16-662 Robot Autonomy Project Final Report
Multi-Robot Motion Planning In Tight Spaces

Aum Jadhav
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
ajadhav@andrew.cmu.edu

Cyrus Liu
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
xiyuanl1@andrew.cmu.edu

Kazu Otani
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
kotani@andrew.cmu.edu

Max Hu
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
yuanh@andrew.cmu.edu

Abstract

We have developed a system for testing multi-robot motion planning algorithms, using Anki Cozmo robots. To demonstrate our framework, we have also implemented a recent algorithm for planning the motion of a fleet of robots operating in tight spaces [1]. The main tasks included establishing centralized multi-robot control of the Cozmo robots, designing a global localization system, and implementing the motion planning algorithm. We successfully tested our system on three robots operating in a tight T-shaped map.

1 Problem definition

Multi-robot planning is the problem of planning trajectories for multiple robots such that they:

- Drive robots from initial to final configurations
- Avoid static and dynamic obstacles
- Avoid inter-robot collisions
- Respect dynamic models of the robots (kinematic model, velocity/acceleration limits)

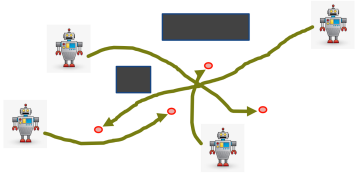

Figure 1: The multi-robot planning problem

While many algorithms have been proposed for multi-robot motion planning, most of them have only been tested in simulation. Our goal was to build a test platform for multi-robot motion planners. We found that when even small amounts of noise or uncertainty are introduced (which will happen in all real-world robotics situations), the performance of the robot fleet degraded quickly and often did

not match their behavior in simulation. We believe that having a physical platform to test motion planning algorithms will help researchers and students design more robust planners.

2 Related works

Our project is based on the recent work on Multi Robot Discrete RRT (MRdRRT) in [1], detailing a sampling-based multi-robot motion planning framework on composite roadmaps. The idea of planning on composite roadmaps is not new; it has been used in previous works [3][4]. The innovation of MRdRRT was that it used a sampling-based algorithm to plan across the graph. The authors show that this makes MRdRRT up to 10 times faster than previous methods, especially in high-dimensional problems. We also referred to [2] for the Local Connector method used in [1]. This algorithm allows us to find a collision-free sequence of robot motions, given start and goal configurations for each robot.

3 System design

In this section, we will discuss the resources and infrastructure we were provided, as well as how that affected the design decisions that shaped our system.

3.1 Hardware

The Anki Cozmo robots has is a differential track drive robot with an actuated forklift and an articulated head. A monocoluar camera is built-in to the head. The forklift allows it to interact with objects in its envrionment, including Lightcubes. The robot also has wheel encoders and optical sensors to aid in localization.

Figure 2: Anki Cozmo robot

Figure 3: Cozmo's 3 Light Cubes

The robot interacts with its Lightcubes visually and wirelessly. The image tags printed on the cube faces allow the Cozmo to infer 6DoF pose information with respect to the cube. The Lightcubes also have accelerometers and LEDs in them that are used during interactive games with Cozmo, but we do not utilize these functions.

3.2 Architecture

To set up centralized communication with the robots, we extended an open-source ROS Cozmo driver [5]. The driver implements several wrapper and helper functions that enable interfacing with Anki's own Cozmo SDK [6] within a ROS framework. To connect the master computer to the robots, we first connect an Android phone with the Cozmo app to each robot. We then put them in SDK mode, and connect the Android phones to the computer with USB debug enabled.

We used ROS namespaces to have an instance of the driver node running for each robot. This way, controlling additional robots is just a matter of adding another robot namespace in the roslaunch file.

Figure 4 shows the node graph for 2 robots. Notice how each namespace has a driver node, but there is only one mrdrrt_node, which is the central planner node. Mrdrrt_node is called with a rosservice call to start planning, and is given some goal configurations. It first queries TF to get the start configurations of the robots, then constructs a path for each of the robots. The central planner node then passes waypoints to the robots one at a time, making sure to wait for all of the robots to

finish the current set of waypoints before sending the next commands. This way, we ensure that the robots move in a synchronized fashion and do not collide with each other.

