Task-Level Motion Planning for Multi-Manipulator System

(Rajendra Singh, final year, Computer Science and Engineering)

28 February 2020

BTP PRESENTATION

IIT PALAKKAD

Indian Institute of Technology Palakkad
भारतीय प्रौद्योगिकी संस्थान पालक्काड
Under Ministry of Human Resource Development, Govt. of India
मानव संसाधन विकास मंत्रालय के अधीन, भारत सरकार

GadgEon
Engineering Smartness

Acknowledgement

I sincerely thank you everyone for their guidance who helped me with this project.

Mentor

[Dr. Chandra Shekar Lakshminarayanan]

Reinforcement learning, Department of Computer Science And Engineering, IIT Palakkad

Guide

[Dr. Santhakumar Mohan]

Robotics and Control, Department of Mechanical Engineering, IIT Palakkad

External guide

[Mr. Girish Kumar]

Director, Innovation Labs, Robotics & Computer Vision, GadgEon Smart Systems, Kochi

Panelist

[Dr. Deepak Rajendraprasad]

Combinatorics and Algorithms, Department of Computer Science And Engineering, IIT Palakkad

Panelist

[Dr. Satyajit Das]

AI On Edge, Department of Computer Science And Engineering, IIT Palakkad

Coordinator

[Dr. Albert Sunny]

Computer Networks, Department of Computer Science And Engineering, IIT Palakkad

TABLE OF CONTENT

1. Motivation……………………..1
2. Problem Statement……......3
3. Previous Relevant Work...5
4. Implementation…………....6
5. Conclusion………………...15

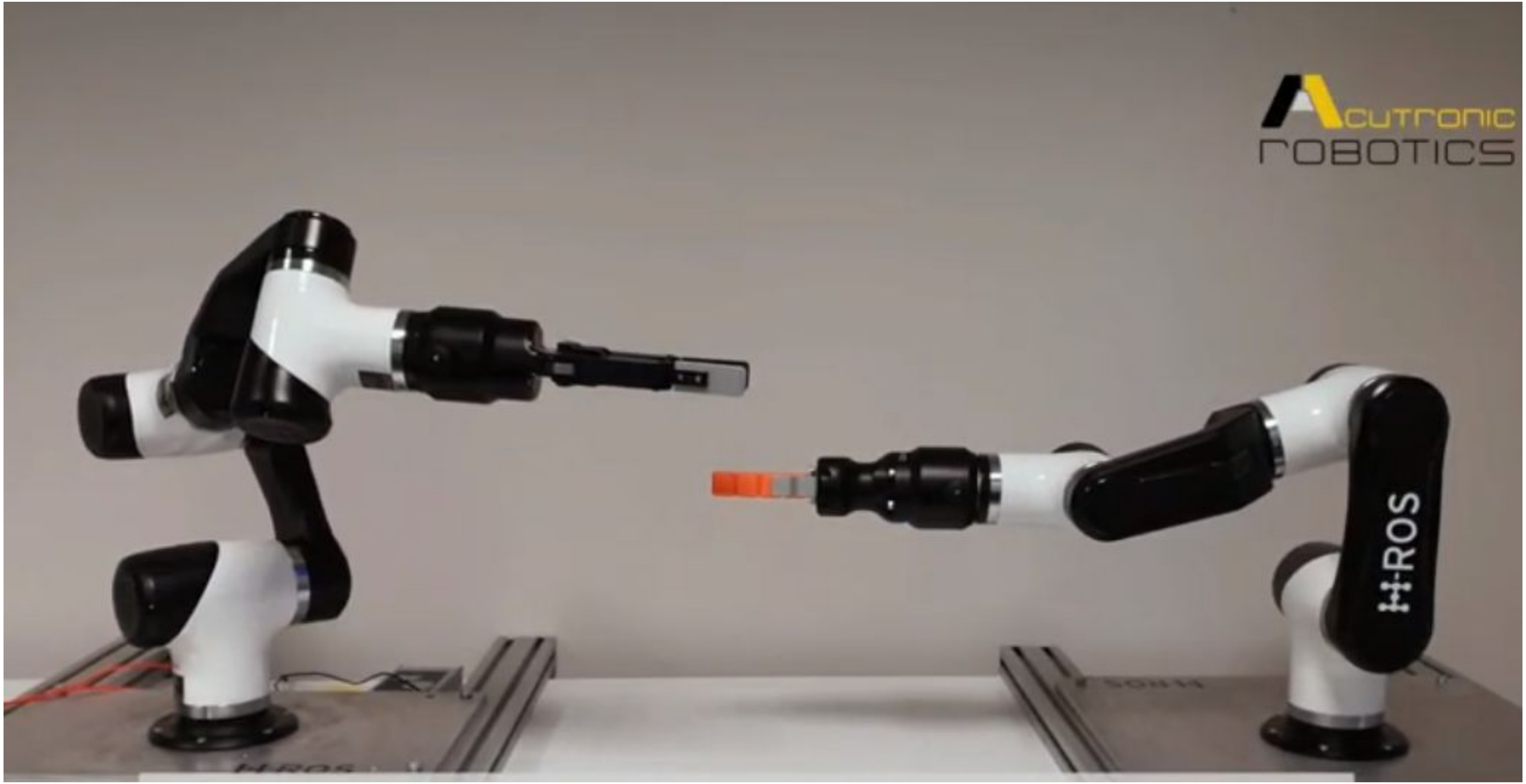
Motivation

Nasa's Robonaut Mission

DARPA Robotics Challenge(DRC)

Reference: https://robonaut.jsc.nasa.gov/R2/

Reference: https://www.darpa.mil/program/darpa-robotics-challenge



Video: https://youtu.be/zGM2lYAHrm4

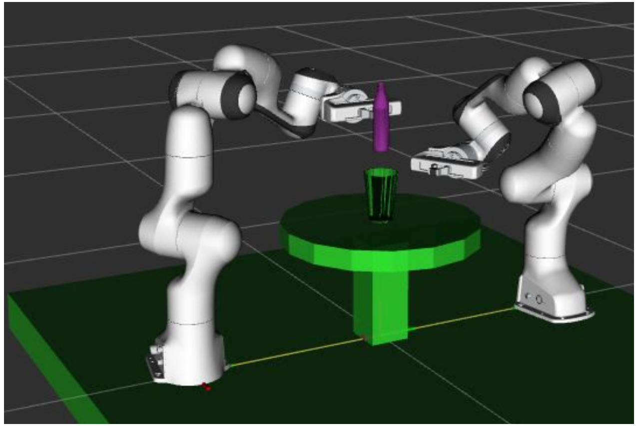
Problem Statement

Problem Statement

Perform complex manipulation task like pick and place, building structures and pouring in multi-manipulator system.

Subtask/Workflow :

1. Simple **Joint** space planning(**move group**)
2. Simple **Cartesian** space planning(move group)
3. **Pick Place** Task (move group)
4. Simple Joint space planning(MoveIt Task Constructor - **MTC**)
5. Simple Cartesian space planning(MTC)
6. Pick Place Task(MTC)
7. **Multi arm** simple Joint space planning
8. Multi arm simple Cartesian space planning
9. Multi arm Simple Pick Place Task(own work)
10. Multi arm Complex Pick Place Task(IIT)
11. Multi arm planning using Serial container
12. Multi arm planning using Parallel container
    12.1 Alternative
    12.2 Fallback
    12.3 Merger
13. Multiple task
14. Single arm pouring task
15. Complex Multi arm pouring
16. Complex Multi arm pouring task with stages intermixing
17. **Complex pouring task using multiple arm with orientation constraint imposed**

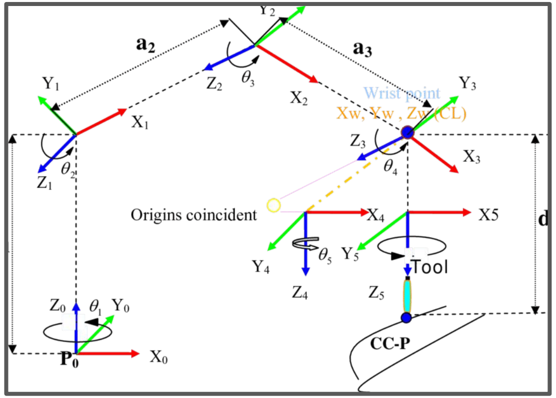


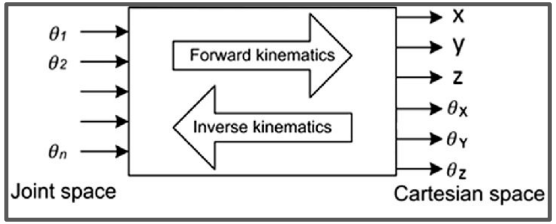
Tools:

1. Panda arm(7 dof arm)
2. Robot operating System(ROS)
3. Motion Planning framework, moveit
4. Moveit_task_constructor(MTC)
5. Open Motion Planning Library(OMPL)

