The Tentacle Arm: Control of a High-DOF Manipulator

Jonathan Coens, Student
Professor David S. Touretzky, Advisor

2009 Senior Thesis
Computer Science Department
Carnegie Mellon University, Pittsburgh, PA

Abstract

Effective control of a high degree-of-freedom [DOF] manipulator increases in computational complexity with the number of joints. A controller for a many-joint arm must include algorithms for an inverse kinematics solver, a path planner, and object manipulation strategies. These algorithms have been developed for a planar "tentacle" arm composed of eight Robotis Dynamixel AX-12 servos in series. The goal is to be able to manipulate objects in real time. With the algorithms in place, a variety of operations are demonstrated on objects of different sizes using an actual tentacle arm. The work will be incorporated into the Tekkotsu robot programming framework.

I. Introduction

A. The Manipulation Task

Traditional robotic arms/manipulators utilize a handful of degrees-of-freedom to move about their environments. In environments with few obstacles, these manipulators effectively accomplish many tasks, e.g., the space shuttle's Canada Arm, with four degrees-of-freedom, deploys satellites, and various industrial robot arms perform painting, welding, and assembly tasks. However, in environments with numerous obstacles that hinder the manipulator's reach, the lack of mobility due to low degrees of freedom severely limits these manipulators from completing the same tasks. For example, a manipulator with three degrees of freedom cannot easily maneuver around three consecutive obstacles. With more degrees of freedom, however, manipulators can reach around multiple obstacles to reach a position that simpler manipulators could not. In the case of an arm reaching inside a narrow tunnel, a high degree of freedom arm can follow the tunnel's curvature with better precision than a lower degree of freedom arm, leading to a farther reach. This has applications for robots being able to reach inside of rubble, manipulate the internals of a computer, or perform surgery without interfering with the surrounding tissue. However, controlling such a high degree-of-freedom manipulator is a computationally demanding task. We developed algorithms for effective control of a particular class of high degree-of-freedom manipulators: planar tentacle arms.

B. Planar Tentacle

Tentacles in nature, as in an octopus arm or an elephant's trunk, are manipulators that have high degrees of freedom and are constructed in consecutive rotational joints. Using this as inspiration, we created a tentacle arm consisting of consecutive servos in series. To reduce the dimensionality, the servos all rotate in the same plane, fixing the arm's mobility to a single plane. See Figure 1 for an example.

- ***Hardware***

To run the system on real hardware, we constructed an eight-link tentacle hand-eye system from Robotis Dynamixel AX-12 servos, an acrylic (later metal) mast, and a Logitech webcam. The mast was fixed to the table using a C-clamp. The tentacle arm attached to the base of the mast as shown in Figure 1, and the webcam to a pan/tilt mount at the top.

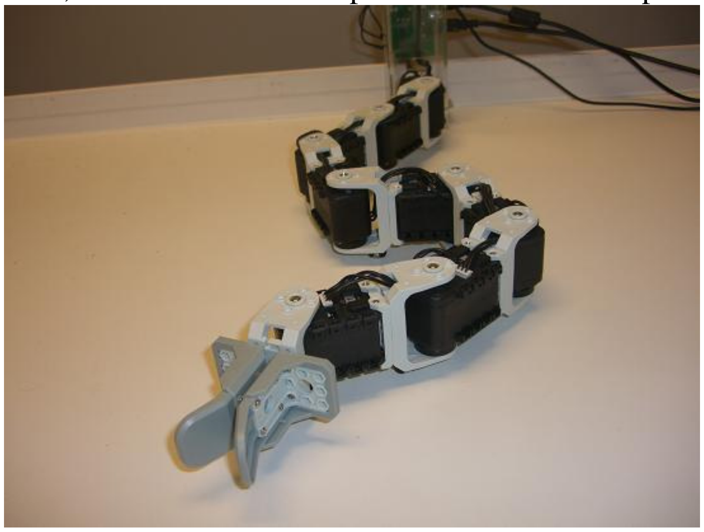

Figure 1: The planar tentacle arm with a dual-finger end effector.

- ***Tekkotsu***

The Tekkotsu software framework [4] is a robot-independent platform to abstract logic of robotic systems, such as vision and motor control, from the hardware. For example, the vision system used to control an Aibo's camera can run on any robot with a camera. This leads to easy portability of color segmentation, object detection, and other hardware independent tasks. We use Tekkotsu to control the arm as well as the webcam on a pan/tilt. The webcam allows for dynamic obstacle detection in the plane of the tentacle using Tekkotsu's dual-coding vision system [5]. Furthermore, all kinematic data (how many joints exist as well as the geometries of the links) are easily accessible through the framework, leading to easily portable algorithms from one arm to another.

- ***Requirements***

In order to manipulate its environment, a manipulator must solve the inverse kinematics problem (described in section II), be able to path plan, and utilize an environment-manipulation strategy depending on the type of obstacles perceived. Since the algorithms must work on a real robot, physical constraints exist: joints have a limited range of motion, links of the arm may not collide with obstacles in its working environment (workspace), and the arm must remain inside the workspace. The inverse kinematics algorithm generates a particular configuration of the arm that satisfies all physical constraints and places the end effector at a particular location. The path planner generates a trajectory of configurations from one state to another without violating any physical constraints. Lastly, a manipulation strategy must decide how to move objects in the workspace using the previous two components as tools. The goal is to have a system that skillfully manipulates objects in real time without any human intervention.

II. Inverse Kinematics (IK)

Given the location and orientation of the base of the arm and a collection of obstacles in the workspace, what joint angles are required to place the end effector at a desired location and orientation? Answering this question analytically is possible for arms with two joints. The two solutions are known as "elbow-up" and "elbow-down" solutions, as seen in Figure 2.