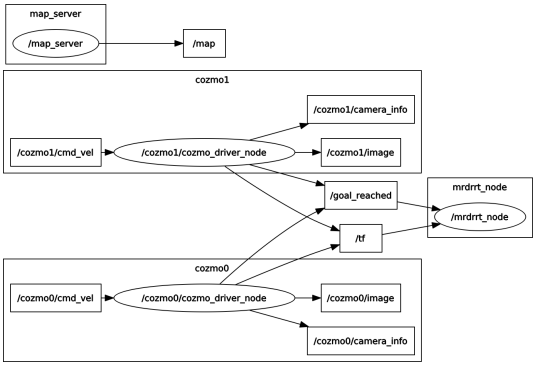

Figure 4: Node graph showing separate namespaces for each robot, and a central planning node

3.3 Interface

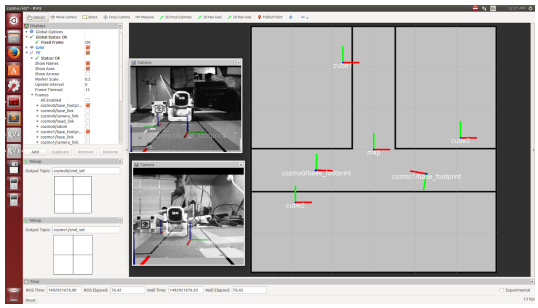

Figure 5: Rviz visualization of multiple robots

Since we are working within the ROS ecosystem, we are able to leverage visualization tools like RViz, that allows us to keep track of the status of the robots. The interface that we have setup in RViz is shown in Figure 5. The interface shows video streams from each of the Cozmos, and frames for the map, cubes, and robots. We have also set up an interface to manually control the robots (two squares on the lower left corner).

3.4 Localization

Cozmo's localization is based on its wheel odometry (dead reckoning) and drift correction when observing Lightcubes. Internally, Cozmo keeps track of its position with respect to its own origin, which is the defined as the point at which the Cozmo was turned on. As the robot moves, the pose of the robot with respect to its internal origin ("odom->base_footprint" frame transform in Figure 6) is updated with wheel odometry. This pose estimate drifts over time, so our cube localization system compensates for this.

When we design the map, we also define the cube locations and orientations in the map. For simplicity, we typically aligned the cube axes with the global map axis. Position coordinates of the cubes are encoded within the cozmo_driver node. When a robot sees a Lightcube, the SDK gives us the estimated transformation from the robot frame to the cube frame. We use this to calculate the robot's global pose, and publish a map->odom transform that compensates for the odom->base_footprint frame drift.

We noticed that the cube pose estimation was quite limited in range and accuracy. It had a maximum range of 35 to 40cm: past that distance, the robot would not register any cubes, even if the cube was in the robot's field of view. There was also significant noise in the pose estimate when the cube is further away, with variances of up to 10cm and 30 degrees. To mitigate the effects of this noise, we used the robot_localization package in ROS to run an Extended Kalman Filter for our pose estimate.

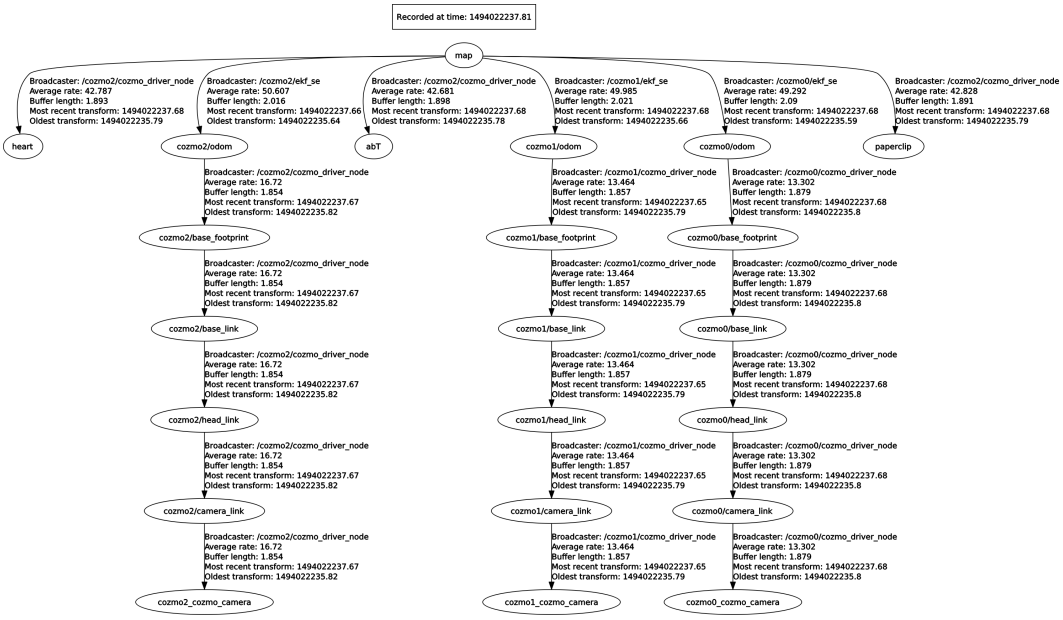

Figure 6: Transform tree showing the relationships between the global map frame, robots, and cubes as

3.5 Environment

We aimed to design an environment that would:

- Showcase the strengths of the MRdRRT algorithm
- Allow the Cozmos to be consistently localized

To achieve the first design goal, we created a constrained T-shaped map, as shown in Figure 7. In this map, it is not possible for two robots to move past each other if they are in the same channel. This forces them to utilize the extra space at the ends of the channels to get out of each other's way in order to achieve their collective goal.