Relevant Work

Inverse Kinematics (Newton methods)

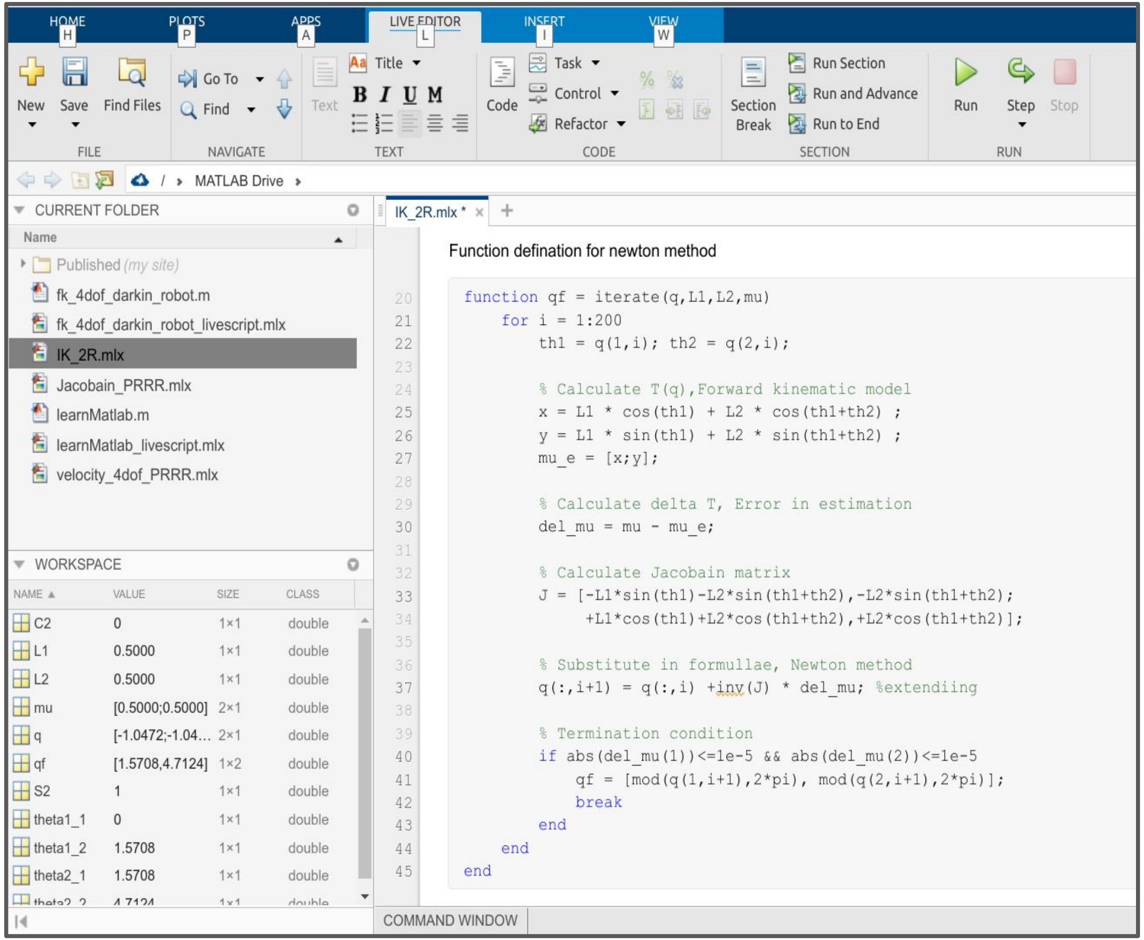




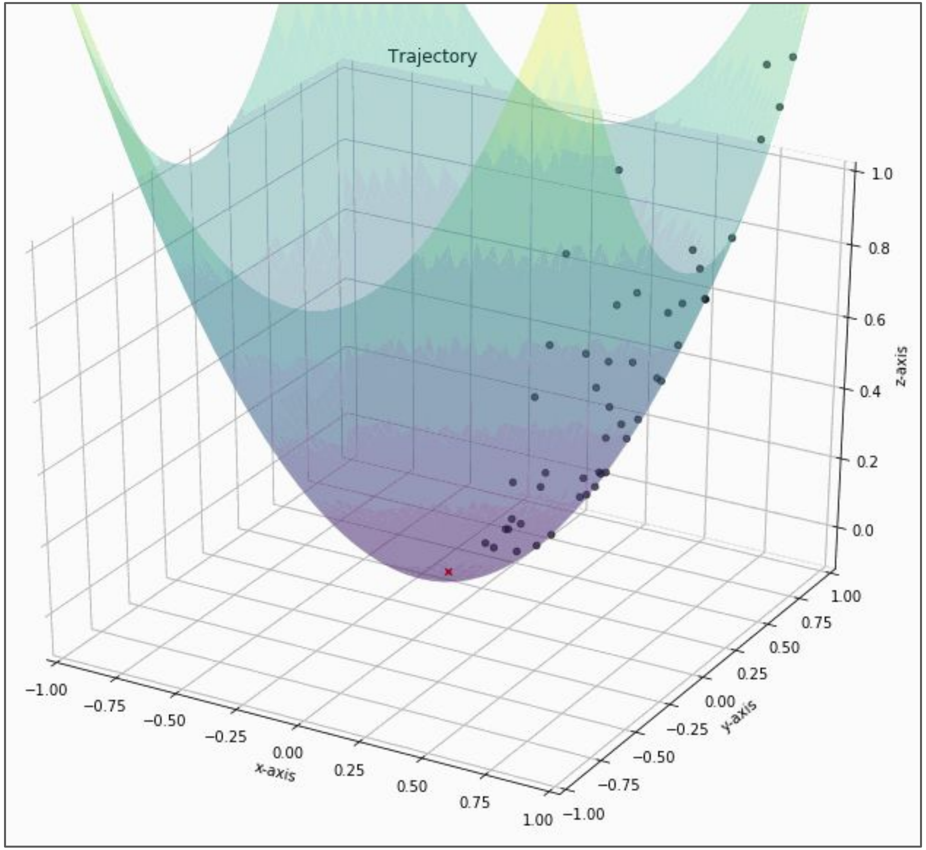
$$\cos q_2 = \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2}$$

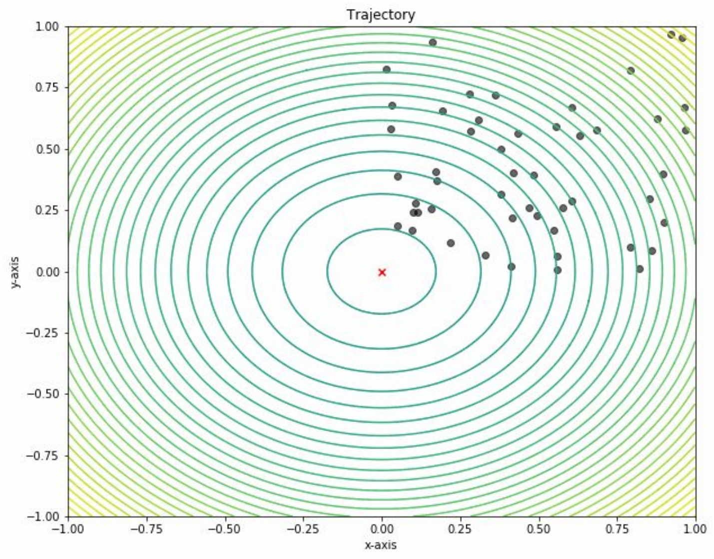
$$q_2 = \cos^{-1}\frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2}$$

$$q_1 = \tan^{-1}\frac{y}{x} - \tan^{-1}\frac{a_2 \sin q_2}{a_1 + a_2 \cos q_2}$$



Particle swarm optimization (PSO)





velocity of particle i at time k+1 →

$$\mathbf{v}_{k+1}^{i} = \underbrace{w\,\mathbf{v}_{k}^{i}}_{\text{current motion}} + c_1\, rand\, \underbrace{\frac{\left(\mathbf{p}^{i} - \mathbf{x}_{k}^{i}\right)}{\Delta t}}_{\text{particle memory influence}} + c_2\, rand\, \underbrace{\frac{\left(\mathbf{p}_{k}^{g} - \mathbf{x}_{k}^{i}\right)}{\Delta t}}_{\text{swarm influence}}$$

