PICKNIK

DUAL ARM

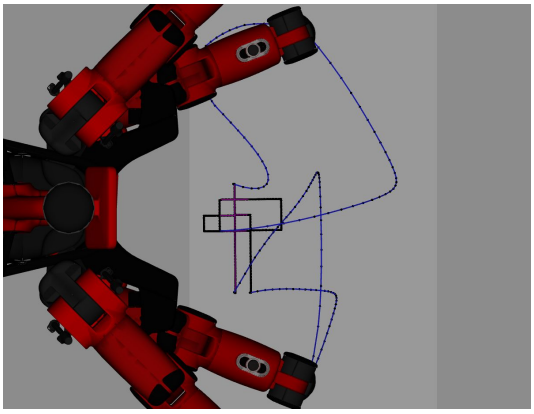
Cartesian Planner

What is PickNik DualArm?

DualArm is a unique motion planner that can generate smooth Cartesian trajectories for two or more arms using underconstrained input reference paths. There are many applications for underconstrained Cartesian planning, including high-speed industrial processes such as routing, grinding, or wiping down surfaces. In these applications the end effector is required to follow a series of waypoints, but each waypoint has some tolerance for the angle in which the task is performed.

How is DualArm different from other Cartesian planners?

Descartes



While Descartes enables roboticists to perform path-planning on under-defined Cartesian trajectories, it lacks support for two or more arms. Additionally, Descartes has not been tightly integrated into MoveIt! and is not enabled for Bolt experience-based planning.

MoveIt!

MoveIt! provides support for dual arm free-space path-planning, but not dual arm Cartesian path-planning. This results in MoveIt!-generated paths not following the expected path along a start and goal state.

What is an underconstrained input reference path?

DualArm takes as input underconstrained reference paths - this means the path is not fully defined in various axis/dimensions. A classic example of this is the task of drawing an image on

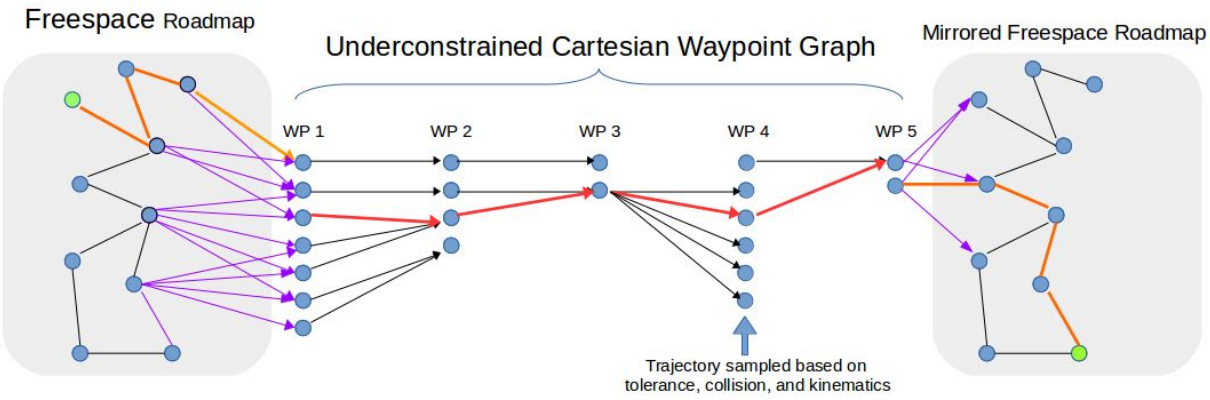
a whiteboard with a marker. The rotation of the marker along the axis pointing towards the whiteboard does not matter, so is considered underconstrained.

What are the benefits and drawbacks to underconstrained paths?

A traditional Cartesian planner will take as input fully constrained paths and produce for each timestep, a set of solutions that solve the inverse kinematics for the robot. These states will be used to populate a graph over which a shortest path search algorithm such as A* is run to find the best path from the start state to the end state through each intermediate point. This approach fails when the algorithm is unable to find a pair of nodes from two sequential time steps with an edge between them that respects the physical constraints of the robot.

PickNik DualArm solves this common problem by using underconstrained input reference paths to increase the set of valid nodes at each timestep thus reducing the chances of returning an incomplete solution. The tradeoff here is an increase in computation time in exchange for better output trajectories. PickNik DualArm makes this trade off explicit by allowing the user to tune the orientations tolerances along the path.

How are the PickNik DualArm trajectories generated?



First a set of candidate Cartesian tolerance poses are generated for each Cartesian pose along the waypoint path by generating alternative poses within tolerance for a specified discretization factor. Next, for each pose an IK solver is used to find redundant joint solutions at a specified discretization factor. This is achieved by iteratively changing the joint value of any free joints - joints beyond the required 6dof or the finite redundancy in a 6dof pose itself. For every point in our nominal path we should have many candidate joint solutions to achieve that point.

Every state will be used to populate vertices in a graph over which a shortest path search algorithm like A* runs. The cost of the number of potential edges that will be generated in our underconstrained Cartesian graph becomes a bottleneck, and so it is important to prune as many vertices and edges ahead of time as possible using collision checking.

We combine the set of candidate joint states for each arm into a unified dual-arm set of joint states.The most complete method for combining both arms is the quadratic with two arms method: for every candidate in one arm, combine with every candidate in the other arm. This results in candidate states for each nominal point along the input trajectory.

Finally, we run a secondary collision checking step with the unified dual-arm robot states to eliminate configurations that are in self-collision.

What kinds of input paths are allowed?

DualArm can accept input from trajectory files generated from CAD meshes, VR devices, or other task generators. The input is a nominal 3D path, specified as two sets of discretized waypoints---one for each arm---as shown in the Figure. Additionally specified are tolerances for each orientation axis (roll, pitch, and yaw) that can vary or remain constant throughout the trajectory.

Can it account for overlapping joints in the torso?

DualArm has not been adapted for this yet, but very similar approaches to the ones described here would still apply.