Given our map design, we then had to decide on cube locations. We aimed to place the cubes in locations that would be seen from as many configurations as possible, taking into consideration field of view, range of detection, and orientation. For our T-shaped map, we decided to place one cube at each junction, and the third cube on the other side of the channel.

Notice that the robots in Figure 7 are placed in orientations from which they can easily see a cube. Before planning, it is necessary to initialize a robot in this manner to ensure that it knows its global location.

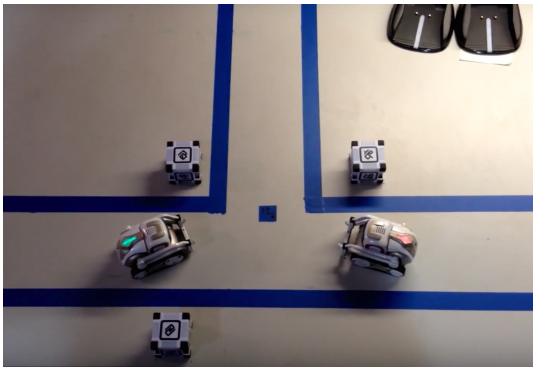

Figure 7: Our final map

3.6 Locomotion

The Cozmo SDK provides functions for driving Cozmo in a straight line for a designated distance, and turning in place by a designated angle (defined as drive_straight and turn_in_place respectively). It also provides a function for commanding the robot to move to a set waypoint. By default, this function considers all coordinates to be in the Cozmo's local origin, which is our 'odom' frame. After implementing localization and coordinate transforms, we were able to use handle waypoints in the global map frame. However, the SDK's go_to_waypoint function seems to use a controller with built-in trajectory shaping, where it moves along an arc towards the goal point. This makes collision checking between waypoints harder for our planner, as we do not know the exact path the robot will take. Hence, we decided to implement our own go_to_waypoint function.

To simplify the motion planning problem to straight lines between waypoints, we implemented the go_to_waypoint function as follows:

1. Turn in place towards goal position.

2. Drive straight to goal position.

3. Turn in place to the goal orientation.

The robot re-localizes at each waypoint (using visible cubes) to correct for drift.

3.7 Planning

To demonstrate our framework, we implemented the MRdRRT algorithm [1].

It is a centralized multi-robot planning algorithm, which makes it well-suited for problems that require tight coordination between robots. Cozmo robots do not recognize each others' presence, so a centralized planner (as opposed to distributed/decentralized) seems to be simpler in implementation.

MRdRRT works by planning over a space of composite configurations, which is the set of robot configurations within a pre-computed probabilistic roadmap (specific to the robots and environment). The vertices in the composite roadmap represent all combinations of collision-free placements of the m robots, and the edges represent all combinations of robot movements where the robots remain collision-free while moving on their respective single-graph edges. Because the size of this graph grows exponentially with number of robots, it may become infeasible to explicitly represent the entire graph in memory. For this reason, MRdRRT represents the composite roadmap as an implicit graph, inspired by [3].

The difference between MRdRRT and previous methods such as M* [4] is that MRdRRT is a sampling-based algorithm, while other algorithms search over the implicit graph. This allows fast solutions to high-dimensional motion planning problems. In each step of the algorithm, a random composite configuration is sampled, similar to the first step in traditional RRT. The tree then expands to the nearest node on the implicit graph (this is the "discrete" part of MRdRRT). By repeating this process, we are able to find a path of composite configurations from start to goal, which can then be broken down into paths for individual robots.

MRdRRT also uses a "Local Connector" step that checks the validity of an edge on the implicit graph by making sure that the robots can move in collision-free paths along the implicit edge. If there are collisions between robots assuming they all move at once, the local connector then checks for an order of movement among the robots that will result in a feasible trajectory. For this step, we used a topological sort on a dependency graph generated by checking the start and goal configurations of the robots. This approach was inspired by and similar to the method used in [1], but slightly different in implementation.