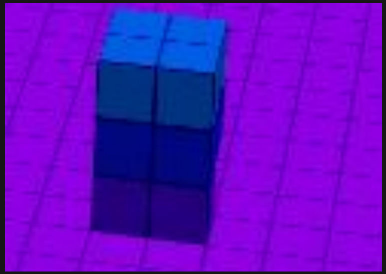
inertia factor
range: 0.4 to 1.4

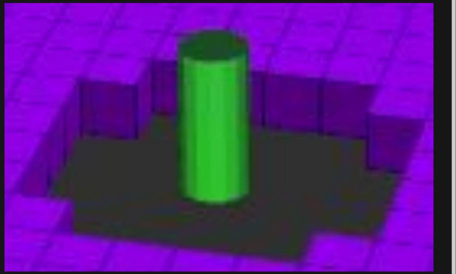
self confidence
range: 1.5 to 2

swarm confidence
range: 2 to 2.5

Perception

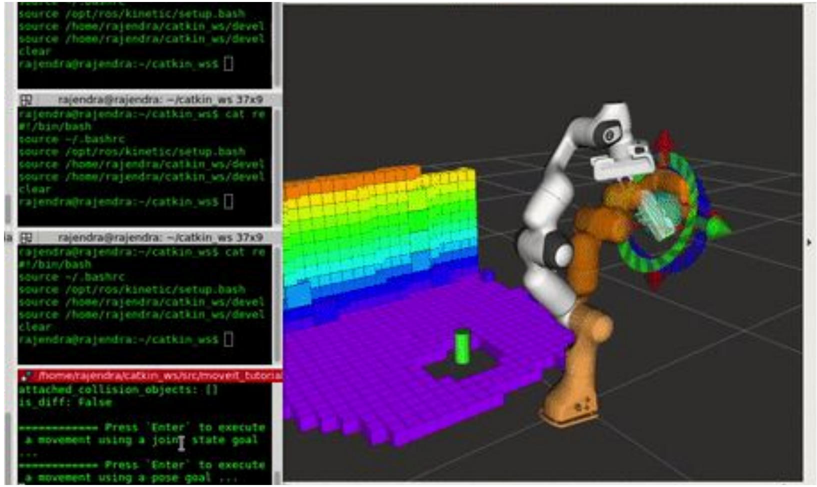
- Converting pointcloud to pcl:PointXYZRGB
- PassThroughFilter
- Compute the point normals
- Detect and eliminate the plane
- Extracting plane normals
- Extract the cylinder
- Compute cylinder_params





Pick and place stack

- Add the collision object and cloud
- Declare the gripper and arm group
- Declare the pre-grasp, grasp and post-grasp approaches
- Chose the planner
- Plan and execute the trajectory



High level planning with Moveit on real robot

Planning group

- **arm**
  - *Joints*
    - Joint1 - Revolute
    - Joint2 - Revolute
    - Joint3 - Revolute
  - *Links*
    - Link1
    - Link2
    - Base
    - Link3
    - Link8
  - *Chain*
    - Base -> Link3
  - *Subgroups*

Pointcloud

Point Cloud

Point Cloud

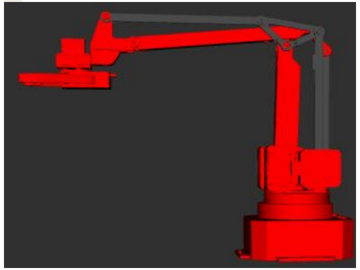
Point Cloud Topic:	/camera/depth/color/points
Max Range:	2.0
Point Subsample:	1
Padding Offset:	0.1
Padding Scale:	1.0
Filtered Cloud Topic:	filtered_cloud
Max Update Rate:	1.0

Collision matrix

	Base	Link1	Link2	Link3	Link8	Link9	Link4	Link5	Link6	Link7
Base		✓								
Link1	✓		✓				✓	✓	✓	
Link2		✓		✓			✓			
Link3			✓		✓	✓	✓	✓		✓
Link8				✓		✓				
Link9				✓	✓		✓		✓	✓
Link4		✓	✓	✓		✓		✓	✓	✓
Link5		✓		✓			✓		✓	✓
Link6		✓				✓	✓	✓		✓
Link7				✓		✓	✓	✓	✓	

Known pose

	Pose Name	Group Name
1	home	arm
2	rest	arm



Virtual Joint

Virtual Joint Name: virtual joint

Child Link: Base

Parent Frame Name: world

Joint Type: fixed

Passive joint

	Joint Names
1	Joint8
2	Joint9
3	Joint4
4	Joint5
5	Joint6
6	Joint7

Inverse Kinematics

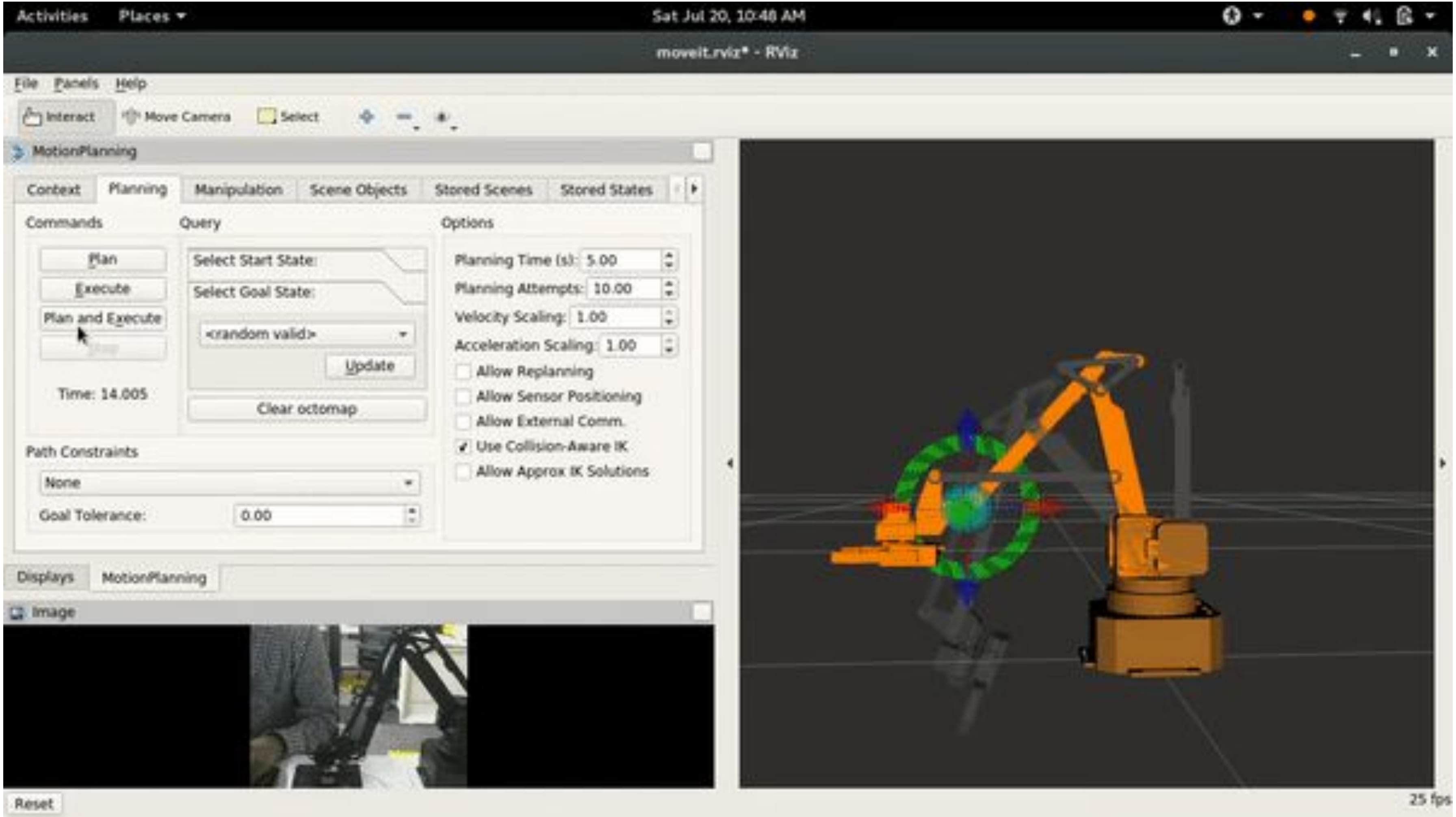
Kinematics

Group Name:	arm
Kinematic Solver:	trac_ik_kinematics_plugin/TRAC_IKKinematicsPlugin
Kin. Search Resolution:	0.005
Kin. Search Timeout (sec):	0.05
Kin. Solver Attempts:	3

