N-Link Arm Extension and Dual-Arm RRT* Motion Planning

Kaya Celebi Rajan Subramanian Nicholas Bykhovsky
Columbia University, Department of Computer Science
{kc3512, rs3166, nb3227}@columbia.edu

***Abstract*—This milestone extends our dual-arm motion planning simulator to support arbitrary n-link arm configurations and integrates RRT* sampling-based motion planning in high-dimensional configuration space. The previous system supported only 2-link planar arms and lacked obstacle-aware collision detection or any actual planning. We refactored the architecture to support 6-link manipulators (12D configuration space), implemented unified collision detection for arm-to-arm and arm-to-obstacle interactions, and integrated a modular RRT* planner. We additionally experimented with two different cost functions for RRT*—one entirely in configuration space, and one defined in the workspace using end-effector distances. The latter produced noticeably more direct motion in workspace. The system now supports interactive testing, obstacle manipulation, and visual debugging, forming the basis for real-world experiments on physical hardware.**

I. Introduction and Changes Since Milestone 1

Milestone 1 introduced a dual 2-link planar arm system with basic forward and inverse kinematics, simple 4D configuration space validation, and arm-to-arm collision detection. However, it lacked extensibility, obstacle interaction, and planning.

In this milestone, we extended the system to support arbitrary n-link arms, added full obstacle-aware collision detection, and integrated RRT*. We additionally implemented both configuration-space and workspace-space cost functions for RRT*, enabling direct comparison between the two approaches.

II. Technical Implementation

A. Abstract Robot Arm Interface

We designed a general `RobotArmBase` class exposing:

- `forward_kinematics(*angles)`
- `jacobian(*angles)`
- `ik_iterative()`
- `get_num_joints()`
- `get_joint_limits()`
- `is_valid_configuration(*angles)`

All motion planning algorithms interact with arms exclusively through these methods. Arms remain stateless, and planning modules store configurations.

B. Six-Link Planar Arm

The 6-link arm is implemented using cumulative-angle forward kinematics:

$$x = \sum_{i=1}^{6} L_i \cos\Big(\sum_{j=1}^{i} \theta_j\Big), \qquad y = \sum_{i=1}^{6} L_i \sin\Big(\sum_{j=1}^{i} \theta_j\Big). \quad (1)$$

The Jacobian accounts for each joint's influence on all downstream links. Inverse kinematics uses Jacobian-transpose iteration with flexible initial guesses and relaxed tolerances, reflecting the ill-conditioned nature of planar 6-DOF IK.

C. Dual-Arm System

The dual-arm system accepts any two instances of `RobotArmBase`. Configuration vectors are concatenated:

$$[\theta_1^L, \ldots, \theta_n^L,\ \theta_1^R, \ldots, \theta_m^R].$$

Arms may be heterogeneous. The system automatically determines total configuration dimensionality.

D. Collision Detection

Collision validation integrates:

- per-arm joint limit and workspace checks,
- arm-to-arm segment intersection,
- obstacle intersection for circles, rectangles, and polygons.

Link segments are extracted through forward kinematics, and obstacles are handled with standard line-circle or line-polygon intersection tests.

III. RRT* Integration

The RRT* planner operates entirely in configuration space for sampling, steering, and collision checking. Each node stores:

$$(\text{config},\ \text{parent},\ \text{cost}).$$

Tree expansion always uses configuration-distance for nearest-neighbor search, ensuring consistent exploration.

A. Two Cost Functions: Configuration-Space vs Workspace-Space

We implemented and compared two independent cost functions:

a) 1. Configuration-Space Cost (Default): The standard RRT* approach uses Euclidean distance between joint configurations:

$$\text{cost} = \|q_1 - q_2\|.$$

This favors smooth joint trajectories but does not guarantee short or intuitive workspace motion. For high-DOF arms, small angular differences can correspond to large swings in workspace.

b) 2. Workspace-Space Cost (Experimental): We added a mode that measures cost based on end-effector distance in workspace:

$$\text{cost} = d_{EE}(q_1, q_2),$$

where

$$d_{EE} = \|EE_L(q_1) - EE_L(q_2)\| + \|EE_R(q_1) - EE_R(q_2)\|.$$

Rewiring and neighbor-finding are then performed in workspace units.

c) Observed Differences: Both cost functions produced valid collision-free trajectories. However:

- Configuration-space cost produced natural joint motions but often created unnecessarily wide workspace arcs.
- Workspace-based cost produced significantly straighter end-effector paths and more direct obstacle navigation.

These two optimization objectives highlight the difference between planning "for the robot's joints" and planning "for the task." We are continuing to compare these behaviors systematically.

IV. Experimental Results

A. 2-Link System

The 4D configuration space is small and RRT* reliably finds paths around obstacles. Planning remains fast and collision checking lightweight.