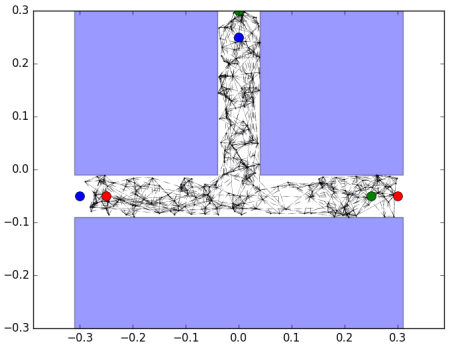

Figure 8: Our planning environment, with start and goal configurations for three robots.

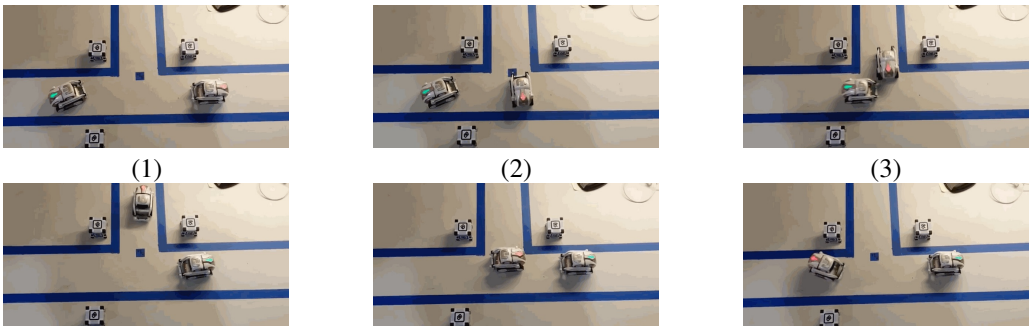

Figure 9: MRdRRT in Motion

4 Challenges

4.1 SDK Hacks

Our framework relies on known cube-to-location mappings for global localization. This was made difficult by the fact that each Cozmo arbitrarily assigns cube IDs to the three cubes upon initialization, despite the fact that the SDK documentation claims to have strict mappings between cube IDs and the markings on the cubes. This resulted in situations where one Cozmo would enumerate one cube as cubeID = 1, while another Cozmo would enumerate the same cube as cubeID = 2. These assignments also tend to shuffle around when the SDK program is re-initialized.

We found the code block in the SDK that causes this problem, and it is shown in Figure 10. Obviously, the developers know this is not a robust solution. However, because the Cozmo is made to be a toy and most consumers only own one robot, this is unlikely to be a problem for the vast majority of users.

We devised a quick and simple solution to get around this issue. Once a Cozmo is initialized, the cube IDs stay constant. We hard-coded cube symbol relationships in the driver, and modified the mapping from cube IDs (the numbers that the SDK reports) to symbols for individual robots after connecting them.

```
def _init_light_cubes(self):
    # XXX assume that the three cubes exist; haven't yet found an example
    # where this isn't hard-coded.  Smells bad.  Don't allocate an object id
    # i've seen them have an ID of 0 when observed.
    self.light_cubes = {
        objects.LightCube1Id: self.light_cube_factory(self.conn, self, dispatch_parent=self),
        objects.LightCube2Id: self.light_cube_factory(self.conn, self, dispatch_parent=self),
        objects.LightCube3Id: self.light_cube_factory(self.conn, self, dispatch_parent=self),
    }
```

Figure 10: Excerpt from Anki's Cozmo SDK

4.2 Localization

Localization for the Cozmo was non-trivial, due to a variety of factors in locomotion, sensing, and software.

- Odometry Drift

  Cozmo's tank treads slip in some situations, especially while rotating in place, leading to odometry drift.

- Noisy pose information from cubes

  Lightcubes helped the robots to re-localize while executing a path, but pose information from the cubes was very noisy. We minimized the effect of the noise with an EKF node that filters the raw pose information.

- Software

  TF (ROS's built-in transform library) does not work with Python 3, but the Cozmo SDK required Python 3.5 as a dependency. Instead of querying TF for transformations between frames, we kept track of all relevant transforms within the cozmo_driver nodes and performed our own matrix operations to calculate the transforms as required.

4.3 Setup

The process for connecting multiple Cozmos to a single ROS master is as follows:

1. Connect each robot to an Android phone via Wi-Fi.
2. Put the robots into SDK mode via the Cozmo app.
3. Make sure USB debug mode is on for each of the phones, and plug them into a USB hub that is plugged into the computer.

This process became more cumbersome as the number of robots increased. Battery life was also a challenge, for both the Cozmos and phones. After about an hour of testing, we had to suspend our activities to recharge. We noticed that the Cozmo app used up battery life at a faster rate than the Android phones could be charged from the USB port.