OMPL Planning

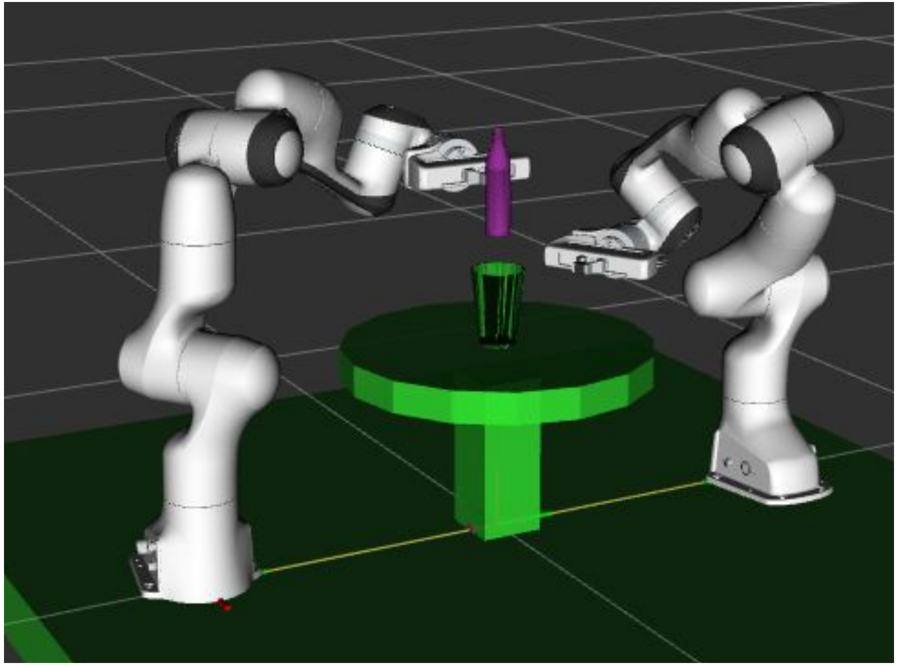
Group Default Planner: RRTstar

Issues with uarm

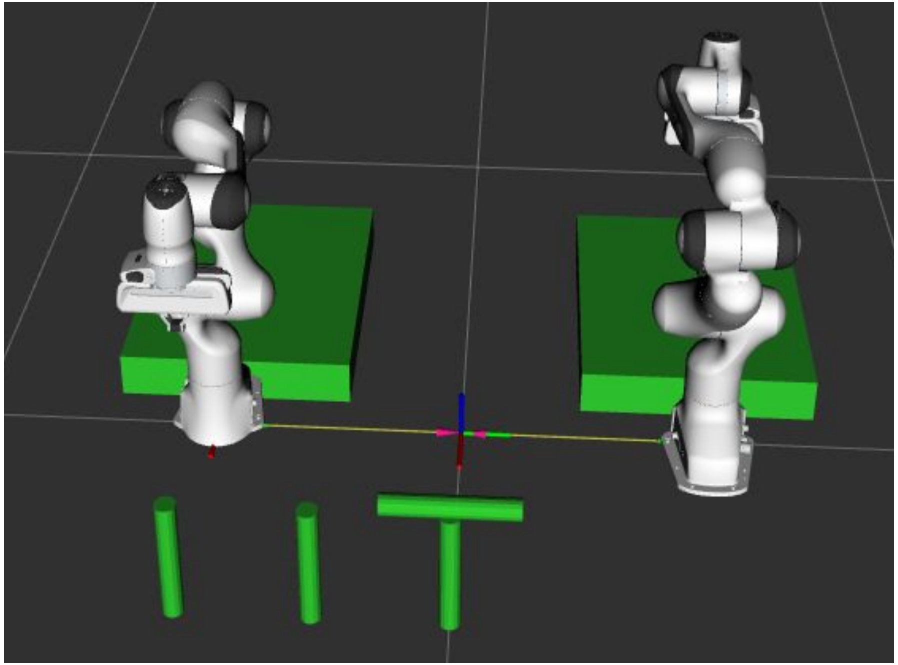


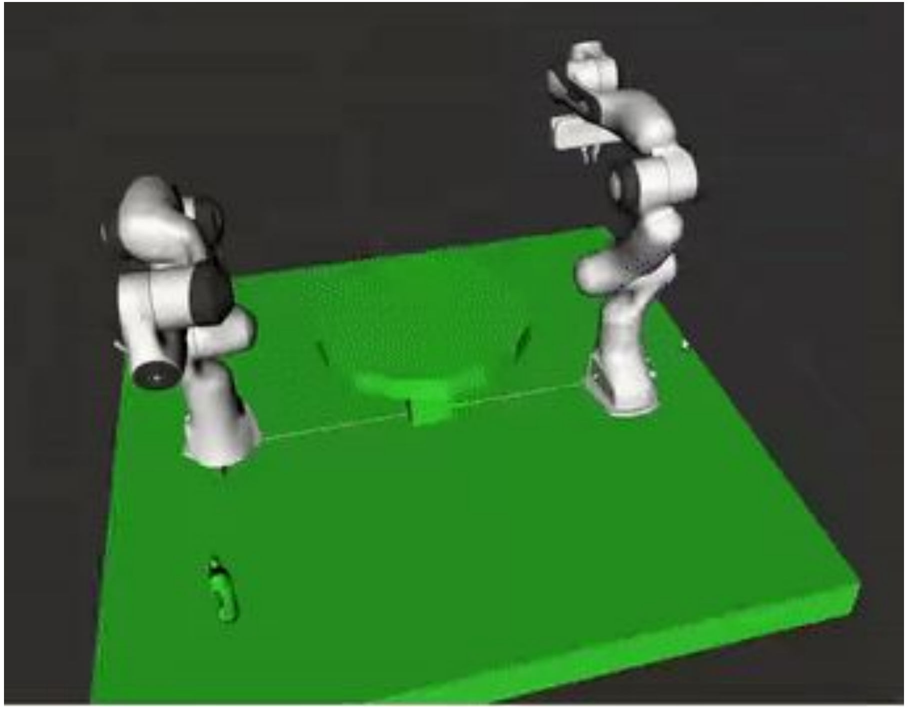
Implementation

Pouring Task

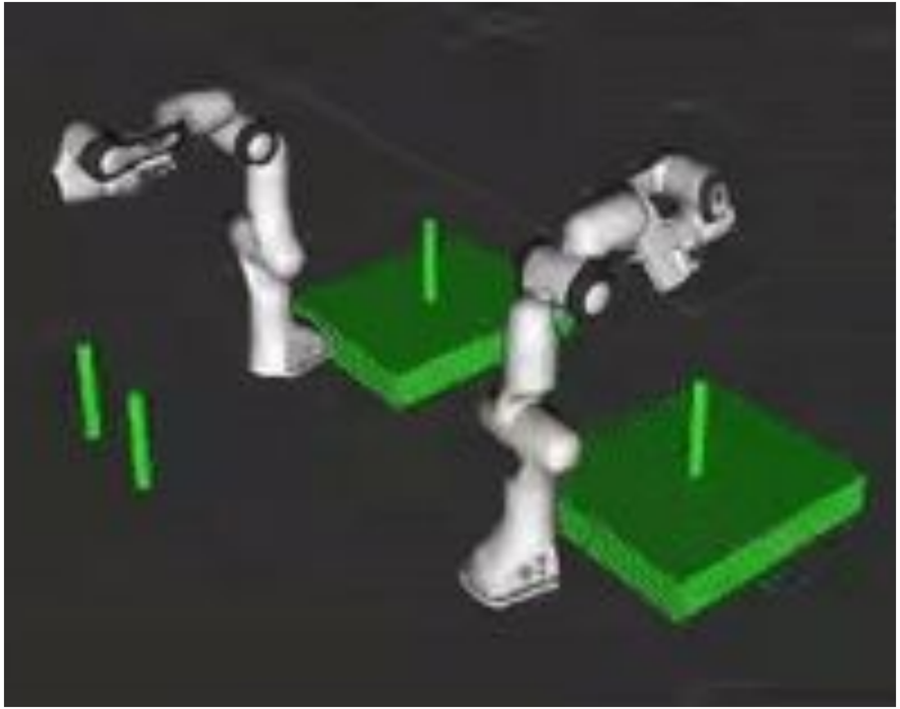


Creating Structure

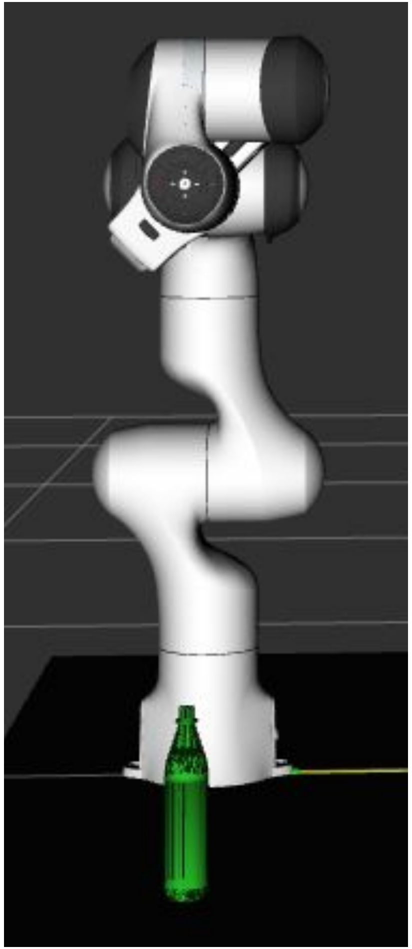




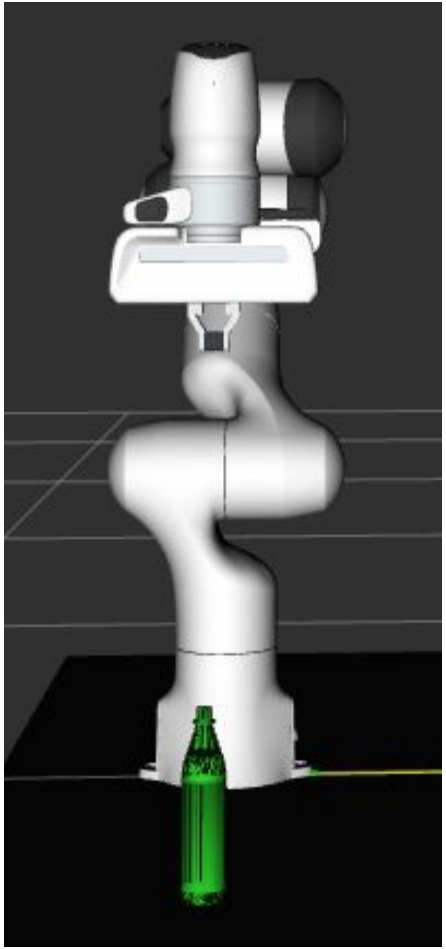
Full Video - https://youtu.be/tS2U0AX3r_M



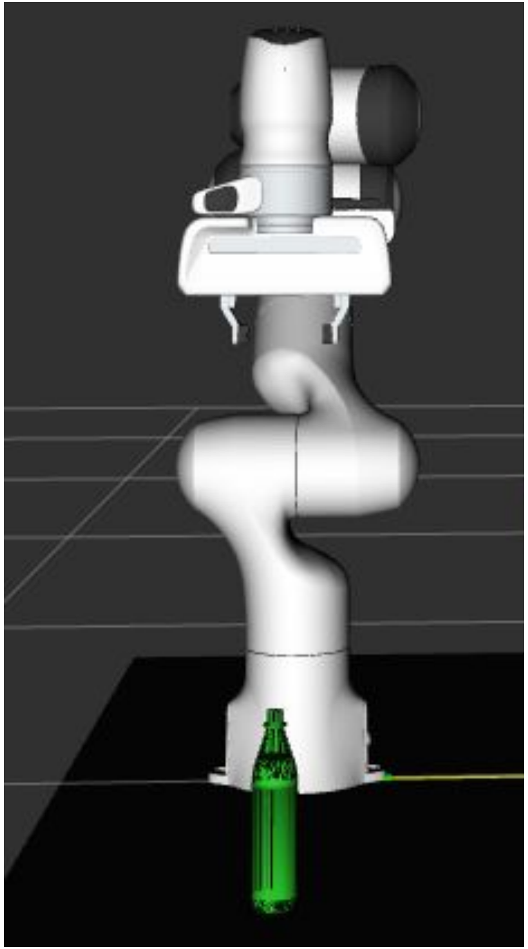
Full Video - https://youtu.be/K7N7RMx9Q88



