Robotic Babysitter Project Final Report

Team Members:

Michael Vander Meiden (mvanderm) **Nima Rahnemoon (nrahnemo)**

Yoga Nadaraajan (yyn) **Kai Li (kail2)** **Benjamin Kuo (benjamik)**

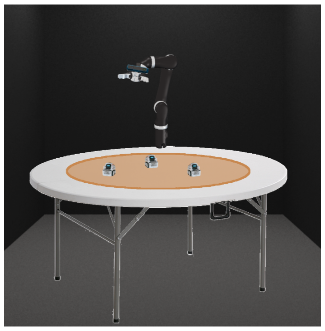

Figure 1: Final vision for the project.

1 Problem Definition

Kinematic planning of robotics arms for manipulating moving objects has been a major challenge in motion planning. Since some modern methods in path planning simply plan on fixed objects, factoring a temporal constraint is not straightforward for some of these plans. In addition, planning in real time is hard because it's almost a circular action to both factor in time, while also spending time for the plan. This project aims to address this problem. We experiment with different strategies for adhering to a real time constraint, reliable constrained trajectories, and methods for tracking moving items of interest. Ultimately we implement a method to flexibly manipulate one moving object to a fixed target location region.

The project was proposed by Dr. Daqing Yi, a postdoctoral researcher at CMU. He was inspired with the idea for the project after watching a video on YouTube[1]. The premise is that a mother of quadruplets is attempting to change all of the babies' diapers. The problem is that the babies are mobile and can and are attempting to move outside a region of interest, specifically a bed. Her problem is to simultaneously change one child, while keeping all the babies on the bed (i.e. a workspace).

The final scope of our problem resulted in the use of the Assistive Dexterous Arm (a Kinova Mico arm) to continuously track and prevent one moving object (Anki Cozmo) from leaving a region of interest and returning the object to a fixed region of "safety". More specifically, our moving object tries to leave a workspace region, but our arm's goal is to move it back roughly to the arm's fixed base position in the workspace. Throughout the object's movement, the arm only actively intervenes if the moving object strays close enough to the boundary of the work space. This is also a boundary of what is capable for the arm's "reaction time".

[1]https://www.youtube.com/watch?v=Pa0wXSh4I34

2 Related Works

Information on the referenced works are on our references page at the end of the document.

2.1 Constrained Bi-directional Rapidly Expanding Random Trees Planning and Task Space Regions

Each of our two main subsystems, path planning and perception, drew from related works in the field. For our path planning, we wanted to use the CBiRRT2 planner with task space regions . This functionality was built into OpenRAVE and was implemented into our system. The reason that we chose the CBiRRT2 planner is because the strengths of this planner line up very well with our project. CBiRRT2 uses Task Space Regions which are general purpose constraints that can be evaluated quickly. They can also be chained together to work on more complex tasks. The reason it is applicable to our project is because we have strong constraints of the motion of the end effector but the constraints of the rest of the arm are much looser. Namely, the end-effector must travel parallel to the plane of the table at all times.

2.2 APRILTags

Prior work done by Edwin Olson on APRILTags has made this project possible. We will discuss later how this was integrated into our project, however it played a key role in being able to prototype quickly. APRILTags were developed at the University of Michigan as an evolution of ARTag. The difference is that the APRILTags are completely open with code and algorithms documented online. The Apriltags allow for a full 6 DOF localization from only a single image and is resistant to various types of noise like occlusion, warping, and lens distortion. This complements the resource constraints we later faced with the quality of our camera sensor we used.

3 Approach

3.1 Cozmo Localization and Trajectory Estimation

A major portion of the system is our perception, and this is the first step in our manipulation pipeline. The major tasks that the vision system performs are surveying the table, marking our Cozmo robot, and marking a base tag. For the motion estimation, we used the APRILTag based visual trackers. For this part we used the library from the Personal Robotics Lab's repository instead of the original APRILTag library, as the PRL Lab's APRILTag library was better integrated with the ROS environment. We were not using them to detect the distance away from the camera because we are working in a mainly 2D-space, however these APRILTags are essential in locating the Cozmo robot.

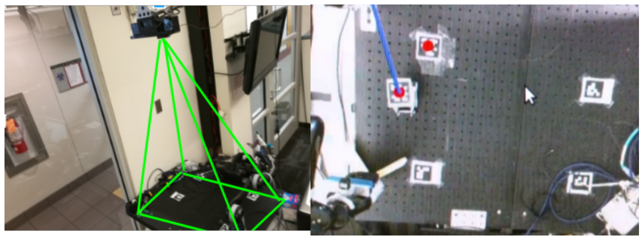
Figure 2: Camera setup (left) and APRILTag detection (right). One APRILTag is fixed to the table and is transformed to the base of the arm and a moving APRILTag atop the Anki Cozmo.