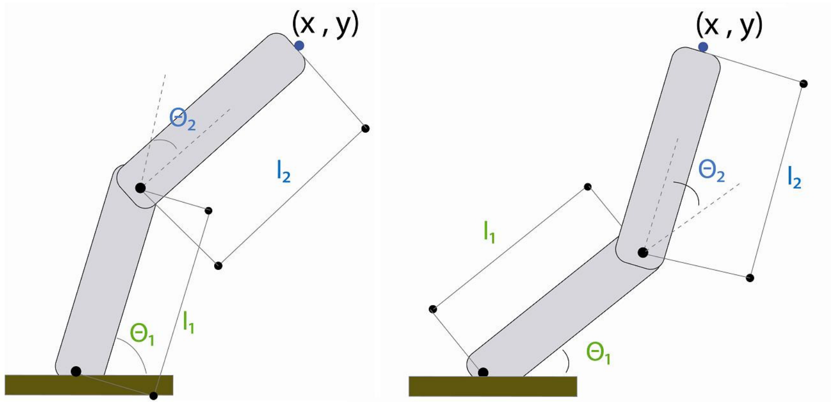

Figure 2: Two-link IK solutions in "elbow-up" on the left and "elbow-down" on the right

Using trigonometry, two pairs of joint angles θ_1 and θ_2 can be computed to allow the arm to place the end effector at the desired location of (x, y). From these two solutions, it is computationally simple to test whether either of these solutions meets all constraints: no joint exceeding physical limits, no two links colliding, no link colliding with an obstacle, and no link leaving the bounds of the workspace. For tentacle arms with higher degrees-of-freedom, more than two consecutive rotational joints lead to an infinite number of valid solutions. However, it is computationally infeasible to test all of these possible solutions until finding one that meets all constraints. Therefore, a different strategy for finding a solution is necessary. Sections A through D explain and analyze our strategy for finding a solution on a high degree-of-freedom arm that obeys all constraints.

A. Previous Work

• The Swan Neck

Solutions to fundamental robotics problems can sometimes be found by observing nature. Hayashi and Kuipers [1] drew inspiration from the motion of a swan's neck to propose an algorithm for controlling high degree-of-freedom arms. A swan uses its neck in a continuous fashion: as the head extends along a path, the segments behind it move so as to follow the same trajectory the head took. This approach was mimicked by Hayashi and Kuipers [1] by moving the tip of a simulated high-DOF arm composed of fixed-length segments to a goal location in an obstructed workspace. The algorithm assumed that the arm was initially wound in a spiral around its base. It decomposed the free space into a collection of connected rectangular regions, and used graph search to construct a path from the base to the goal while minimizing overall curvature. It then fit the arm to the path by dividing the path into equal-length segments and dividing the arm into pairs of links, and analytically solving each two-link IK problem. The arm could then unwrap itself from the base and reach the goal location by moving the tip along the path and constraining subsequent joints to follow the tip's trajectory.

The swan neck approach is inefficient for moving between arbitrary workspace locations because the arm must rewrap itself around the base before proceeding out to the new goal. Other solutions to the IK problem have therefore been sought. But Hayashi and Kuipers' curve decomposition idea has merit, and we have generalized it in our implementation, described in Section B.

• Relaxation-Based Approach

The first attempt at solving IK for the planar tentacle arm used a relaxation approach suggested by Professor Howie Choset. To trace a minimum-energy path in the workspace from the base of the arm to the goal using gradient descent, we defined a potential field with repulsive obstacles and an attractor at the goal location. Starting at the base of the arm in the potential field, trace the continuous path of least resistance, avoiding obstacles, until arriving at the attractor. The number of points on the path was a function of the length, and was usually several times greater than the number of links in the arm. Since gradient descent is susceptible to local minima, the strength of the attractor was increased whenever the algorithm became stuck. With this path defined, we used a relaxation process to alter the shape of the path until its length matched that of the arm. This relaxation process utilized two opposing forces. One directed a point to move toward the midpoint of its neighbors; this reduced both the curvature and overall length of the path. The second force moved points away from obstacles, which could increase the curvature and path length. The relative strengths of these two forces were adjusted depending on whether the path needed to grow or shrink.

Once the path was at its target length, the arm was fitted to the path by starting at the base and swinging the first link until it intersected the path. This link was then pinned, and the algorithm proceeded to recursively fit the remainder of the arm to the remainder of the path. This heuristic did not always succeed because the fixed-length arm segments could not exactly fit a continuously curved path. If the end-effector undershot or overshot the goal, we adjusted the target path length by a proportion of this error. We ran the relaxation process once again, and then re-attempted to fit the arm to the new path. If multiple iterations were unsuccessful at placing the end-effector at the goal location, the algorithm reported failure.

Figure 3 shows a solution produced by this algorithm. Its main strength is that it tries to produce low curvature solutions while maximizing distance from all obstacles. However, it does not always succeed. One reason is that following the potential field gradient to detour around an obstacle might not lead to a viable path of the required length; the only viable solution may lie on the other side of the obstacle. Another problem is that certain obstacle configurations could lead the relaxation algorithm to produce high curvature paths that exceed the turning limits of the joints, which the algorithm had no way to correct.

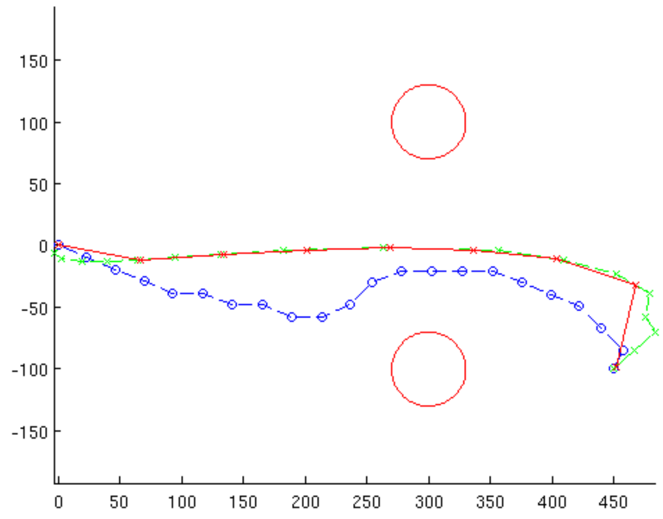

Figure 3: A sample solution from the initial relaxation algorithm. The base of the arm is at (0,0). The blue line is the gradient descent path, the green line is the relaxed curve of desired length, and the red line is the final link configuration found by fitting the arm to the relaxed curve.

