The Journal of Open Source Software

constrained_manipulability: A ROS 2 library to Compute and Visualize Constrained Capacities for Robotic Manipulators

Mark Zolotas [1*¶], **Philip Long** [2*], **Keerthi Sagar** [3], **and Taskin Padir** [1,4]

1 Northeastern University, USA (at the time of this work) **2** Atlantic Technological University, Ireland **3** Irish Manufacturing Research Limited, Mullingar, Ireland **4** Amazon Robotics, USA ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.07481

Software

- Review
- Repository
- Archive

Editor: Chris Vernon

Reviewers:

- @askuric
- @JHartzer

Submitted: 15 August 2024
Published: 22 April 2025

Summary

This paper presents `constrained_manipulability`, a C++ library to compute and visualize a robot manipulator's constrained motion capacities. Manipulability polytopes provide a geometrical tool to evaluate the volume of free space surrounding the robotic arm when considering both environmental and intrinsic robot constraints, such as collisions and joint limits, respectively. Moreover, the polytopes defined by these convex constraints represent feasible configurations for the whole robot, that can be utilized in inverse kinematics (IK) optimization algorithms to produce collision-free motion trajectories.

The library is encapsulated in a Robot Operating System (ROS) package (Quigley et al., 2009). We include ROS 1 and ROS 2 (Macenski et al., 2022) implementations of the core C++ library. The `constrained_manipulability` package also heavily depends on our `robot_collision_checking` package (Zolotas et al., 2025), a ROS interface for collision checking via the Flexible Collision Library (FCL) (Pan et al., 2012).

The main program of the `constrained_manipulability` package reads a Unified Robot Description Format (URDF) kinematic chain from a user-defined root of the robot (e.g., the robot's base frame) to a tip (e.g., the end-effector or tool). Joint position and velocity limits are also read from the URDF, while a collision world is maintained to express environmental constraints on the robot's motion. A variety of collision objects can be added or removed using common ROS message types, such as OctoMaps (Hornung et al., 2013), mesh files, or solid primitives. These joint limit and obstacle constraints are then stacked together to define different polytopes, such as the allowable motion polytope (Long et al., 2019) and the constrained velocity polytope (Long & Padır, 2018).

Polytopes computed by `constrained_manipulability` are published using ROS messages from the `visualization_msgs::msg` namespace. The primary type is `MarkerArray`, which utilizes `Marker::TRIANGLE_LIST` markers for facets and `Marker::SPHERE_LIST` markers for vertices. As a result, the manipulability polytopes calculated by our package for a given robot manipulator can also be visualized using standard RViz (Kam et al., 2015) tools. Additionally, the generated polytopes are published in vertex and hyperplane form, which is easily interpretable by third-party libraries.

Statement of Need

Generating complex constraint geometries efficiently is a significant challenge for real-time robot manipulation. For applications, like motion planning or remote teleoperation, being able to represent the space of obstacle-free allowable motion around an end-effector is vital.

Given such a representation a collision-free IK solution can be obtained to maximize the free space of a robot moving in a constrained environment. Manipulability polytopes represent manipulability capacities, e.g., the capacity for the robot to transmit velocities and forces from the configuration space to the task space. The benefit of using convex polytopes is that they capture exact velocity bounds, rather than the approximation provided by for example ellipsoids (Yoshikawa, 1984). While the aforementionned ellipsoids have been the dominant paradigm due to historic computational constraints, more efficient polytope generation methods (Sagar et al., 2023; Skuric et al., 2023) coupled with more computation availability has led to an increased usage. Furthermore, since a polytope is defined by a set of inequality constraints, additional constraints can be easily incorporated into existing polytopes, e.g., mobile robot toppling constraints (Rasheed et al., 2018), friction cones (Caron et al., 2016), and maximum danger values.