We built our tracker and estimation on top of the detections provided by the APRILTag Detector. Since we keep the arm and camera in a known fixed position, we have created a base APRILTag and a moving APRILTag that is made to move with the Cozmo. Though there is a difference in the distance from plane of the table to the top of the Cosmo, we think this distance is negligible. This issue is further reduced, as the scale of the APRILTag is irrelevant; i.e., we only care about the center position of the APRILTag. We get the relative pose between these two tags and track the tags over time. We have used these data in order to estimate the position and the velocity relative to the arm. Based on the direction of motion, we can extrapolate the trajectory of Cozmo, and figure out where the Cozmo might end up. We also had to take into account the speed at which the Cozmo is moving and the reaction-time of the arm. Based on these competing values of speed and reaction-time, we calculated a distance beyond which the arm will start its trajectory and move the Cozmo back to the start location. However, this is a maximum bound on this distance because our reaction-time actually varies depending on the region of the workspace.

3.2 Manipulation Strategy

There were a variety of directions we could have taken with the manipulation of the Cosmo robot. Initially we wanted to actually grasp the moving robot, but because of the very difficult nature of real time planning, we opted to choose a different strategy. We settled on the idea of predicting where the Cosmo would end up as specified before, and assumed that the plan, and execution to that position would take less time than the specified predicted point. This resulted in a custom designed end effector we had the ADA arm hold. The end effector was a V-shaped corralling tool. Using this strategy we would correctly predict where the cosmo would end up, and the tool would be there to catch the Cozmo robot when it did reach there. Without actually having to know if the Cozmo had made contact, our arm would then sweep back, and the Cozmo robot would be in the path of this motion. Thus, the Cozmo would be returned to a region that is farther from the edge of the workspace. The coralling tool also reduced the need for the planner to figure out the grasping strategies to pick up the Cozmo which was our earlier approach. Making ADA pick up the Cozmo is a particularly hard problem since at any moment, the Cozmo can be oriented in any direction and the grasping strategy for each orientation would be different.

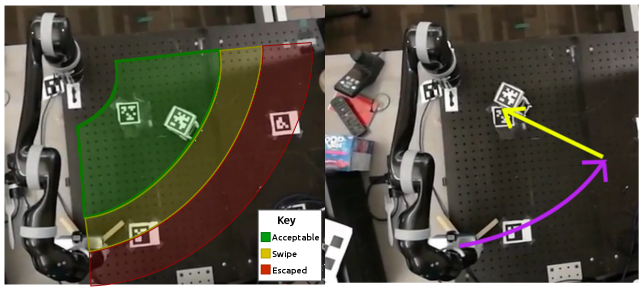

Figure 3: Different zones in the arm's workspace (left) and different pre-computed trajectories for swiping the Cozmo (right).

3.3 Planning Strategy

We used multiple planners and trajectory generation for keeping the Cozmo within the boundary. Each planner was used for different tasks based on each planner's advantages and disadvantages.

3.3.1 Snap Planner

The simplest planner is the Snap Planner. The snap planner takes in a final configuration and produces two waypoints as part of a trajectory that "snaps" the arm to the final configuration. Since this planner only needs to generate two waypoints, the planning time is relatively quick. Although the snap planner is fast, it fails a lot as it cannot handle collisions. It should be noted that the SnapPlanner does not take the straight line path from the end-effector's current position and the end-effector's desired position. Instead, it uses a linear interpolation between the current joint angle and the final joint angle for each joint. As a result, even if there are no obstacles between the start location and end location of the arm, it's still possible for the planner to fail, as the linear interpolated plan may intersect with obstacles. The snap planner only returns a trajectory with two waypoints and if any of those waypoints causes self collisions, the planner fails. We use this planner to correct any minor discrepancies between the start location and the arm's current position. Since the snap planner was fast compared to the other planners used, our initial approach was to use the planner throughout the project for dynamic replanning for each sweep. However, once we passed this planner through the simulator, it started failing because some of the trajectories produced by this planner involved the arm going through the table.

3.3.2 CBiRRT

The next planner that we used was the “Constrained Bidirectional Rapidly Expanding Random Tree” (CBiRRT). The underlying model behind this planner is the RRT planner. The main differentiating factor is that this planner is able to take into account constraints that we want the planner to adhere to and produce a plan based on the constraints given. One of the ways that the planner takes in a constraint is based on a chain of Task Space Regions. Task Space Region specify the region in which the arm is supposed to operate in and the planner constraints the trajectory to fit into this space. Since we needed a smooth sweeping motion, these regions are stitched together to form a chain of task spaces and by constraining the motion into this space,we get a smooth sweeping action. This planner then takes in the task space regions and samples points in that region while generating the RRT tree. Since it only samples the points within the given regions, it is able to produce paths that are strictly constrained to the plane that we are working on. This planner is much slower than the snap planner but its more robust. The CBiRRT planner is able account for different start configurations and also uncertainties in the initial positions.