1. Current State



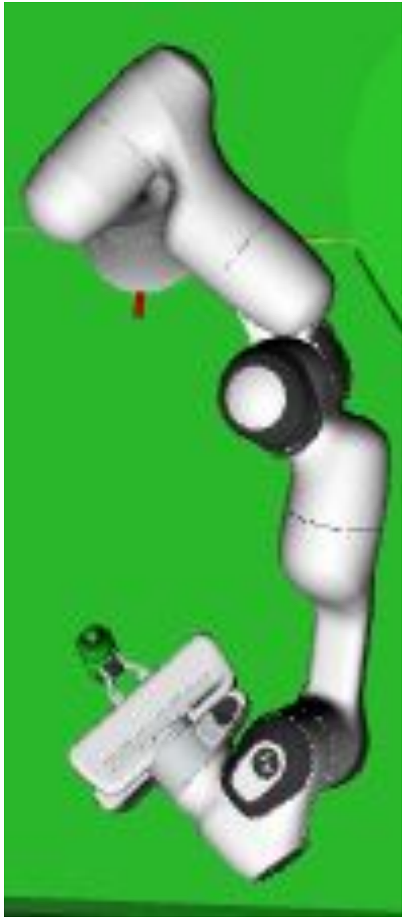
2. MoveTo Home



3. Open Hand

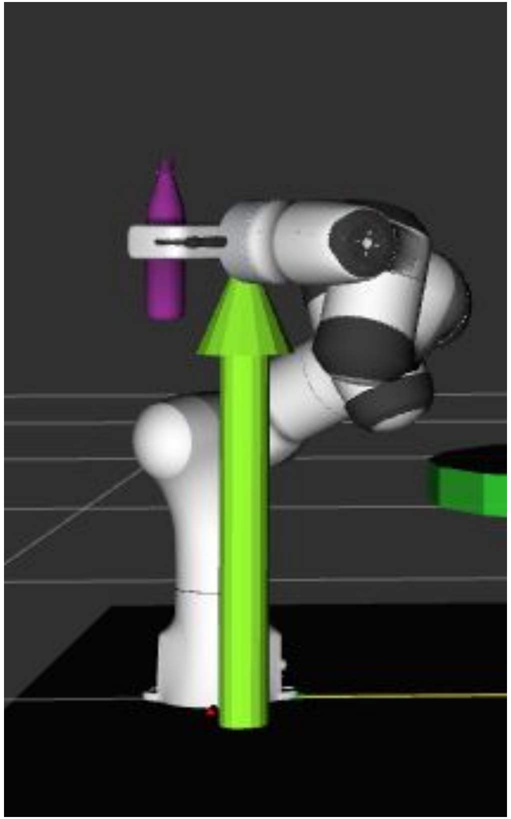
4. MoveTo Pick

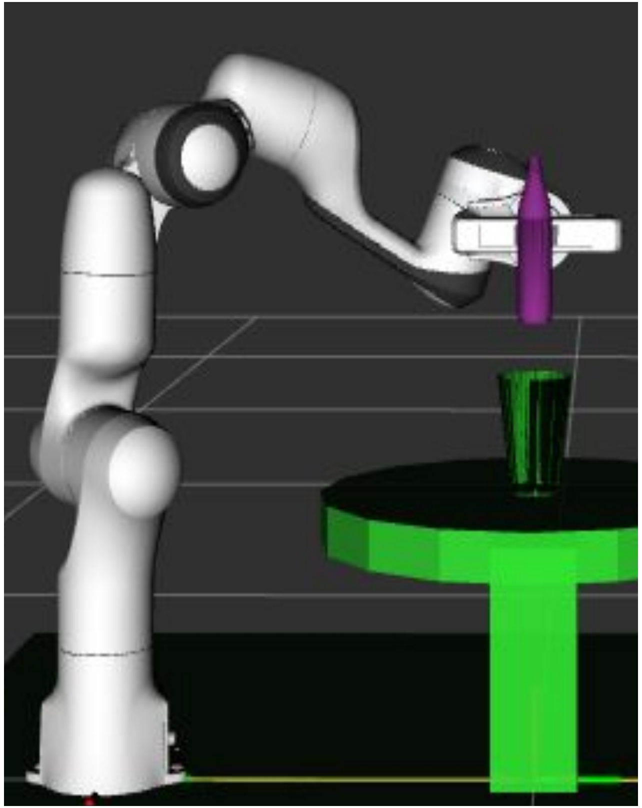
5. Approach

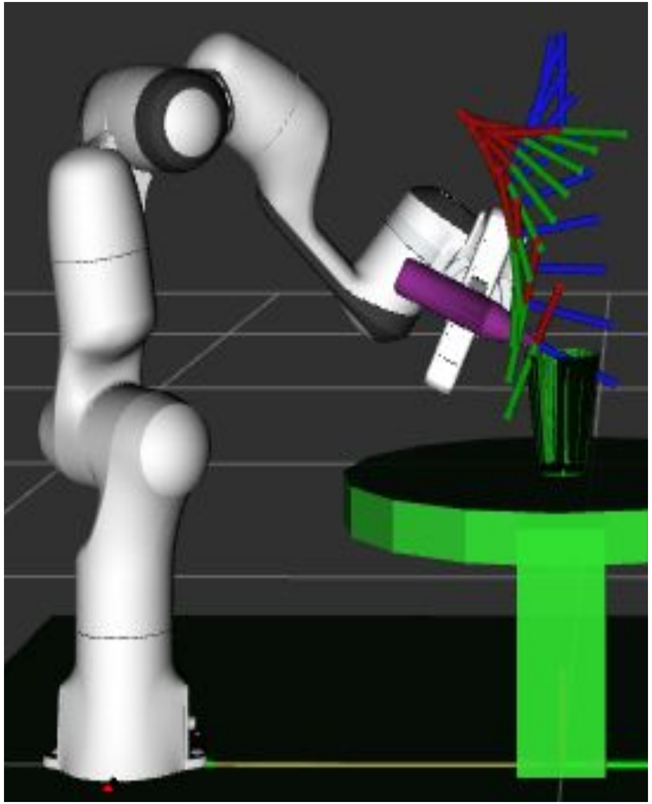


6. Grasp

7. Attach



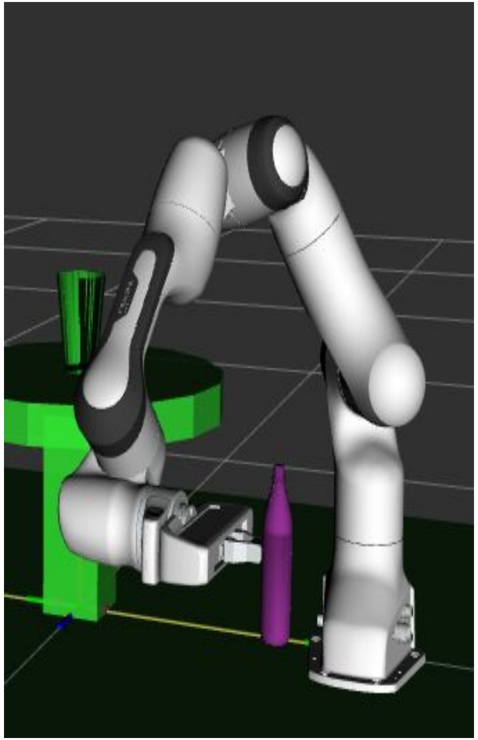




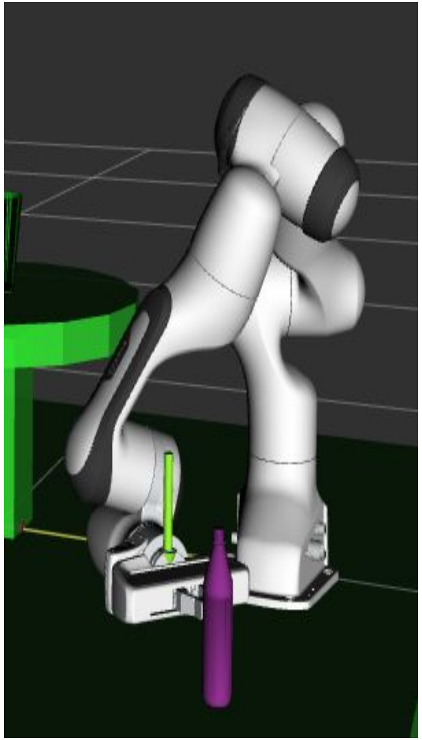
8. Lift

9. MoveTo Pre-Pour

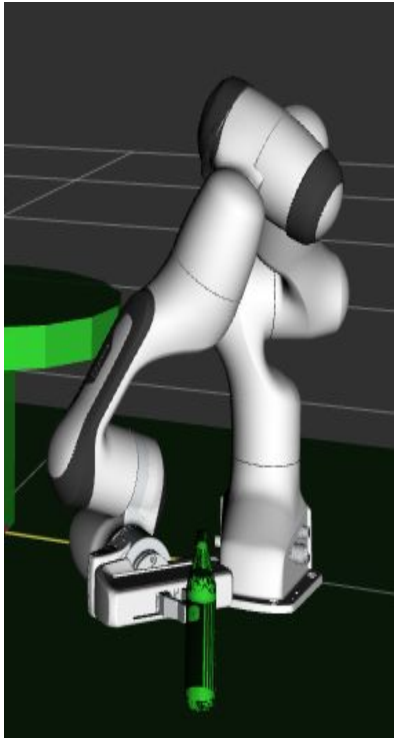
10. Pouring



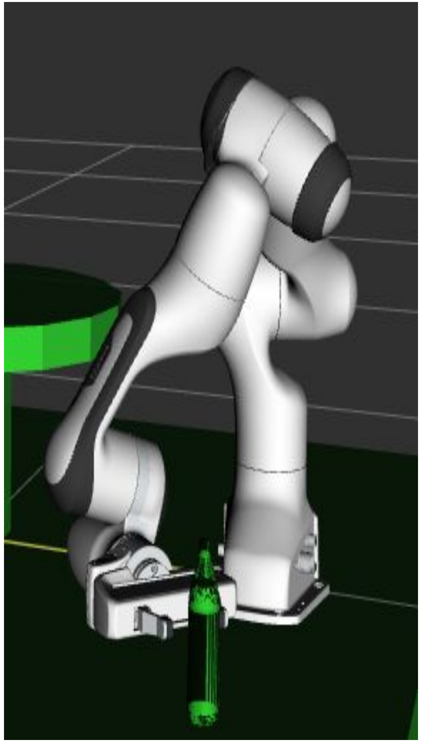
11. MoveTo Place



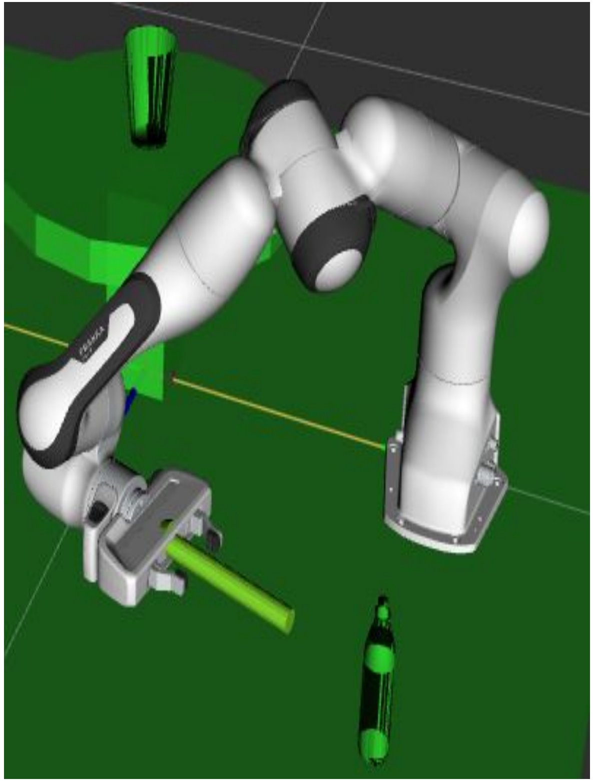
12. Lower



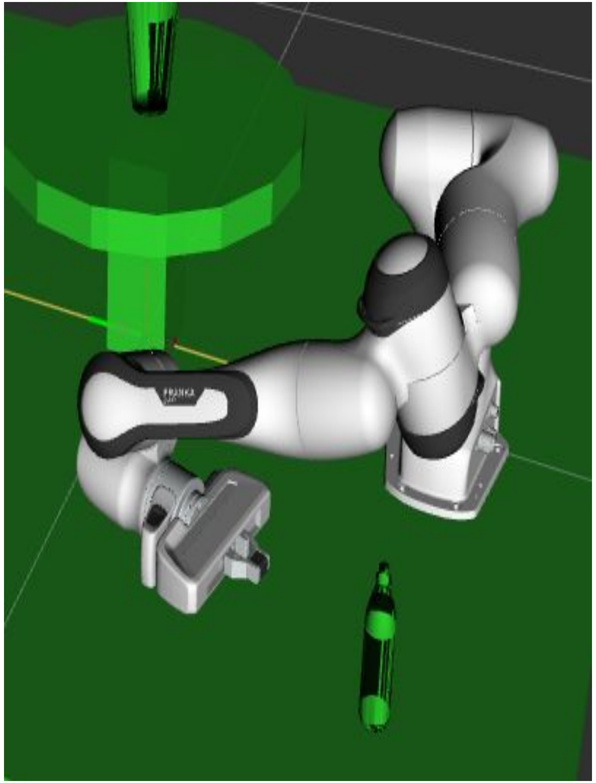
13. Detach



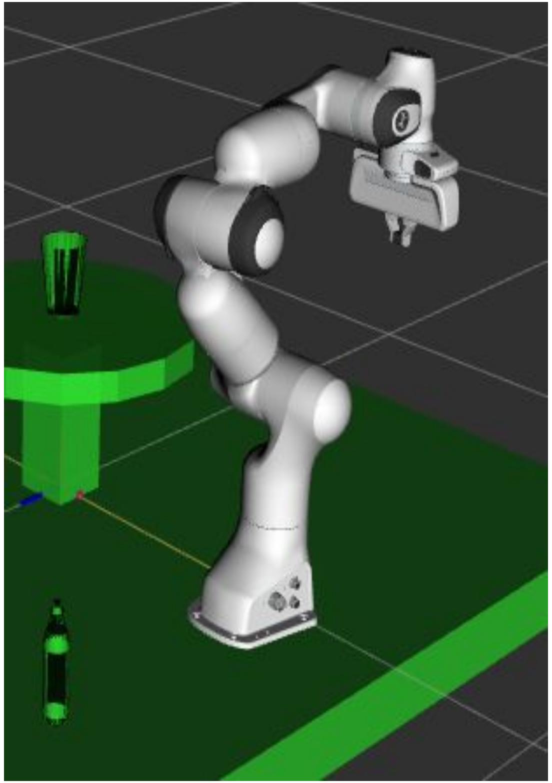
14. Open Hand



15. Retreat



16. Close Hand



17. MoveTo Home back

Motion Planning Tasks		
pick_place_task	8	0
applicability test	1	0
current state	1	0
move home	1	0
open hand	1	0
move to pick	17	0
pick object	18	0
approach object	21	26
grasp pose IK	47	11
generate grasp pose	25	0
allow collision (hand,object)	47	0
close hand	47	0
attach object	47	0
allow collision (object,support)	47	0
lift object	19	28
forbid collision (object,surface)	19	0
move to place	10	0
place object	9	0
allow collision (object,support)	9	0
lower object	17	47
place pose IK	64	45
generate place pose	47	0
detach object	64	0
open hand	61	3
forbid collision (hand,object)	61	0
retreat after place	9	52
close hand	9	0
move home2	9	0

