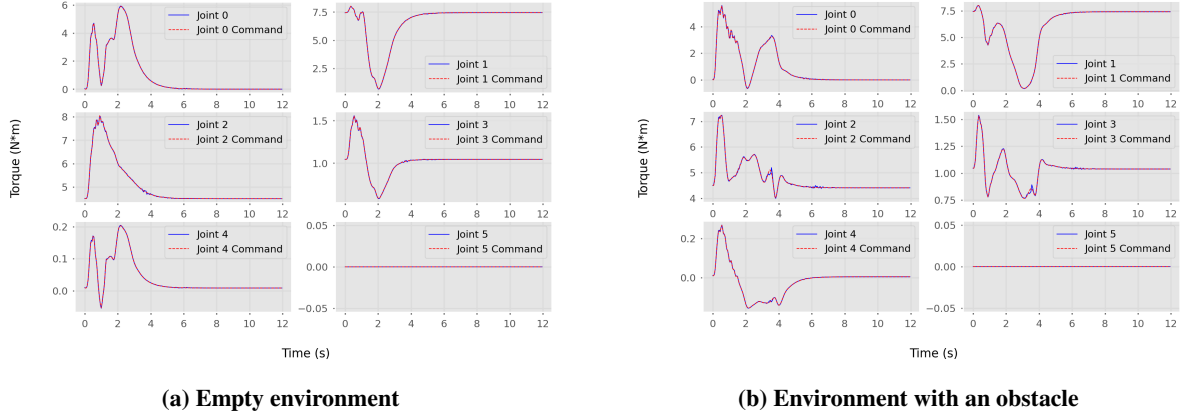
This work presents an initial study into the use of MPC as a method for handling load constraints in trajectory generation problems. The results shown in figure 3 demonstrate that the UR3 experienced joint torques very close to those that were commanded during the a-priori trajectory generation process. From this we can conclude that a manipulator following a trajectory generated by MPC will obey joint torque constraints that the MPC controller was subject to, even when a different controller is used to track the trajectory. For manipulators carrying large loads close to their operational limits, this trajectory generation method can be used to ensure that load constraints are not violated.

Future work will implement a purpose-built solver for MPC to enable faster convergence times. The algorithm will be implemented and tested in current ISAM projects at LaRC, which will provide data on how the algorithm works on manipulators carrying heavy loads and operating close to their operational limits, as well as in cluttered environments.

Acknowledgments

The work in this paper was performed as part of Langley Research Center's Precision Assembled Space Structure (PASS) project, and funded by NASA's Game Changing Developments (GCD) program.

Fig. 3 Planned joint torques vs real joint torques



References

- [1] Kunz, T., and Stilman, M., “Time-Optimal Trajectory Generation for Path Following with Bounded Acceleration and Velocity,” *Robotics: Science and Systems*, 2012, pp. 9–13.
- [2] Karaman, S., and Frazzoli, E., “Incremental Sampling-based Algorithms for Optimal Motion Planning,” 2010.
- [3] LaValle, S. M., et al., “Rapidly-exploring random trees: A new tool for path planning,” 1998.
- [4] Cooper, J. R., Loudermilk, D. C., Quartaro, A. M., Puig-Navarro, J., Henshaw, C. G., and Allen, B. D., “Clutter adaptive trajectory generation,” *Frontiers Robotics and AI*, 2023.
- [5] Cooper, J. R., and Allen, B. D., “Autonomous vehicle allocation based on clutter assessment,” 2021.
- [6] Walker, M., and Orin, D., “Efficient dynamic computer simulation of robotic mechanisms,” *Journal of Dynamic Systems, Measurement, and Control*, Vol. 104, 1982, pp. 205–211.
- [7] Denavit, J., and Hartenberg, R., “A kinematic notation for lower-pair mechanisms base on matrices,” *Journal of Applied Mechanics*, Vol. 22, 1955, pp. 215–221.
- [8] Corke, P., and Haviland, J., “Not your grandmother’s toolbox—the Robotics Toolbox reinvented for Python,” *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 11357–11363.
- [9] Coumans, E., and Bai, Y., “PyBullet, a Python module for physics simulation for games, robotics and machine learning,” <http://pybullet.org>, 2016–2021.
- [10] Fiedler, F., Karg, B., Lüken, L., Bradner, D., Heinlein, M., Brabender, F., and Lucia, S., “do-mpc: Towards FAIR nonlinear and robust model predictive control,” *Control Engineering Practice*, 2023, p. 105676.