3.3.3 RRT

The last planner that we tried was the built-in RRT planner. The RRT planner was reasonably fast but produced paths that were too complicated and the trajectory took a long time to execute. This planner however was required to initialize the position before we began our normal operation sequence. The advantage of this planner is that given a goal configuration that is not in an intersection with another object, it rarely fails to converge to a valid plan. The disadvantage is that it takes a long time to plan and the plan may intersect objects in the real environment that were not modelled in the simulated environment.

3.4 Implementation Considerations

Thought the CBiRRT was decent in producing plans, it was still too slow for our applications. Because of our specific knowledge of the problem at hand, we knew there were a discrete number of trajectories that would be sufficient in solving our problem statement. Our region of interest was a quarter of a circle. In order to have a sufficiently fast reaction time, we actually precomputed 90 trajectories for each angle of such a region. In order to execute precomputed trajectories, we had to use both the aforementioned Snap Planner and RRT. Namely, the Snap Planner was used to quickly move to a position near the table and the RRT planner was used to move the arm into the correct exact start configuration. It’s important that the start configuration be the same as the start configuration in the pre-computed trajectories, otherwise the trajectory will fail to execute. This is namely because a trajectory is a simple list of configurations over time; therefore, if the initial configuration is different than the current configuration, the robot won’t be able to execute the trajectory.

3.5 Cozmo Behavior

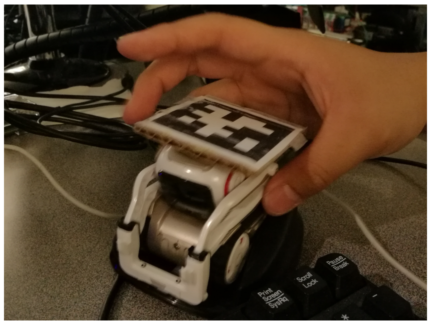

Figure 4: The Anki Cozmo has an APRILTag placed atop it to allow for smooth tracking. It was programmed using the SDK via an iOS phone.

The other main component of the project was programming the Anki Cozmo to move away from the arm in an attempt to escape the arm’s workspace. The group had initially planned on using multiple

Anki Cozmo's but opted against it once we determined the arm's maximum speed is only sufficient to keep pace with one Cozmo when the Cozmo is moving at its minimum programmable speed. Furthermore, due to difficulties in moving the robotic arm, we opted against using a full 360 degree workspace for the arm, instead limiting the workspace to a 90 degree area. As such, the workspace was too small to accommodate multiple Cozmos.

The basic behavior of the Cozmo was to have it move in a straight line in a direction away from the base of the arm. The control of Cozmo was basically through Cozmo SDK and there are plenty of API in motion control. We basically used DriveStraight and TurnInPlace classes. Initially we gave the Cozmo a random starting angle and made it move in straight line, when it was detected and pushed back by the arm, Cozmo would first pick another random angle by itself and keep escaping again. Also because of the pushing back behavior by the arm, new random angle would be introduced to Cozmo, which increased the chance for escaping.

After implementing this simple random escaping behaviors, we indeed considered doing an adversarial approach in which the Cozmo tries to move in a manner that prevents the arm from blocking its passage out of the workspace. However, we ran into two problems which forced us to keep the Cozmo planning simple. Namely, Cozmo requires Python 3.5.1 version whereas our whole system using ROS supported python 2.7, so it's not easy to combine Cozmo part with the other part using ROS; and we found a Cozmo ROS driver suitable to solve this problem but we only had one day left before presentation, so we just used current behaviors and kept Cozmo a independent subsystem.

4 Results

Overall, we were able to combine these subsystems and technologies to form a working system that met all of our goals. We were able to use the perception system in order to track the Cozmo robot as it moved about our workspace. Once it is about to leave the "safety region", our Assistive Dexterous Arm was commanded to move to the point where the robot was heading and sweep the robot back towards the center. Our arm was able to do this quickly and indefinitely. We could theoretically have the anki moving on a random path and leave it for hours. The arm would always make sure that it would be staying inside of the desired area the entire time. Given that these results match our criteria outlined during the project proposal, we view this project as a success.

4.1 Challenges

While the project may have been successful, it was not without it's challenges. Challenges ranged from big fundamental things such as understanding the software, to smaller-yet-shows-topping issues such as the lighting in the room.