move home2	6	0
open hand2	6	0
move to pick2	2	0
pick object2	3	0
approach object2	3	5
grasp pose IK2	45	3
generate grasp pose2	150	0
allow collision (hand2,object2)2	9	0
close hand2	9	0
attach object2	9	0
allow collision (object2,support)2	9	0
lift object2	3	6
forbid collision (object2,surface)2	3	0
move to pre-pour pose2	8	0
pre-pour pose2	46	0
pose above glass2	9	0
pouring2	6	3
move to place2	3	0
place object2	3	0
allow collision (object2,support)2	3	0
lower object2	5	3
place pose IK2	18	0
generate place pose2	9	0
detach object2	9	0
open hand2	9	0
forbid collision (hand2,object2)2	9	0
retreat after place2	4	5
close hand2	4	0
move home	3	0
move home2	3	0

Total of 60 stages

MoveIt! Task Constructor for Task-Level Motion Planning

Michael Görner*, Robert Haschke*, Helge Ritter, Jianwei Zhang

***Abstract*— A lot of motion planning research in robotics focuses on efficient means to find trajectories between individual start and goal regions, but it remains challenging to specify and plan robotic manipulation actions which consist of *multiple interdependent* subtasks. The Task Constructor framework we present in this work provides a flexible and transparent way to define and plan such actions, enhancing the capabilities of the popular robotic manipulation framework *MoveIt!*.[1] Subproblems are solved in isolation in black-box planning stages and a common interface is used to pass solution hypotheses between stages. The framework enables the hierarchical organization of basic stages using *containers*, allowing for sequential as well as parallel compositions. The flexibility of the framework is illustrated in multiple scenarios performed on various robot platforms, including bimanual ones.**

I. Introduction

Motion planning for robot control traditionally considers the problem of finding a feasible trajectory between a start and a goal pose, where both are specified in either joint or Cartesian space. Standard robotic applications, however, are usually composed of multiple, interdependent sub-stages with varying characteristics and sub-goals. In order to find trajectories that satisfy all constraints, all steps need to be planned in advance to yield feasible, collision-free, and possibly cost-optimized paths.

A typical example are pick-and-place tasks, that require (i) finding a set of feasible grasp and place poses, and (ii) planning a feasible path connecting the initial robot pose to a compatible candidate pose. This in turn involves approaching, lifting, and retracting – performing well-defined Cartesian motions during these critical phases. As there typically exist several grasp and place poses, any combination of them might be valid and should be considered for planning.

Such problems present various challenges: Individual planning stages are often strongly interrelated and cannot be considered independently from each other. For example, turning an object upside-down in a pick-and-place task renders a top grasp infeasible. Whereas some initial joint configuration might be adequate for the first part of a task, it could interfere with a second part due to inconvenient joint limits.

The present work proposes a framework to describe and plan composite tasks, where the high-level sequence of actions is fixed and known in advance, but the concrete realization needs to be adapted to the environmental context. With this, we aim to fill a gap between high-level symbolic

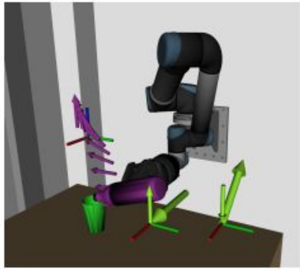


Fig. 1. Example task: a UR5 robot executes a task composed of (a) picking up a bottle from the table, (b) pouring liquid into a nearby glass, and (c) placing the bottle in a different location. Markers show key aspects of the task, including approach and lift directions during (a), bottle poses for (a) and (c), and the tip of the bottle during (b). Fig. 5 illustrates the associated task structure.

task planning and low-level, manipulation planning, thus contributing to the field of *Task and Motion Planning*.

Within the framework, *tasks* are described as hierarchical tree structures providing both sequential and parallel combinations of subtasks. The leaves of a task tree represent primitive stages, which are solved by arbitrary motion planners integrated within MoveIt!, thus providing the full power and flexibility of MoveIt! to model the characteristics of specific subproblems. To account for interdependencies, stages propagate the world state of their sub-solutions within the task tree. Efficient schedulers are proposed to first focus search on critical parts and cheap-to-compute stages of the task and thus retrieve cost-economical solutions as early as possible. Continuing planning can improve the quality of discovered solutions over time, taking into consideration all generated sub-solutions.

Additionally, the explicit factorization into well-defined stages and world states facilitates error analysis: individual parts of the task can be investigated in isolation and key aspects of individual stages can be visualized easily. Fig. 1 illustrates an example task with supporting visualizations.

II. Related Work

The scope of this work lies between two fields of research. On the one side, *manipulation planning* emphasizes the problem of trajectory planning with multiple kinematic and dynamic constraints [1], [2]. These approaches can cope with

* These authors contributed equally to this work.

This work was supported by the DFG Center of Excellence EXC 277, the DFG Transregional Research Centre CML, TRR-169, and has received funding from EU project SaraFun (grant 644938).

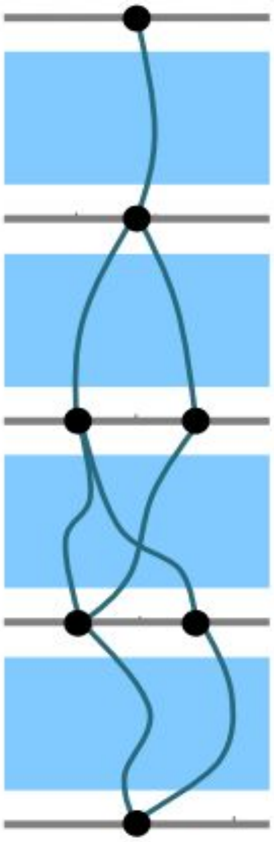
[1] The Task Constructor framework is publicly available at https://github.com/ros-planning/moveit_task_constructor

Following this research paper

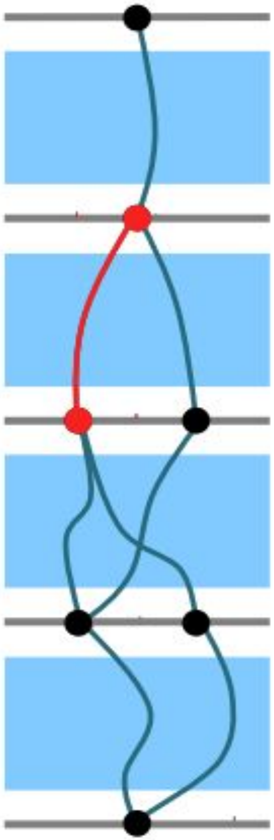
Author:

Michael Görner
Robert Haschke
Helge Ritter
Jianwei Zhang

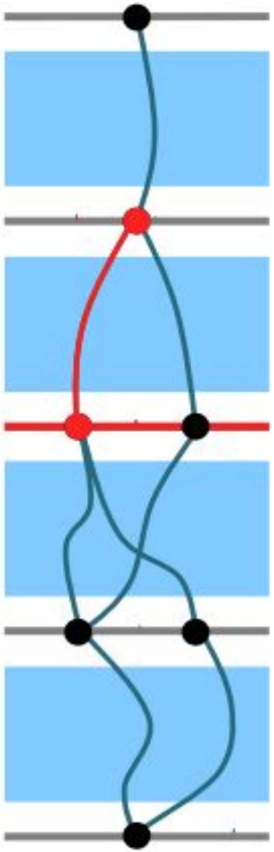
Reference - https://pub.uni-bielefeld.de/download/2918864/2933599/paper.pdf



