TriP: A Python package for the kinematic modeling of serial-parallel hybrid robots

Jan Baumgärtner[*1] **and Torben Miller**[†2]

1 Heidelberg University **2** Independent Researcher

Summary

Robots can be classified according to their mechanical structure. Serial mechanisms like robotic arms are mechanisms where each moving part (called a link) is connected to only the one before and the one after it. They are often used when a large workspace is required, meaning the robot needs a long reach. In parallel mechanisms, the links of the robot form loops causing them to be structurally stronger and stiffer.

If both a large workspace and structural strength are required, hybrids between serial and parallel mechanisms are used. These mechanisms contain both serial and parallel mechanisms. While hybrid mechanisms combine the mechanical advantages of both parallel and serial mechanisms they also combine their modeling disadvantages:

Finding an explicit solution for either forward or inverse kinematics is often impossible. Using numerical approaches instead leads to complicated constrained optimization problems for both forward and inverse kinematics.

While serial mechanisms are very well supported by current robotic frameworks, parallel mechanisms and hybrid mechanisms especially are often not supported at all. A great overview of the supported robot types for different robotic frameworks was compiled by Kumar (2019).

TriP is a python package designed to close this gap using a modular modeling framework akin to the one described by Kumar et al. (2020). It allows the modeling of arbitrary hybrid mechanisms and is capable of calculating forward and inverse kinematics.

The calculations are performed using a symbolic framework. This makes it easy for users to implement custom case-dependent mathematical solvers.

Statement of Need

While a huge number of researchers use hybrid serial parallel systems such as Pisla et al. (2013), Kanaan et al. (2009), or Zoss et al. (2006), most modern kinematics frameworks still lack support. Examples include openrave (Diankov & Kuffner, 2008) used in the moveit stack (Chitta et al., 2012) or the matlab robotics toolbox (Corke, 2017) which only supports serial mechanisms. This lack of support often leaves developers with essentially two choices:

Either shoehorn their hybrid robots into a framework not designed to handle them or be left to implement their own kinematic solvers.

TriP is a lightweight and easy-to-use package directly modeling hybrid mechanisms and calculating their kinematics. Although TriP is fast for a python package it is not built for robust hard-real time control applications. Instead, it is aimed at researchers and engineers who quickly want to build kinematic models to test their mechanical designs.

---

[*]co-first author
[†]co-first author

Overview

TriP models robots using its *Robot* class. A *Robot* object is made up of *Transformation* and *KinematicGroup* objects. The *KinematicGroup* objects are used to model parallel mechanisms while the *Transformation* objects model serial mechanisms. See Figure 1 for reference. Here the links of each robot are colored according to the *KinematicGroup* or *Transformation* it belongs to, while joints are either light green or orange, depending on whether they are actuated or not.
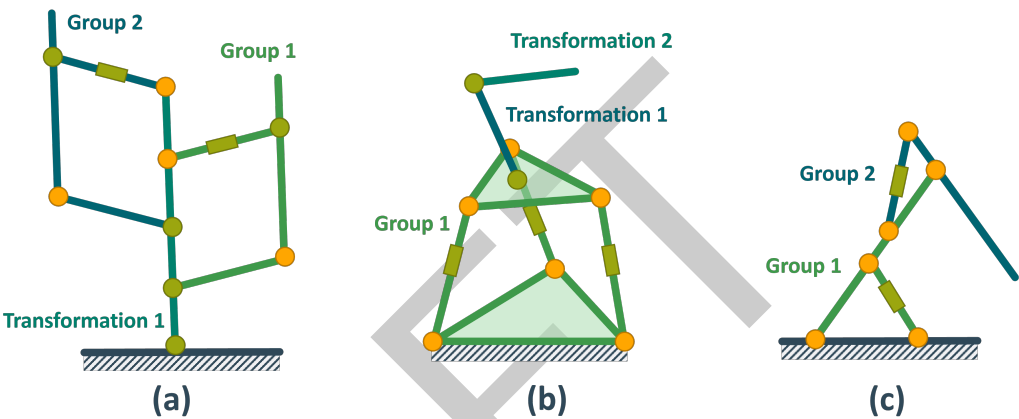

Figure 1: Different Hybrid Robot types and their object structure

Both *KinematicGroups* and *Transformations* can be connected to form branching mechanisms as indicated in Figure 1.

Transformation objects implement homogeneous coordinate transformations between coordinate frames. These can be either dynamic or static with dynamic transformations implementing joints. A few example joints can be seen in Figure 2.

	Transformation(name = "Revolute Joint", values = { 'rz': 0}, state_variables = ['rz'])
	Transformation(name = „Spherical Joint", values = {'rx': 0, 'ry': 0, 'rz': 0}, state_variables = ['rx', 'ry', 'rz'])
	Transformation(name = „Prismatic Joint", values = { 'tz': 0}, state_variables = [,tz'])
	Transformation(name = „Planar Slider Joint", values = {,tx': 0, 'ty': 0, 'rz': 0}, state_variables = ['tx', 'ty', 'rz'])

Figure 2: Sample Joints using the Transformation class

50 While all example joints use Euler angles in roll pitch yaw convention to describe rotation,
51 quaternions are also supported.

52 *KinematicGroups* model parallel mechanisms using the abstraction approach described by Kumar et al. (2020) . This approach models a parallel manipulator as a virtual serial manipulator
53 and a mapping that maps the virtual joint state to the true actuated joint state of the parallel
54 and a mapping that maps the virtual joint state to the true actuated joint state of the parallel
55 manipulator.