B. 6-Link System

The 12D configuration space requires more extensive exploration. Collision checking dominates runtime due to the increased number of segments. RRT* still finds paths consistently, and the workspace-based cost improves path efficiency in workspace.

Figures 1, 2 illustrate typical trajectories:

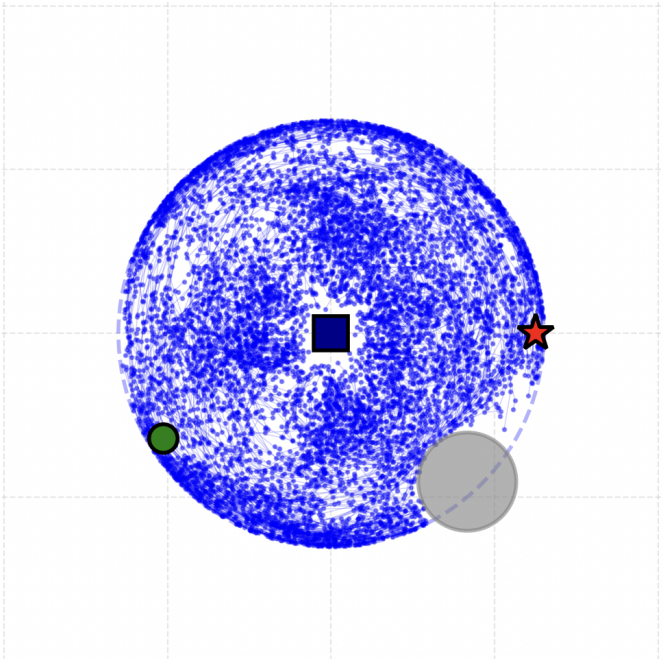

Fig. 1. Example RRT* result for 2-link arms in the configuration space before rewiring (all tried points).

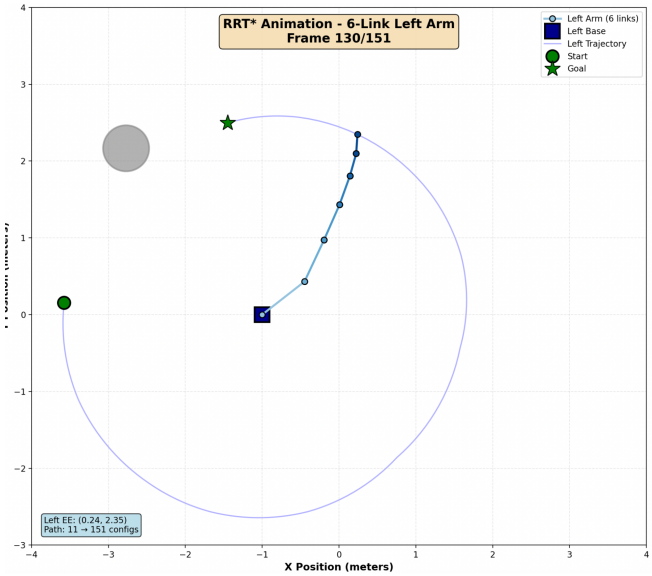

Fig. 2. Example RRT* result in the workspace for 6-link arms after rewiring. *Above one can observe how the C-space cost function produces a simple path for the joint angles, but a supotimal route in the workspace.*

V. Validation Tests

Two test suites were created:

- `test_motion_path.py` validates interpolation-based paths without running RRT*.
- `test_rrt_workspace_cost.py` and `test_rrt_config_cost.py` directly compare configuration-space and workspace-based cost strategies.

The workspace-based RRT* was also evaluated in an interactive setup where start and goal positions, as well as

obstacles, can be dragged in real time. This enables immediate visual comparison between the two cost models. This setup mirrors how we intend to extend the system into the real world using an SO-ARM100. The planner already accepts externally supplied obstacle geometry; a wrist-mounted camera on the SO-ARM100 will provide these observations. A small perception layer will convert camera detections into updated obstacle positions for the planner, enabling continuous replanning as the workspace changes. This serves as the bridge to our remaining work.

VI. Remaining Work and Real-World Integration

We will focus on:

- systematic comparison between configuration-space and workspace-space cost functions,
- smoothing and post-processing for both path types,
- support for dynamic obstacles and online replanning

We also acquired a pair of SO-ARM100 manipulators and will begin transferring the pipeline to hardware. Our goals include:

- calibrating each arm's link lengths and joint offsets,
- building an interface between perception and our internal workspace representation (with the associated topics recently covered in homework 3),
- validating workspace-cost planning under real kinematics and sensing noise,

These real-world experiments will form the basis for the final demonstration.

VII. Conclusion

This milestone implements a scalable dual-arm planning framework supporting n-link manipulators, modular collision detection, and configurable RRT* cost functions. Introducing workspace-aware cost significantly changes path characteristics and produces more direct task-centric motions. Combined with our incoming SO-ARM100 robots, the system is positioned for direct real-world evaluation of dual-arm obstacle-aware planning.