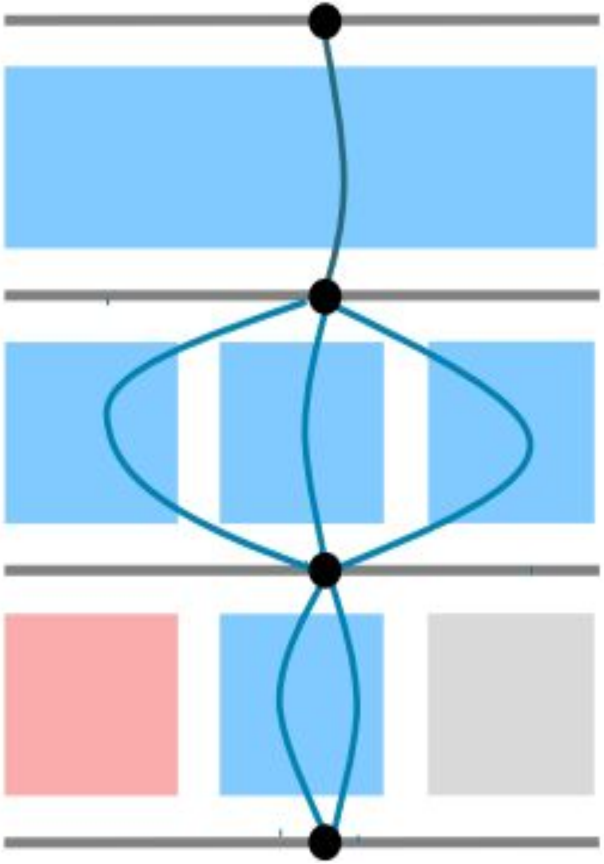
Pipeline of stages



Multiple Solutions



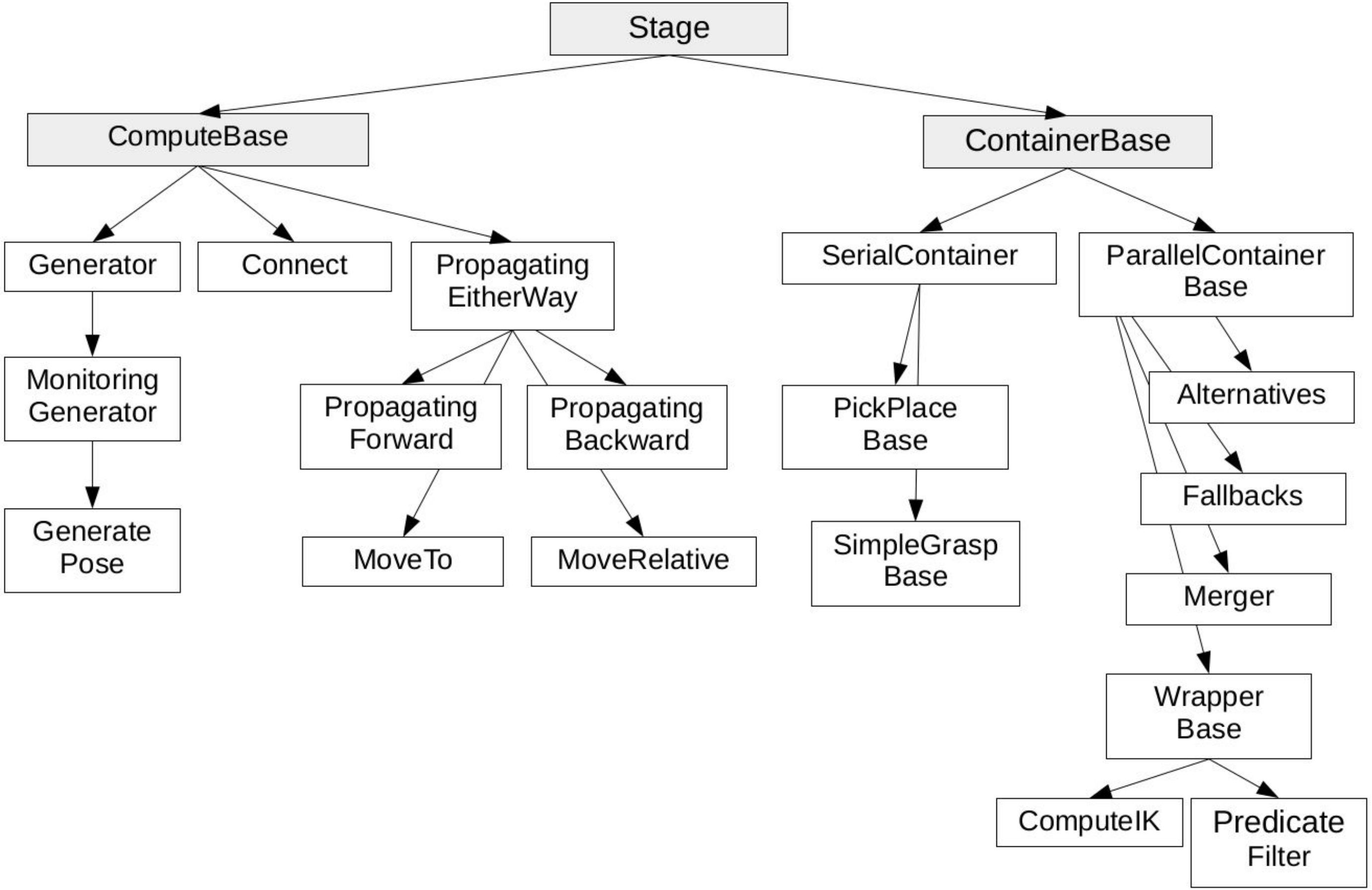
InterfaceState



Hierarchical Structuring

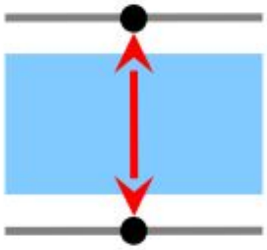
Reference: https://ros-planning.github.io/moveit_tutorials/doc/moveit_task_constructor/moveit_task_constructor_tutorial.html

Hierarchical Structuring



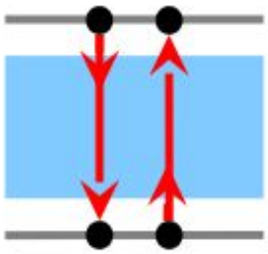
Reference: https://ros-planning.github.io/moveit_tutorials/doc/moveit_task_constructor/moveit_task_constructor_tutorial.html

Generator



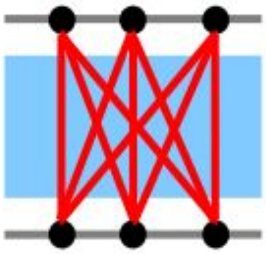
- Produces and propagates InterfaceStates to adjacent Stages
- E.g. IK generator

Propagator



- Receives an input InterfaceState, solves a problem and propagates the solution state
- E.g. MoveTo, MoveRelative

Connector



- Connects InterfaceStates of both adjacent stages
- E.g. Free-motion plan between start and goal states

Reference: https://ros-planning.github.io/moveit_tutorials/doc/moveit_task_constructor/moveit_task_constructor_tutorial.html

Discussing Issues/Challenges

rhaschke commented 10 days ago — Collaborator

It's not possible to merge trajectories generated by a serial container.
Consider some substages in the serial container that just modify the planning scene (e.g. attaching/detaching an object, or modifying the ACM). How should we merge such a modification into another motion trajectory?

v4hn commented 2 days ago — Member

> On Tue, Feb 25, 2020 at 12:27:02AM -0800, Rajendra Singh wrote:
> > To call your two execute_helpers ...
> Thank you I understood. Can we change this preempt behaviour of action goal?

This is a matter of changing the ExecuteTaskSolution capability, at least, to a general `ActionServer`.
This requires additional bookkeeping, probably a similar transition in general plan execution in MoveIt and would basically "only" add support for your current use-case where you want to execute independent controllers.

Of course, you're welcome to provide a pull-request that achieves this behavior, but the more reasonable
solution for yourself might be to run two independent `PlanExecution` classes locally, or even execute the subtrajectories of the solutions yourself
by sending them to the correct `FollowJointTrajectory` actions. This is of course not very elegant, though...

rhaschke commented 3 days ago — Collaborator

To call your two execute_helpers independently, you can just use two threads directly.
But, even if you manage this, I don't think, a single move_group node can handle two execution requests in parallel. As the corresponding capability relies on a `SimpleActionServer` the following doc applies:

> only one goal can have an active status at a time, new goals preempt previous goals based on the stamp in their GoalID field (later goals preempt earlier ones)

3. Conclusion

Single arm ✓✓ | **Multi-arm ✓✓** | **Parallelising Task ?** | **Moveit2 & MTC ✗✗**

Done:
Joint state goals
Cartesian goals
Pick Place task
Pouring task

Done:
Joint state goals
Cartesian goals
Pick Place task
Pouring task

Done:
Merger, Alternative, Fallout, Multi task planning

In Progress:
Multi move_group, non preemptable goals

Future Work:
I would work on moveit2 and MTC which would also involve porting them to ROS2.

THANK YOU!

Any Question?