B. New Hybrid Approach

Our new hybrid approach builds on the curve segmentation idea of Hayashi and Kuipers. Ignoring obstacles and joint angle limits, curve segmentation can put the end effector at any location that is within reach. We defined an initial curve using the same gradient descent approach as before, fitting the links along this curve using curve segmentation. In an attempt to avoid obstacle collisions and respect joint turning limits, we chose the appropriate non-colliding elbow-up or elbow-down solution to each two-link IK solution if one existed.

Upon implementing this approach, a few shortcomings became apparent. If the gradient descent path was too long, some of the two-link IK problems had no solution. To combat this, we generalized the curve segmentation approach to explore a richer space of partitions of the arm segments by considering compound links instead of just single links. Another problem was that choosing between elbow-up and elbow-down solutions to each two-link IK problem was not sufficient to avoid all obstacle collisions. However, a collision-free solution can be produced by swinging obstructed links out of the way and using recursive decomposition to re-fit the remaining links. Finally, if the gradient descent path was short, the solution produced by curve segmentation could contain sharp angles that exceeded the turning limits of some of the joints. Introducing a post-processing step with a new relaxation strategy that enforces joint turning limit constraints would ensure a valid solution. In the following paragraphs, each step of the above algorithm is described in more detail.

• Generating Structures

The Hayashi and Kuipers curve segmentation method utilized a simple pair-wise partitioning of the arm. For an eight-link manipulator we denote this partition as (1,1)(1,1)(1,1)(1,1). We generalized this idea to search a richer combinatorial space that includes asymmetric partitions. For example, the configuration (2,1)(2,1)(1,1), shown in Figure 4, contains two (2,1) structures, which means the first link contains two arm segments (the joint-angle between them is fixed to have no bend) and the second link contains one segment. The minimal partitioning (4,4), where both links are composed of four arm segments, is shown in Figure 5.

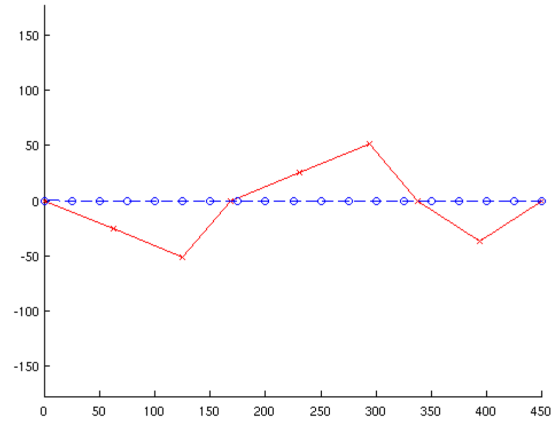
Figure 4: The asymmetric partitioning (2,1)(2,1)(1,1) contains a mixture of one-segment and two-segment links. The gradient descent path is shown in blue and the arm in red.

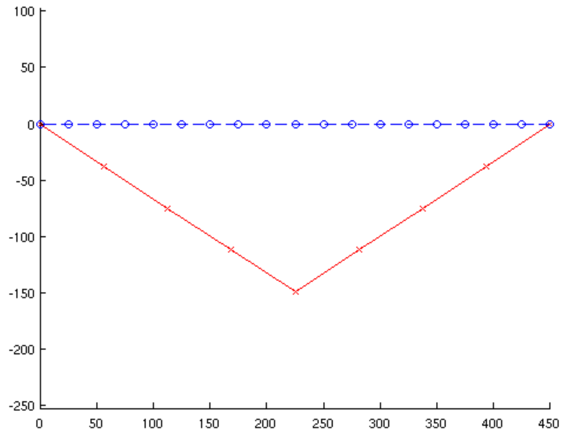
Figure 5: The minimal partitioning (4,4).

A partitioning can fail if there are unresolvable collisions, or if joint turning limits are exceeded and post-processing relaxation is unable to correct the problem. If a partitioning fails, a new one is tried. By initially ignoring collisions and joint limits, fitting the links via curve segmentation can quickly determine if a candidate partitioning has any chance of placing the end effector at the goal location.

- **Collision Resolution**

If both of a two-link IK problem's elbow-up and elbow-down solutions collide with obstacles, the offending links are marked but generating the arm configuration continues as if there were no collision. To resolve these, we start with the distal link that collides with an obstacle, pin the distal end of this link, detach the proximal end, and rotate the link until it is free of the obstacle. This is now a new IK problem to solve: fitting the remainder of the arm to a new curve resulting from re-running the gradient descent with the goal location being the proximal end of the link that was rotated. The algorithm succeeding in finding a solution to this sub-problem assures a collision-free solution to the original problem. In the event that the sub-problem cannot be solved, we rotate the colliding link in the opposite direction until it is again free of the obstacle to attempt to solve the new sub-problem. If this also fails, the partitioning is rejected.

Figure 6 illustrates this process. The blue line is the path initially found by gradient descent. (The path doubles back on itself due to a local minimum in the potential field that must be overcome by increasing the gain of the attractor component.) The green line is the original (4,4) partitioning, in which link A collides with one of the obstacles. The collision resolution algorithm swings link A clear of the obstacle and then generates a gradient descent path (shown in magenta) to the new goal location. The remainder of the arm is then recursively fit to this magenta path, producing a (3,3) partitioning that reaches the new goal. Combining this solution with the distal portion of the original produces the final solution shown in red, where link B is the new position of link A.