Ideally, we would be able to connect to the Cozmos directly from the computer. A potential method is to have multiple instances of an Android emulator running, but this requires having each emulator connect to a unique Wi-Fi network for each Cozmo robot, further complicating the setup.

5 Results

To demonstrate our multi-robot planning platform, we implemented MRdRRT planning for 3 robots on our constrained T-map. A video of the demonstration, along with our code for the multi-robot Cozmo driver can be found here: `https://github.com/mrsd16teamd/cozmo_mrdrrt`.

We found that many of the plans generated by MRdRRT, especially for 3 robots, had many back-and-forth movements and were clearly suboptimal. This is the nature of the algorithm's sampling based strategy, as noted in [1]. The path shown in the video is the best path we got from many runs.

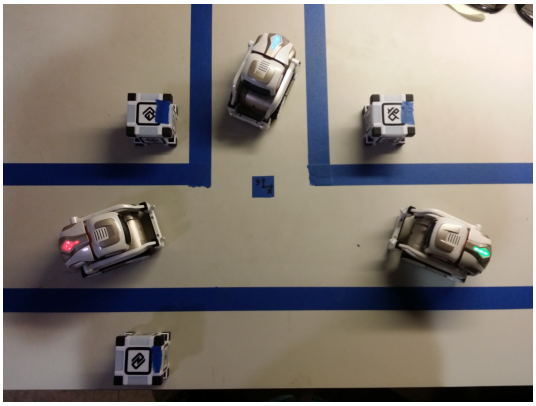

Figure 11: 3 robots attempt to switch places in a tight T-shaped map

6 Limitations

6.1 Assumptions in environment setup and collision checking

In our environment representation, we are currently assuming that all robots are circular holonomic robots moving in 2D. We further assume that all obstacles are axis-aligned rectangles. This allowed us to simplify our collision-checking to simple points in the configuration space.

6.2 Localization

Cozmo has a limited ability (in precision and range) to recognize and localize cubes. Currently, we are manually designing a map with 3 cubes. For larger environments, it will be necessary to extend on Cozmo's default localization system.

7 Future work

- Use additional "objects" in environment

  Cozmo's SDK provides the ability to create additional "objects" in the environment by printing out (physically, on paper) AR tags that are distinct from those found on the Lightcubes. These objects actually have unique object names and IDs, contrary to the Lightcubes. These objects can act as additional visual landmarks, allowing more robots to be added for localization in larger environments.
- Integrate Flexible Collision Library(FCL)

  With our current implementation, the environment and obstacle representation is very limited. In the future, we recommend that planners interfacing with our multi-Cozmo system use FCL or other collision checking libraries that allow the robots and obstacles to be represented as complex polygons/polyhedrons, in 2D and 3D.
- Implement planning algorithms in C++ for speed in higher dimensional planning problems

  We implemented our planning algorithm in Python for development speed and readability. This worked for our simple planning problem. However, the complexity of planning increases exponentially with the number of robots, so more complex planning problems will be better solved with a C++ implementation.

8 Work division

- Aum Jadhav - Planning algorithm implementation, infrastructure
- Cyrus Liu - Infrastructure, Planning algorithm implementation
- Kazu Otani - Planning algorithm implementation, multi-robot communication
- Max Hu - Localization and environment design, multi-robot communication

Acknowledgments

We would like to express our gratitude to our project sponsor, Dr. Oren Salzman, (osalzman@andrew.cmu.edu) for his technical guidance, and Vinitha Ranganeni (vrangane@andrew.cmu.edu) for the infrastructural support she has provided for this project.

References

[1] Solovey, K., Salzman, O., & Halperin, D. (2015). Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. In *Algorithmic Foundations of Robotics XI* (pp. 591-607). Springer International Publishing.

[2] van Den Berg, J., Snoeyink, J., Lin, M. C., & Manocha, D. (2009, June). Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Robotics: Science and systems* (Vol. 2, No. 2.5, pp. 2-3).

[3] Svestka, P., & Overmars, M. H. (1998). Coordinated path planning for multiple robots. *Robotics and autonomous systems*, 23(3), 125-152.

[4] Wagner G & Choset H (2015) Subdimensional expansion for multirobot path planning. Artificial Intelligence 219: 1–24.

[5] Ogura T. & Rudolph P. (2017). Anki Cozmo ROS driver. Github repository. https://github.com/OTL/cozmo_driver.

[6] Anki, Inc. (2017). Anki Cozmo Python SDK. Github repository. https://github.com/anki/cozmo-python-sdk.