The `constrained_manipulability` ROS 2 package aims to fulfil this need for constrained robot capacity calculation by supplying the robotics community with a fast C++ implementation that computes various types of polytopes. These geometrical constructs can also be visualized for workspace analysis or to guide an operator through the Cartesian motions available due to joint limits, kinematic constraints, and obstacles in the workspace, as illustrated in Figure 1. The utility of these visualizations has proven advantageous in remote teleoperation scenarios involving virtual reality headsets (Zolotas et al., 2021). Moreover, the package interfaces with the `robot_collision_checking` package (Zolotas et al., 2025) to perform distance and collision checking between a robot manipulator and the environment.

There are currently few software libraries capable of computing robot manipulator capacities as polytopes while seamlessly interfacing with popular robotics middleware, such as ROS. Among them, the one most similar to `constrained_manipulability` is `pycapacity` (Skuric et al., 2023), which is implemented in Python rather than C++. Unlike `pycapacity`, where velocity and force polytopes are solely computed based on the intrinsic constraints of a kinematic chain, our package accounts for additional constraints, such as positional joint limits and environmental obstacles, to represent the constrained motion space.

Figure 1: Collision-free path for a robot manipulator visualized as a trajectory of polytopes generated by maximizing the volume of allowable motion around the robot's end-effector.

Polytope and Manipulator Model

A polytope, $\mathcal{P}$, can be represented as the convex hull of its vertex set, or as a volume bounded by a finite number of half-spaces, known as the $\mathcal{V}$-representation and $\mathcal{H}$-representation, which are denoted as $\mathcal{P}^V$ and $\mathcal{P}^H$, respectively. Converting between the $\mathcal{V}$ and $\mathcal{H}$ representations can be carried out in several ways, however the `constrained_manipulability` package uses the double description method (Fukuda & Prodon, 1996).

The `constrained_manipulability` provides two types of polytope representations: 1. constrained motion polytopes, which modify the classic manipulability velocity polytope (Kokkinis & Paden, 1989) to include joint limits and capacity reductions due to nearby obstacles; and 2. allowable motion polytopes, which are a linearization of their constrained counterpart used to generate a measure of free space or allowable motion in which a robot can move while satisfying all constraints.

In both cases, the polytope is constructed as follows. First, a system of linear inequalities is constructed in joint space, for instance based on joint velocity limits:

$$\mathcal{Q}^H = \begin{bmatrix} \mathbb{I}_n \\ -\mathbb{I}_n \end{bmatrix} \dot{\mathbf{q}} \leq \begin{bmatrix} \dot{\mathbf{q}}^{max} \\ -\dot{\mathbf{q}}^{min} \end{bmatrix}, \tag{1}$$

where $\dot{\mathbf{q}}^{max}$ and $\dot{\mathbf{q}}^{min}$ are the robot's maximum and minimum joint velocities. Using the double description method, an equivalent polytope in the $\mathcal{V}$-representation (i.e., defined by its

vertices) is written as:

$$\mathcal{Q}^V = \{\ \dot{\mathbf{q}}_1^v,\ \dot{\mathbf{q}}_2^v,\ \dots,\ \dot{\mathbf{q}}_q^v\ \}, \tag{2}$$

where $\dot{\mathbf{q}}_i^v$ denotes the i^{th} vertex of the polytope $\mathcal{Q}$, given q vertices in n-dimensional space. A Cartesian polytope, denoted as $\mathcal{MP}$, can then be obtained by transforming the vertices of Equation 2 to Cartesian space using the forward kinematics. $\mathcal{MP}$'s vertex set representation of p vertices in 3-dimensional space is given as:

$$\mathcal{MP}^V = \{\ \boldsymbol{\nu}_1^v,\ \dots,\ \boldsymbol{\nu}_p^v\ \} = \{\ \mathbf{J}_n\dot{\mathbf{q}}_1^v,\ \dots,\ \mathbf{J}_n\dot{\mathbf{q}}_p^v\ \}, \tag{3}$$

with $\boldsymbol{\nu}_j^v = \mathbf{J}_n\dot{\mathbf{q}}_j^v$ and $p \leq q$. The convexity of a polytope is preserved under affine transformation, thus a bounded volume of $\mathcal{MP}$ that represents the system's manipulability can easily be obtained to serve as an exact indicator of robot performance.