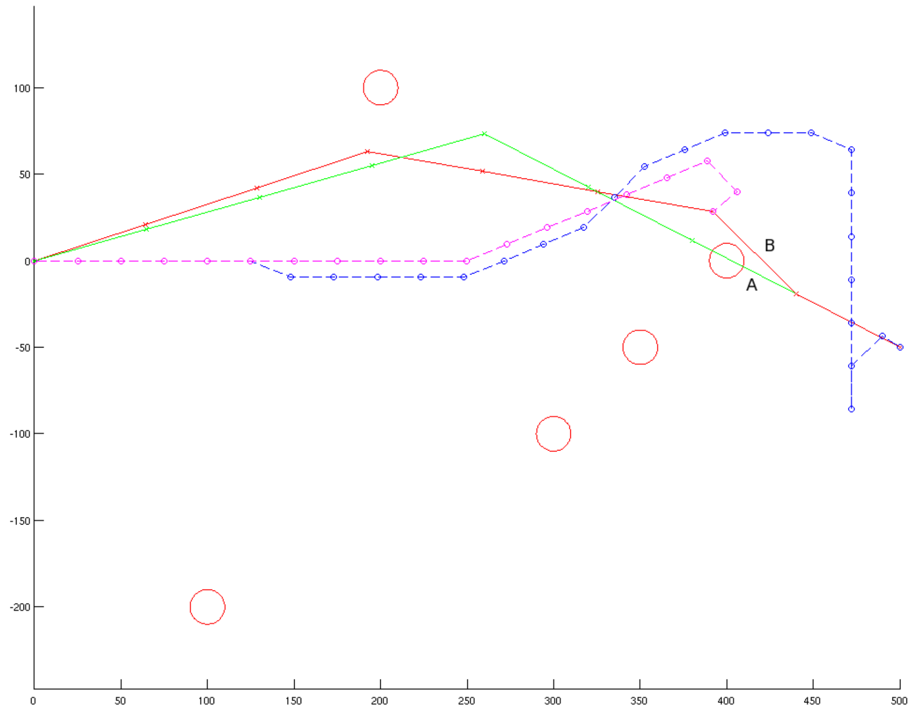

Figure 6: Successfully resolving an obstacle collision.

- **Post-Processing Relaxation Step**

The analytic portion of the hybrid algorithm is rapid, but it may produce solutions with steeper joint angles than the arm is physically capable of. We introduced a post-processing relaxation step in which all joints simultaneously adjust their positions to try to resolve any joint limit violations. Four forces act on all the joints. (1) If a joint's angle exceeds its limits, the two links that meet at that joint are rotated outward by a small amount. Doing so alters the lengths of these links, which remain connected to the rest of the arm. (2) If a link is not at its correct length, its two endpoints are shifted slightly inward or outward, as appropriate. Doing so alters the angles of the joints at those endpoints. (3) If a link passes too close to an obstacle, both of its endpoints are moved slightly away from the obstacle. (4) If two non-consecutive links pass too close to each other, both links' points are moved away from one another. Although these forces may, at times, work against each other, the overall effect is to adjust the arm configuration toward an equilibrium point where all constraints are satisfied. Figure 7 illustrates this process. A (4,4) partitioning with a sharp elbow angle (red line) is relaxed by moving other joints outward so that the middle joint can move inward and assume a more shallow angle (green line).

The relaxation may not terminate because the constraints may not be solvable. However, empirically when a solution exists, it is found in fewer than 200 iterations, so the process is cut off at that point, rejecting that partitioning.

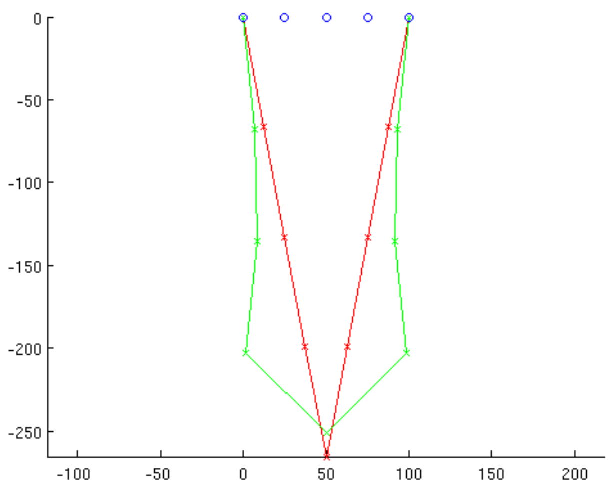

Figure 7: A successfully resolved joint angle violation. The red line corresponds to the initially generated (4,4) structure with a very sharp turn at the elbow, while the green line shows the relaxed configuration.

C. Analysis

• Behavior

Figure 8 shows a path through a tightly constrained workspace where the arm must remain relatively straight to pass through a bottleneck, so the majority of the length accommodation occurs in the distal segments. The solution started with a (1,1)(1,1)(1,1)(1,1) partitioning and found a collision in the first segment that generated a seven-link sub-problem. This sub-problem used a (2,2)(2,1) partitioning and found a collision in the first link, resulting in a six-link sub-problem that was solved with a (1,1)(1,1)(1,1) partitioning with a collision in the third link. The resulting three-link sub-problem was solved with a (2,1) partitioning, and this was the only sub-problem that required relaxation.

In Figure 9, the goal location is close to the arm, but the obstacles restrict the arm’s ability to bend. The solution resulted from a (1,1)(1,1)(1,1)(1,1) partitioning with a collision at the fourth link that was resolved by moving that link, successfully solving the remaining three link sub-problem, and then relaxing the entire configuration. Figure 10 shows a solution where the arm winds around an obstacle in order to reach the target. The shorter green line shows that relaxation was invoked for a sub-problem after a collision resolution step, but was not required for the main problem.

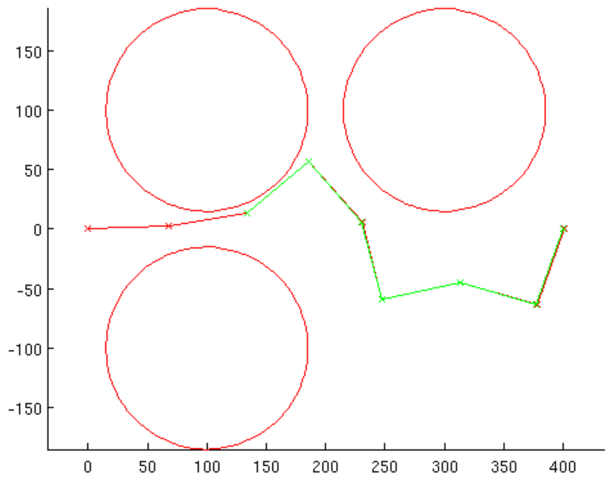

Figure 8: Finding a solution in a heavily constrained workspace.