56 An illustrative example of this model is an excavator with two hydraulic cylinders. Each
57 cylinder is part of a parallel mechanism resulting in two *KinematicGroups*. Both can be seen
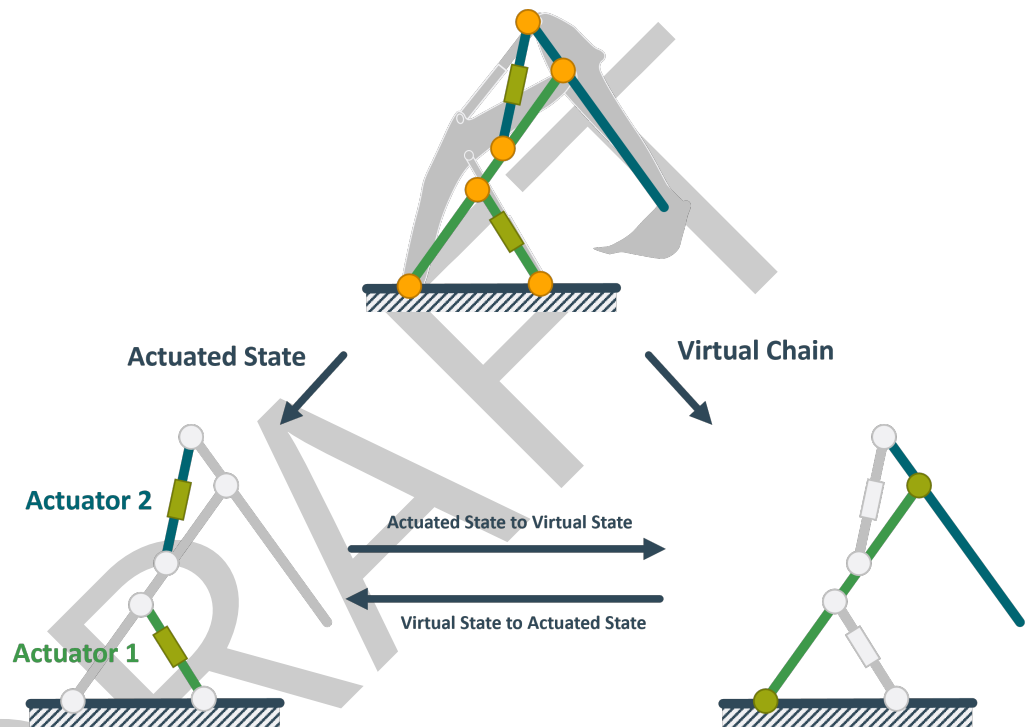58 in Figure 3 where one is green and the other one is blue.

Figure 3: Excavator Arm build from two Groups (green and blue)

59 The abstraction approach models the excavator as a serial manipulator where the joints are
60 directly actuated. Using two mappings to convert the state of the hydraulic cylinders to
61 the state of the joints and vice versa it is possible to calculate both forward and inverse
62 kinematics. In this example, the mapping between cylinders and joints can be expressed using
63 trigonometry. Since an explicit formulation of the mappings might not always be possible,
64 TriP can also compute the mapping by solving the closure equation of the parallel manipulator.

65 TriP can generate symbolic representations of robots using casadi (Andersson et al., 2019).
66 This symbolic representation can be used to set up a solver object that then solves the inverse
67 kinematics. While the library already implements a simple inverse kinematics solver, the
68 symbolic representation makes it easy to implement custom solvers.

69 All features of TriP are thoroughly documented with tutorials and examples to help people
70 get started.

References

72 Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., & Diehl, M. (2019). CasADi

– A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, *11*(1), 1–36. https://doi.org/10.1007/s12532-018-0139-4

Chitta, S., Sucan, I., & Cousins, S. (2012). Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, *19*(1), 18–19. https://doi.org/10.1109/MRA.2011.2181749

Corke, P. I. (2017). *Robotics, vision & control: Fundamental algorithms in MATLAB* (Second). Springer.

Diankov, R., & Kuffner, J. J. (2008). *OpenRAVE: A planning architecture for autonomous robotics*.

Kanaan, D., Wenger, P., & Chablat, D. (2009). Kinematic analysis of a serial–parallel machine tool: The VERNE machine. *Mechanism and Machine Theory*, *44*(2), 487–498. https://doi.org/10.1016/j.mechmachtheory.2008.03.002

Kumar, S. (2019). *Modular and analytical methods for solving kinematics and dynamics of series-parallel hybrid robots* [PhD thesis]. Universität Bremen.

Kumar, S., Woehrle, H., Fernández, J., Mueller, A., & Kirchner, F. (2020). A survey on modularity and distributivity in series-parallel hybrid robots. *Mechatronics*, *68*. https://doi.org/10.1016/j.mechatronics.2020.102367

Pisla, D., Szilaghyi, A., Vaida, C., & Plitea, N. (2013). Kinematics and workspace modeling of a new hybrid robot used in minimally invasive surgery. *Robotics and Computer-Integrated Manufacturing*, *29*(2), 463–474. https://doi.org/10.1016/j.rcim.2012.09.016

Zoss, A. B., Kazerooni, H., & Chu, A. (2006). Biomechanical design of the berkeley lower extremity exoskeleton (BLEEX). *Mechatronics, IEEE/ASME Transactions on*, *11*, 128–138. https://doi.org/10.1109/TMECH.2006.871087