One of the smaller issues that was not necessarily related to robotics had to do with our perception system. We had gotten the APRILTags software working and tested APRILTag tracking and logic. When we went to implemented this in our final setup, nothing was working. When we went back to testing, however, the APRILTags seemed to work. We determined the paper we printed the APRILTags on was causing glare from the overhead lights in the Personal Robotics Lab, and this only manifested itself as a problem when the APRILTags were facing the ceiling in their final configuration. After much experimenting, this glare problem was remedied by placing opaque scotch tape atop the APRILTags.

Other challenges included the time and effort it took to setup the environment both in simulation and real life. The setup and programming in the simulation took us longer than expected mainly due to the various dependencies. At one point of time, we decided that rather than having the full environment in simulation, we had a simpler setup in simulation and moved our results to the real arm. This is when we realized that the correlation between the simulation and real arm was not good. After shifting to the real arm, we had to rethink most of the system design through experimentation with the OpenRave environment and APIs.

The challenge on the Cozmo side was mainly due to version conflicts between Cozmo app and Cozmo SDK. Previously we had already finished all the motion-planning on Cozmo, but Anki released its new version of app that forced both Android and iOS users to upgrade to the newest version just two days before our presentation. This caused an incompatibility error between the app and SDK. Also, when following the guidance on the official site to upgrade the SDK in order to match the newest version of app, we only upgraded some of the packages in the SDK while others were still the old version. After seeking help from Cozmo technicians, we finally made Cozmo controllable via the SDK again.

4.2 Future Work

There is a lot of opportunity to continue the project, implementing more robust swiping strategies that can be extended to multiple Anki Cozmos. One issue we had for the project was that we only had access to one Anki Cozmo. We had automatically upgraded one of the Cozmos by accident while connecting to the Internet. Due to some issues with compatibility with the latest SDK, we were told to not upgrade the other Cozmos. As a result, we only had access to one Cozmo for this project.

That being said, the project could have been extended to multiple Cozmos. To have accomplished this, we would have had to extend the workspace of the arm from using only a quarter circle, to using a full 360 degrees around the arm. By doing so, multiple Cozmos would have the space to roam around the workspace area. In order to have an effective system, we would have to implement a stop-and-go strategy for the Cozmos, as the arm could barely keep up speed with one Cozmo at the lowest speed.

In addition, we would have liked to have implemented a more adversarial strategy between the Anki Cozmo and the ADA arm. Namely, the Cozmos should be able to coordinate with one another to try to trick the arm into escaping, while the arm uses a more intelligent strategy than simply swiping when the Cozmo reaches past a threshold. This also implies that the arm would have a more dynamic planner rather than simply performing one of two precomputed swiping motion.

5 Work Division

Task	Owner
Setup workstation computer with setup for ROS, OpenRave and RViz.	Nima and Yoga
Setup the environment with ADA, the table and cube kinbodies to simulate Cozmo.	Ben and Nima
Setup the ceiling mounted camera.	Michael
Get pose of cube from April tag detection.	Yoga and Michael
Setup Cozmo workspace and interfaces for control.	Kai
Capture image of the table and the Cozmo cubes from the ceiling-mounted camera.	Yoga and Michael
Detect location and orientation of Cozmos from the April tags.	Yoga and Michael
Add 1 static Cozmo to the scene and write planner for ADA to push the Cozmo to the center of the table.	Ben and Nima
Move Cozmos in straight lines	Kai
Calculate velocity vectors for Cozmo using a few image captures.	Yoga and Michael
Determine if Cozmo is in "danger."	Yoga
Add 1 moving Cozmo to the scene and write planner for ADA to push the Cozmo to the center of the table.	Nima
Program real Anki Cozmo to keep moving in straight line and turn in place.	Kai
Migrate all code to the non-simulated ADA and Cozmos. Test to ensure system works as expected.	All

6 Demo Link

https://www.youtube.com/watch?v=tzoztPCbwhk

References

[1] A. Menon, B. Cohen, M. Likhachev, "Motion Planning for Smooth Pickup of Moving Objects," in Robotics and Automation (ICRA), 2014 IEEE International Conference on, 453-460

[2] Erdmann, M. & Lozano-Pérez, "On Multiple Moving Objects," in T. Algorithmica (1987) 2: 477.

[3] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner, "Task Space Regions: A Framework for Pose-Constrained Manipulation Planning," in The International Journal of Robotics Research, 2011.

[4] Edwin Olson, "AprilTag: A robust and flexible visual fiducial system," in the IEEE International Conference on Robotics and Automation, 2011.

[5] Personal Robotics Lab, CMU GitHub Repository, https://github.com/personalrobotics/

Code

[1] April Tracking Github Repository, https://github.com/LitterBot2017/Robotic-Babysitter-April-Tracker

[2] ADA Arm Controller Github Repository, https://github.com/LitterBot2017/Robotic-Babysitter-ADA