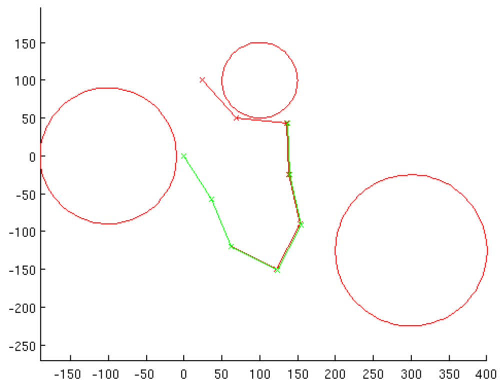

Figure 9: A solution resolving collisions and exceeded joint limits.

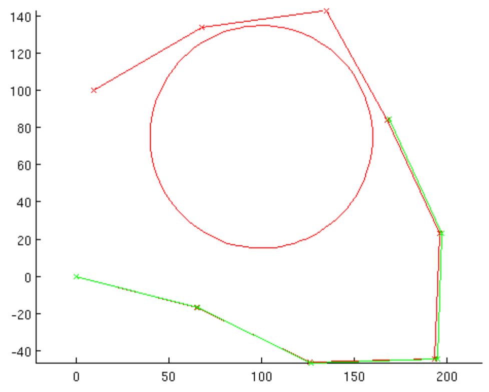

Figure 10: A solution

• Comparison with Initial Relaxation Algorithm

In the initial algorithm, the correlation between changing the length of the overall path and changing the curvature led to unrecoverable situations. The new hybrid approach does not have the same faults, but presently it does not maximize distance to obstacles or find minimal curvature solutions. This tradeoff can be seen in the solution of the same problem by both algorithms: the first attempt in Figure 11 and the new hybrid approach in Figure 12.

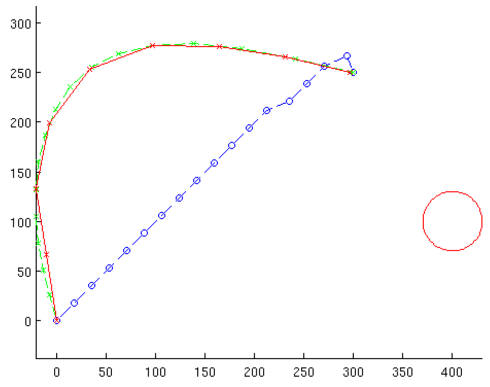

Figure 11: Successful solution to this obstacle configuration and goal location found by the first path planning algorithm. Note the smooth continuous properties of the final configuration.

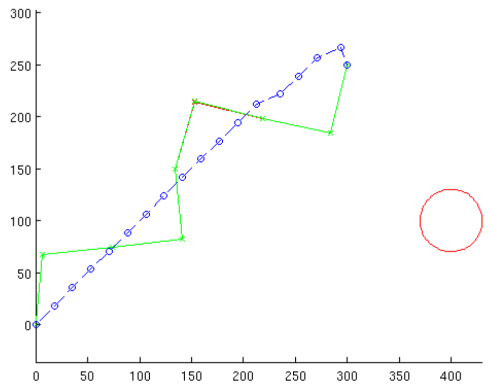

Figure 12: Successful solution via the second, hybrid approach. Note the accordion shape of the final configuration: this is a higher curvature solution.

However, the new hybrid approach often succeeds where the first approach fails. Using the same gradient walk, Figure 13 shows the solution generated by the hybrid approach that goes between the obstacles, where the first approach would fail to bow out. Similarly, Figure 14 shows a solution that successfully resolves a curve with high curvature that the first approach could not solve.

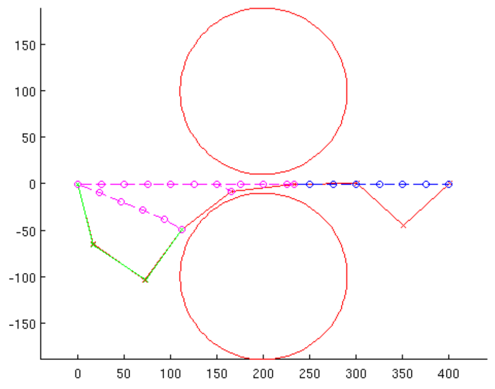

Figure 13: A configuration which the hybrid solution solves that the first attempt could not due to failure to utilize free-space (space not inside obstacles).

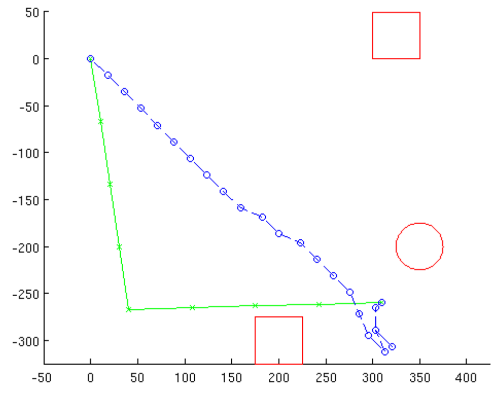

Figure 14: A solution that resolves a curve with high points of curvature.

D. Implementation

The algorithm was first prototyped in MATLAB, allowing for easier graphical representation of the algorithm's actions. It was then ported to C++ as an extension to Tekkotsu. The C++ version runs orders of magnitude faster than the MATLAB prototype due to the C++ version running natively. In worst case, the algorithm takes no more than ten seconds to complete on a Pentium 4 machine.