For constrained motion polytopes, the following constraints are supported: joint velocity limits, joint velocity damping due to positional joint limits, and joint velocity damping due to obstacle proximity to the kinematic chain. For allowable motion polytopes the following constraints are supported: positional joint limits, positional limits due to obstacle proximity to the kinematic chain, and linearization limits. Increasing the values of the linearization expands the free space virtual fixture at a cost of reduced fidelity. Our implementation uses a scalar linearization limit for all joints that can also be altered at run time, e.g., could be increased to enlarge the solution or shrunk to guide the user towards a defined goal configuration, as defined in Eq. 20 of Zolotas et al. (2021). Finally, in our implementation, we select at every instant the point along each link nearest to all environmental obstacles. Hence, the system considers the set of instantaneous collision-free joint motion for each link along the robot's kinematic chain, considering all surrounding obstacles.

A more detailed formulation of the polytopes discussed here is available in Section II.B of Long et al. (2019).

Block Diagram

Any joint position and velocity limits are extracted from the robot's URDF description via ROS 2 parameter operations. Polytopes are calculated as described above using these limits and by obtaining the minimum distance from each link on the robot to objects in the collision world. FCL (Pan et al., 2012) is required to compute these distances and is accessible via the interface package: `robot_collision_checking` (Zolotas et al., 2025). Polytopes in Cartesian space can then be returned from getter functions:

```
Polytope getConstrainedAllowableMotionPolytope(
  const sensor_msgs::msg::JointState& joint_state,
  bool show_polytope,
  Eigen::MatrixXd& AHrep,
  Eigen::VectorXd& bHrep,
  Eigen::Vector3d& offset_position,
  const std::vector<double>& color_pts,
  const std::vector<double>& color_line);
```

where `AHrep` and `bHrep` represent joint space polytope constraints.

A diagram summarizing the `constrained_manipulability` package architecture and ROS 2 communication flow is displayed in Figure 2.

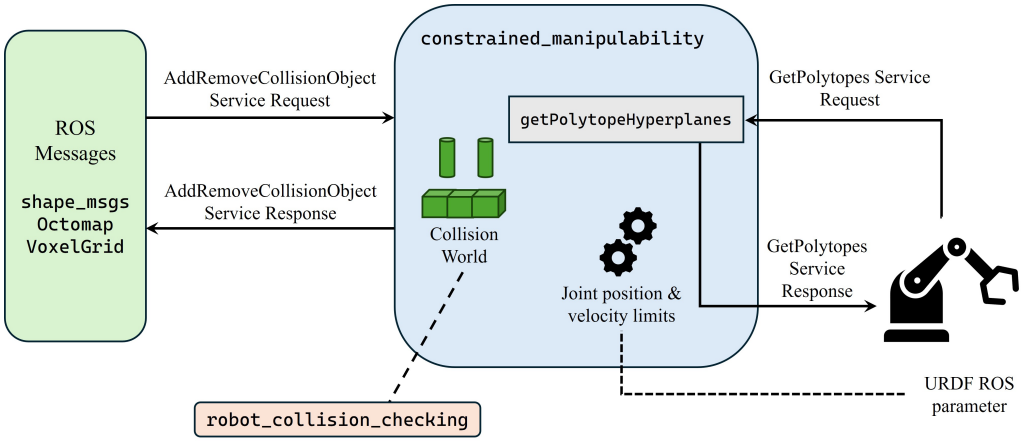

Figure 2: Block diagram for the constrained_manipulability ROS 2 package.

Future Work

The `constrained_manipulability` package will continue to be maintained as a ROS 2 library for computing and vizualizing a robot manipulator's constrained capacities as polytopes. To extend the widespread utility of this package, future work could also involve creating a standalone library that is independent of ROS. For example, by creating Python bindings for the core C++ library and its underlying algorithms. This would enable easier interfacing with other popular robotics software, such as Pinocchio (Carpentier et al., 2021).

Conflict of Interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be constructed as a potential conflict of interest.