E. Assumptions

This hybrid-approach algorithm exploits the assumption that all of the obstacles are convex. If all of the obstacles are not convex, then the gradient descent step could fail to define a path from the start to the goal. This is acceptable since all of the links in use are rectangular and all obstacles perceived are circular. Furthermore, each link is represented as a line in the algorithm but physically has width. This is overcome by bloating every obstacle by half the width of the widest link. If a segment does not intersect a bloated obstacle, then it is impossible for the link that corresponds to that segment to collide with the actual obstacle.

III. Path Planning

Given two arm configurations, a start and an end, and a collection of obstacles, what trajectory of joint configurations can move the arm from the start to the end without causing any collisions? Many analytic solutions exist for solving this problem, such as computing the line-of-sight graph or Voronoi diagrams, but these require substantial pre-processing on the workspace. To path plan in a dynamic environment in real time, the employed algorithm is not guaranteed to find a solution, but the algorithm works well in practice.

A. Rapidly-Exploring Random Trees (RRTs)

• Generating RRTs

We opted to use rapidly-exploring random trees (RRTs), developed by LaValle and Kuffner [2]. Our algorithm to generate RRTs is a randomized algorithm that attempts to explore as much of the configuration-space (the space of all possible configurations of the arm) as possible when attempting to find a path. Obstacles in the arm's environment have clearly defined boundaries, but since each joint is rotational, the obstacle's projection in configuration-space is a complex shape. Computing these complex obstacle boundaries in configuration-space is a taxing procedure, so the RRT algorithm forgoes this step through randomization. First, we create two trees, one rooted at the start configuration and the other at the desired end configuration. On each iteration, the algorithm generates random configurations R until it finds one that does not intersect any obstacles. Then it finds the closest configuration S to R in the tree rooted at the start. It then creates a new node N that is a small step from S toward R, and connects N to S in the tree. Next, it finds the closest configuration T in the tree rooted at the goal, and creates a new node M that is a small step from T toward N, connecting M to T in its tree. See Figure 15.

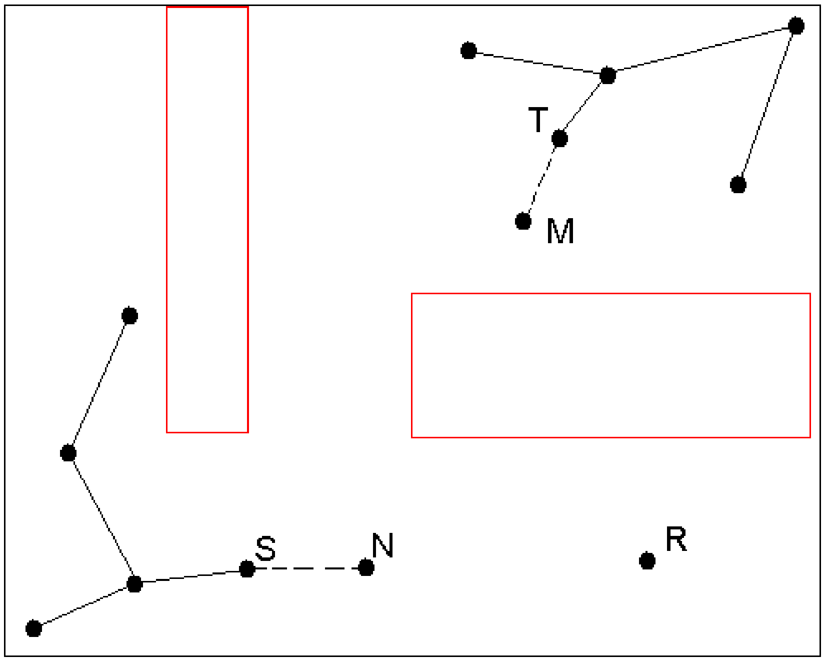

Figure 15: A 2D joint space single RRT iteration; see text for explanation. Our tentacle works in 8D joint space.

If either N or M collides with an obstacle, the algorithm discards N or M and proceeds to the next iteration. Finally, the algorithm runs a second iteration with the roles of the two trees swapped. By interpolating between nodes, the algorithm moves each joint a constant fraction of its linear error from the desired node's corresponding joint. In two-space, this is drawing a short segment (such as the dotted lines in Figure 15) toward the other point. Repeating this process of gradually stepping each tree toward the randomly generated configurations as well as each other will eventually lead to a node that will be reachable from both trees. Following the paths along both of these trees from the roots to this node yields a valid non-colliding path.

This process builds trees in configuration-space whose generated configurations are guaranteed to neither collide with obstacles by themselves, nor collide when transitioning between connected configurations due to the small steps between those configurations. Extracting a collision-free trajectory for an arm to travel along requires a single walk per tree, which is simple. Furthermore, it does not require any preprocessing of the workspace or configuration-space, which is ideal for a dynamic environment.

- **Exploiting Biases**

In order to bias the generation of the trees to more quickly and efficiently explore the configuration space for a tentacle arm, we exploit both the methods of finding the closest configuration in a tree to some other configuration and expanding a configuration toward some other configuration. When finding the closest configuration in a tree to some other configuration, we use a custom error function to compare configurations and take the closest one as the one with smallest error. An ideal error function would be the total amount of new workspace area covered by one configuration that isn't covered in the other. Unfortunately, this is a computationally expensive operation, so we must use a heuristic that successfully biases the majority of the configuration space. On average for a tentacle arm in a random configuration, moving joints early in the chain is more likely to drastically change the workspace area of the arm than moving joints further along the chain. Calculating the difference between corresponding joints of two configurations and weighting the difference of joints closer to the base higher than joints closer to the end effector successfully exploits this concept. Even though there are cases where this fails, such as a configuration that puts the end effector directly next to the base of the arm, it works well enough in practice.

Furthermore, we can alter the method of expanding a node to move toward another node to explore more of the configuration-space per step. Instead of taking a single small step from a configuration toward the other and stopping, we continue to take consecutive small steps toward the other configuration until a configuration with a collision is generated or the target configuration is reached, connecting each step as a chain in the tree. LaValle and Kuffner call this the RRT-Connect method, which has the property of searching longer paths down the configuration space than the original RRT growth method. This helps by requiring fewer iterations to explore heavily constrained environments, yielding a speed up in time. Since the tentacle arm could potentially work in heavily constrained environments, we use this RRT growth variant for all tree expansions.