Acknowledgements

Mark Zolotas is currently at Toyota Research Institute (TRI), Cambridge, MA, USA. This paper describes work performed at Northeastern University and is not associated with TRI.

Taskin Padir holds concurrent appointments as a Professor of Electrical and Computer Engineering at Northeastern University and as an Amazon Scholar. This paper describes work performed at Northeastern University and is not associated with Amazon.

References

Caron, S., Pham, Q.-C., & Nakamura, Y. (2016). Zmp support areas for multicontact mobility under frictional constraints. *IEEE Transactions on Robotics*, *33*(1), 67–80. https://doi.org/10.1109/tro.2016.2623338

Carpentier, J., Budhiraja, R., & Mansard, N. (2021, July). Proximal and Sparse Resolution of Constrained Dynamic Equations. *Robotics: Science and Systems 2021*. https://doi.org/10.15607/rss.2021.xvii.017

Fukuda, K., & Prodon, A. (1996). Double description method revisited. *Combinatorics and Computer Science*, *1120*, 91. https://doi.org/10.1007/3-540-61576-8_77

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, *34*, 189–206. https://doi.org/10.1007/s10514-012-9321-0

Kam, H. R., Lee, S.-H., Park, T., & Kim, C.-H. (2015). Rviz: A toolkit for real domain data visualization. *Telecommunication Systems*, *60*, 337–345. https://doi.org/10.1007/s11235-015-0034-5

Kokkinis, T., & Paden, B. (1989). Kinetostatic performance limits of cooperating robot manipulators using force-velocity polytopes. *Proceedings of the ASME Winter Annual Meeting*, 151–155.

Long, P., Keleştemur, T., Önol, A. Ö., & Padir, T. (2019). Optimization-based human-in-the-loop manipulation using joint space polytopes. *International Conference on Robotics and Automation*, 204–210. https://doi.org/10.1109/ICRA.2019.8794071

Long, P., & Padır, T. (2018). Evaluating robot manipulability in constrained environments by velocity polytope reduction. *IEEE-RAS International Conference on Humanoid Robots*, 497–502. https://doi.org/10.1109/HUMANOIDS.2018.8624962

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, *7*(66), eabm6074. https://doi.org/10.1126/scirobotics.abm6074

Pan, J., Chitta, S., & Manocha, D. (2012). FCL: A general purpose library for collision and proximity queries. *IEEE International Conference on Robotics and Automation*, 3859–3866. https://doi.org/10.1109/ICRA.2012.6225337

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. (2009). ROS: An open-source robot operating system. *ICRA Workshop on Open Source Software*, *3*.

Rasheed, T., Long, P., Marquez-Gamez, D., & Caro, S. (2018). Tension distribution algorithm for planar mobile cable-driven parallel robots. *Cable-Driven Parallel Robots: Proceedings of the Third International Conference on Cable-Driven Parallel Robots*, 268–279. https://doi.org/10.1007/978-3-319-61431-1_23

Sagar, K., Caro, S., Padır, T., & Long, P. (2023). Polytope-based continuous scalar performance measure with analytical gradient for effective robot manipulation. *IEEE Robotics and Automation Letters*, *8*(11), 7289–7296. https://doi.org/10.1109/LRA.2023.3313926

Skuric, A., Padois, V., & Daney, D. (2023). Pycapacity: A real-time task-space capacity calculation package for robotics and biomechanics. *Journal of Open Source Software*, *8*(89), 5670. https://doi.org/10.21105/joss.05670

Yoshikawa, T. (1984). Analysis and control of robot manipulators with redundancy. *International Symposium on Robotics Research*, 735–747.

Zolotas, M., Long, P., & Padir, T. (2025). Robot_collision_checking: A lightweight ROS 2 interface to FCL (flexible collision library). *Journal of Open Source Software*, *10*(105), 7473. https://doi.org/10.21105/joss.07473

Zolotas, M., Wonsick, M., Long, P., & Padır, T. (2021). Motion Polytopes in Virtual Reality for Shared Control in Remote Manipulation Applications. *Frontiers in Robotics and AI*, *8*, 286. https://doi.org/10.3389/frobt.2021.730433