- **Post-processing Step**

There is no guarantee that a generated path from the start configuration to the end configuration is optimal. In the scheme of generating random configurations to grow toward, a path may cause the arm to take steps that do not contribute to moving closer to the end configuration. In an effort to root out these meaningless steps, we perform a post-processing step on the path. Over a number of iterations proportional to the total number of steps in the path, we randomly pick two configurations in the path and see if a direct interpolation between the two paths is possible, and if so, replace these steps with the direct interpolation, eliminating any detours. Although still not guaranteed to be optimal, choosing enough pairs of configurations to short cut yields a new path that takes fewer unnecessary steps.

B. Implementation (C++)

The implementation of this algorithm is written in C++. The algorithm works for tentacle arms of an arbitrary number of joints and any rectangular geometric links. To optimize it for use with the actual tentacle arm we constructed, we compared two methods for computing the closest configuration in a tree: a brute-force search and an approximate nearest neighbor (ANN) algorithm [1]. For all cases considered where the arm had a potential path from start to end, the brute-force approach was approximately twice as fast as the ANN algorithm. Thus, calculating the closest configuration in a tree is done via brute force. Furthermore, we pre-allocated a pool of nodes to avoid the costs of memory management mid-iteration. This leads to a faster run time and allows the same memory space to be used in multiple consecutive runs of the algorithm for a lesser re-run time. Lastly, the collision detection methods were specifically optimized for any pair of rectangular or circular obstacles. Since the robot has rectangular links and perceives circular obstacles, this covers all possible collision cases.

C. Analysis

The RRT approach is an excellent choice for path planning with tentacle arms. It is good for high dimensionality since the algorithm linearly depends on dimensionality. This means that as more joints are used in a tentacle arm, which increases the dimensionality, the algorithm remains scalable. Furthermore, the RRT approach is probabilistically complete; given enough time and space, the algorithm will find a path if one exists. This guarantee is desirable, but depending on how constrained the workspace is, real-world time and space constraints could run out before a path is found. We empirically chose a cutoff level that usually finds a path if one exists. It is also worth noting that of all portions of the manipulation task, this step takes the most amount of time. Further optimizations to this step would greatly help the overall performance of the system.

IV. Manipulation

A. Maintain Contact

Given methods for solving inverse kinematics and path planning, we implemented a strategy of choosing particular configurations and paths to achieve some goal. The tentacle has a dual fingered end effector (see Figures 1 and 16), so to move an object from one position in the workspace to another, the path the arm takes must place the end effector in contact with the object and maintain contact with that object while moving in order to push it to a desired location. One way to do this is to enforce this constraint on the configurations added to the RRT in the path planning step.

The constraint must be introduced every time the RRT interpolates between nodes to ensure that all paths generated in the RRT satisfy the constraint. Therefore, we needed a generalized procedure to satisfy the contact constraint between two configurations. Given the two configurations, where the arm is moving from and where the arm is moving to, it is easy to compute the trajectory of the end effector in the workspace. Maintaining contact with the obstacle along this trajectory necessitates that the angle of the end effector is almost perpendicular to the direction of movement. Figure 16 shows a situation where the arm needs to move to the right, so the end effector's workspace angle points almost straight down.

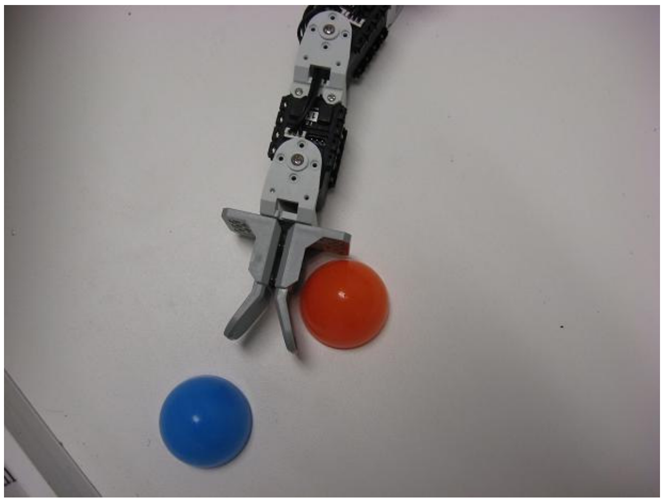
Figure 16: The end-effector maintaining contact with an object

However, there is no guarantee that the trajectory between the two configurations will align with the end effector's orientation in either configuration. Therefore, we introduce new configurations in between the original two configurations being interpolated. Using an analytic IK solver employing only the last three joints of the arm, we generate configurations that rotate the end effector about the object until its angle is almost perpendicular to the direction of travel. If a desired orientation is unattainable, we reject the interpolation. After generating the configuration with an orientation close enough to perpendicular to the direction of travel, we attempt to change the orientation of the configuration being interpolated to in the RRT step to be the same as the direction of travel. This ensures that the end effector will keep an appropriate relative angle to the direction of travel during movement. If that orientation is unattainable, however, we then reject the interpolation. If these steps succeed, they guarantee that any two connected nodes accepted into the RRT maintain the desired contact constraint.

B. Shortcomings

Upon implementing this approach, it became apparent that this strategy over-constrained the allowable configuration space for the RRT. In order to find a valid path in an acceptable amount of time, the allowable discrepancy between the direction of travel for the end effector and its relative angle needed to be wider than the allowable discrepancy of the physical dual-finger manipulator. Attempting to find a path using the dual-finger's allowable discrepancy rarely succeeded in finding a path. Furthermore, introducing obstacles in the workspace for situations where a path was found also rarely found a path in an acceptable amount of time. Both of these observations suggest that introducing the contact constraint on the RRT for a high dimensionality arm over-constrains the algorithm. Observing the algorithm's behavior confirmed that this is due to the majority of interpolations using the RRT-Connect method failing to reach their goal configurations.

The over-constraint problem is likely due to the combination of having high dimensionality for the arm as well as only allowing a small range of discrepancy in end effector angle and direction of travel. In future work, using an end effector with a larger allowable discrepancy for the tentacle arm may allow this strategy to succeed. Unfortunately, we did not have time to complete this. Furthermore, alternative strategies for maintaining the contact constraint over a path should be considered.

V. Conclusions

A. System Wide Analysis

Despite the shortcomings of the implemented manipulation strategies, the system effectively allows for dynamic path planning to place the end effector in any reachable position. Even in heavily constrained environments, the system effectively maneuvers around multiple obstacles. In our implementation, the path planning stage takes orders of magnitude more time than the inverse kinematics stage. However, in the worst case the combination of inverse kinematics and path planning takes no more than a minute. This is an acceptable level of real-time behavior, but improvements to the path planning stage could greatly improve overall running time.

The system can effectively perform tasks if non-static end effectors are used. For example, given a closable gripper, the manipulator could obtain form closure on an object and move it around the work space. With an end effector that maintains form closure, there would be no need to maintain the contact constraint during the path planning stage, which the system can handle. This task seems menial, but it can be achieved in heavily constrained environments.

B. Future Work

• 3D

This system works in a two-dimensional world, so a natural extension is to expand these concepts into three dimensions. The path planning step is easily extended to three dimensions since only the computational geometry changes. However, the inverse kinematics and manipulation strategies may require more substantial changes to effectively work in three dimensions. If such a system existed, many new tasks could easily be accomplished, such as using a robotic arm to build a block tower or dynamically grasp objects along their centers of gravity. For example, Professor Howie Choset's snake-robots effectively work in 3D.

• IK and Path Planning Interaction

In some cases, the inverse kinematics algorithm's solution to place an end effector at a desired location may lead to heavy computation in the path planning stage. The IK algorithm could be biased to incorporate the current state of the arm in how it generates a solution configuration. Ideally this would lead to a more reliable real-time system that aptly finds inverse kinematics solutions and reduces the amount of time required to path plan.

• The Arm Itself as an Effector

Instead of relying on a unique end effector for manipulation techniques, the arm itself could be used as a manipulator. To effectively move a ball around the workspace, the arm could wrap around the ball with the distal joints (as seen in Figure 17) and use the remaining joints to move the effector.

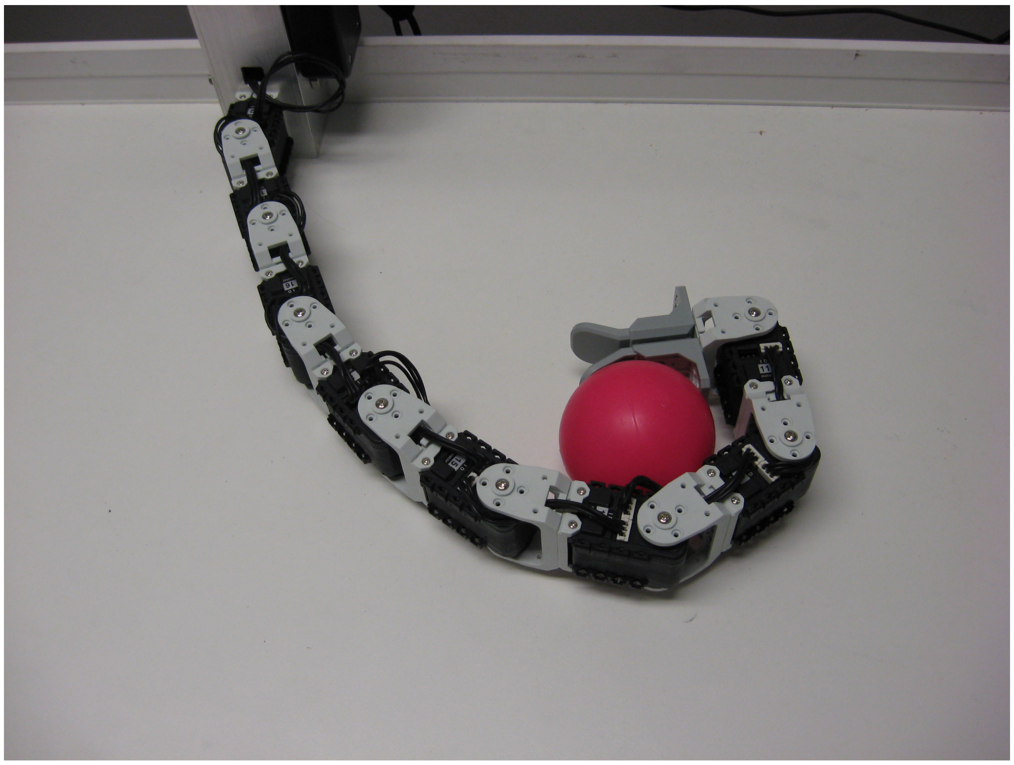

Figure 17: Using the distal joints as an effector to wrap a ball.

VI. Acknowledgements

I would like to thank Professor Howie Choset for helping me identify the IK problem and suggesting the relaxation-based curve fitting approach; Ethan Tira-Thompson for help with implementing my algorithms in the Tekkotsu software framework on the real arm; and Professor Dave Touretzky for introducing me to the tentacle arm and guiding me through its implementation.

References

[1] A. Hayashi and B. Kuipers, "Path Planning for Highly Redundant Manipulators using a Continuous Model," in *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, Cambridge, MA: AAAI/MIT Press, 1991.
[2] J. Kuffner and S. LaValle, *RRT-Connect: An Efficient Approach to Single-Query Path Planning*. 2000.
[3] D. Mount and S. Arya, *ANN: A Library for Approximate Nearest Neighbor Searching*. University of Maryland Department of Computer Science, http://www.cs.umd.edu/~mount/ANN/
[4] Tekkotsu home page: http://tekkotsu.org
[5] D. S. Touretzky, N. S. Halelamien, E. J. Tira-Thompson, J. J. Wales, and K. Usui, *Dual-coding representations for robot vision programming in Tekkotsu*, Auton Robot (2007) 22:425-435