Surgical Motion Planning for Variable Curvature Needle Systems

Charles Hong

Abstract

In this work, a novel algorithm for needles of variable curvature is presented. The approach utilizes basic geometric motion primitives with a combined dynamic programming approach with the look-ahead algorithm. A heuristic function is derived to relate the work being done to the brain with the motion primitives. The function can be easily changed depending on additional experimental classification. Compared to other methods dominating the field, I present a non-sampling motion planning approach capable of 2D planning. Obstacle avoidance is conducted with an A* approach with a visibility graph to generate possible virtual waypoints and a path is found utilizing these waypoints as guides. A 3D algorithm based on 2D algorithm is proposed.

Index Terms

Motion Planning, Surgical Robotics: Needle Steering, Ribbons, Lorenzo, Appa, Mapo Tofu

I. Introduction

Minimally invasive surgery is used for many different operations as it leads to better patient outcomes. The current research uses bevel-tip needles and programmable bevel-tipped needles [1-2]. These devices are constrained by constant or variable curvature.

A. Motion Planning for Bevel Tipped Needles of One Curvature

Ron Alterovitz is the leader in motion planning for bevel-tipped needles. Janine Hoelscher's thesis, a PhD student under Dr. Alterovitz, highlights many current state-of-the-art methods [3]. The problem is constrained with very simple motion primitives based on the needle design. Webster's work describes bevel-tipped needles as a non-holonomic Dubins car [1]. This means the needle can only move far left, far right, and straight. With these constraints, a sampling approach is viewed as the most popular method to determine if a path is possible. The thesis highlights RRT and RRT* with backward path planning to easily find a solution. RRT can guarantee completeness and optimality based on the resolution of the RRT. It can search through infinite space and combinations of the motion primitives to find a solution. Alterovitz mainly focuses on RRT as a starting point and tries to optimize the search space to find effective strategies for needle planning. Personal Thoughts: The use of RRT to guide motion planning for bevel-tipped needles is a very good solution with limited primitive selection. However, for increases in motion primitives, planning becomes exponentially longer with the RRT method. Furthermore, the curvature changes or inflection points are not necessarily added to the heuristic functions. Changes in curvature or action can be changed with every action.

B. Motion Planning for Bevel Tipped Needles of Multiple Curvature

Ferdinando Rodriguez y Baena is the principal investigator for the Eden 2020 project, a needle with multiple different curvature constraints [2]. The solution focuses on using the RRT and heuristics to provide the best path. The current method uses an Adaptive Fractal Tree, a two-tree system. The first tree will try to find the best solution and the second tree with a finer resolution will search for a more efficient path [4]. It is adaptive in the sense it will find solutions near the end. Personal Thoughts: The device does not take advantage of the multiple different curvature possibilities. Still RRT reliant for motion planning.

II. Methods

The following methods section is formatted chronologically to explain the thought processes behind identifying a possible algorithm for variable curvatures.

A. Constraints

The needle used for the work is the tendon-driven ribbon device. The ribbon is inserted and will generate a crack. The crack will follow the shape of the robot when inserted. However, there are multiple problems with the current robot. The tendons require a large pulling force for higher curvature curves. In the previous work, the relationship between the force and curvature was found [5]. Changes in the crack orientation are difficult as a new crack needs to be made whenever the tendon changes direction. Lastly, the setting for the tendon operation is the brain, to reduce damage to the brain, the path length should be reduced. Ultimately, the constraints can be summarized as the following: Path length is minimized (brain damage is reduced), curvature changes are minimized (crack generation is difficult), and low curvature paths are preferred (less force needed to make the curve). All these constraints tie into work on the brain which for this study is equivalent to damage. The last thing that should be mentioned is the path must be continuous and the waypoints are already assigned.

B. Bezier Curves

Initially, Bezier curve methodology was thought to be an efficient solution to the problem. However, Bezier curves only work to validate continuous curvature constraints. In other words, the path is continuous. However, my takeaway from studying Bezier is all paths can be described geometrically.

$$P * (1 - t) + P1 * t \tag{1}$$

This is a parametrized Bezier function that can be used to find the x and y coordinates. However, as mentioned previously the problem with the Bezier curves is the lack of constant curvature based on the control equation used. The control equation of the Bezier function will push and pull the trajectory using control points. For the planning of the ribbon, we want the paths to go through the actual waypoints. As a result, Bezier functions should not be used for path-solving.

C. Motion Primitives

While Bezier curves are not useful, it is good to think about the problem mathematically. The primary goal is to determine possible ways to interact with two waypoints and connect them with the most limited amount of curvature changes possible. Based on these constraints, I determined that 2 line segments were the maximum number of line segments used to connect 2 points. Based on this, here are the developed motion primitives which can be further illustrated in Fig. 1.

1) Connecting 2 Points: There are 2 ways to connect 2 points. Both will lead to infinite solutions.

I. The first solution utilizes one line to connect the two points in a curve. The center of the curve must always lie on the centerline of the two points. However, this centerline has no limit leading to infinite solutions. An example is showcased in Fig. 1 MP I, where multiple arcs can connect 2 different points. There is no way to distinguish which arc should used to connect the 2 points based on the information.

II. The second solution relies on a curve and a straight line. If you have 2 points and 2 lines you have an infinite solution as you have essentially 2 variables for 1 solution. However, there is one caveat, the created arc cannot have the endpoint inside the arc. This just means you cannot make a curve so that the line never intersects the point.

2) Connecting 2 Points with an initial or final trajectory : Adding an initial or final trajectory (tangency imposed at the initial or endpoint) constrains the solution set of the previously described motion primitives.

III. When we add the trajectory constraint to the one curve MP there is only one solution. The solution can be found mathematically as two rules need to be followed: The center point of the arc must lie on the perpendicular of the trajectory, as this will ensure tangency between line segments, and the center point must lie on the midsection line between the two points being connected, this was previously defined in the Connecting 2 Points section. Two line segments will intersect and give one solution.

IV. The second solution is very similar to solution II. The only change is that there is an added constraint that the position of the center point of the arc must lie on the perpendicular of the trajectory. Infinite solutions still exist as the size of the curve can change easily.

3) Connecting 2 Points with an initial AND final trajectory or Connecting 3 Points: This section highlights potential MP that were not considered for the current algorithm due to the added complexity.

V. There is one or no solution to connect two points that have two existing trajectories. A line extending the shorter side is drawn and then connected with an arc.

VI. There is one solution to connect 3 points, however the trajectories from the points are predetermined. This MP is the most efficient solution to solve the problem. However, the use cases are limited as the MP constrains the other paths which may not lead to optimality outside of the primitive.

Typically, VI and V will need to be used together to complete trajectories. As a result, expanding the current algorithm to include these MP will be an interesting continuation of the work. The algorithm that I proposed but not yet implemented will utilize MP VI to find connections between sets of three points throughout the workspace. Afterward, you will create a path from the starting waypoint with a predefined trajectory. MP V. will be utilized to connect the two existing trajectories and the algorithm will just fill in the gaps.

D. Look Ahead Algorithm

Previous works utilized a backward planning methodology to find a path. With the developed motion primitives (I - IV) going in the forward direction is the most logical as the initial starting trajectory will be known. This will immediately constrain the problem and we can primarily build the path using MP III and MP IV. Unfortunately, there are infinite variations of MP

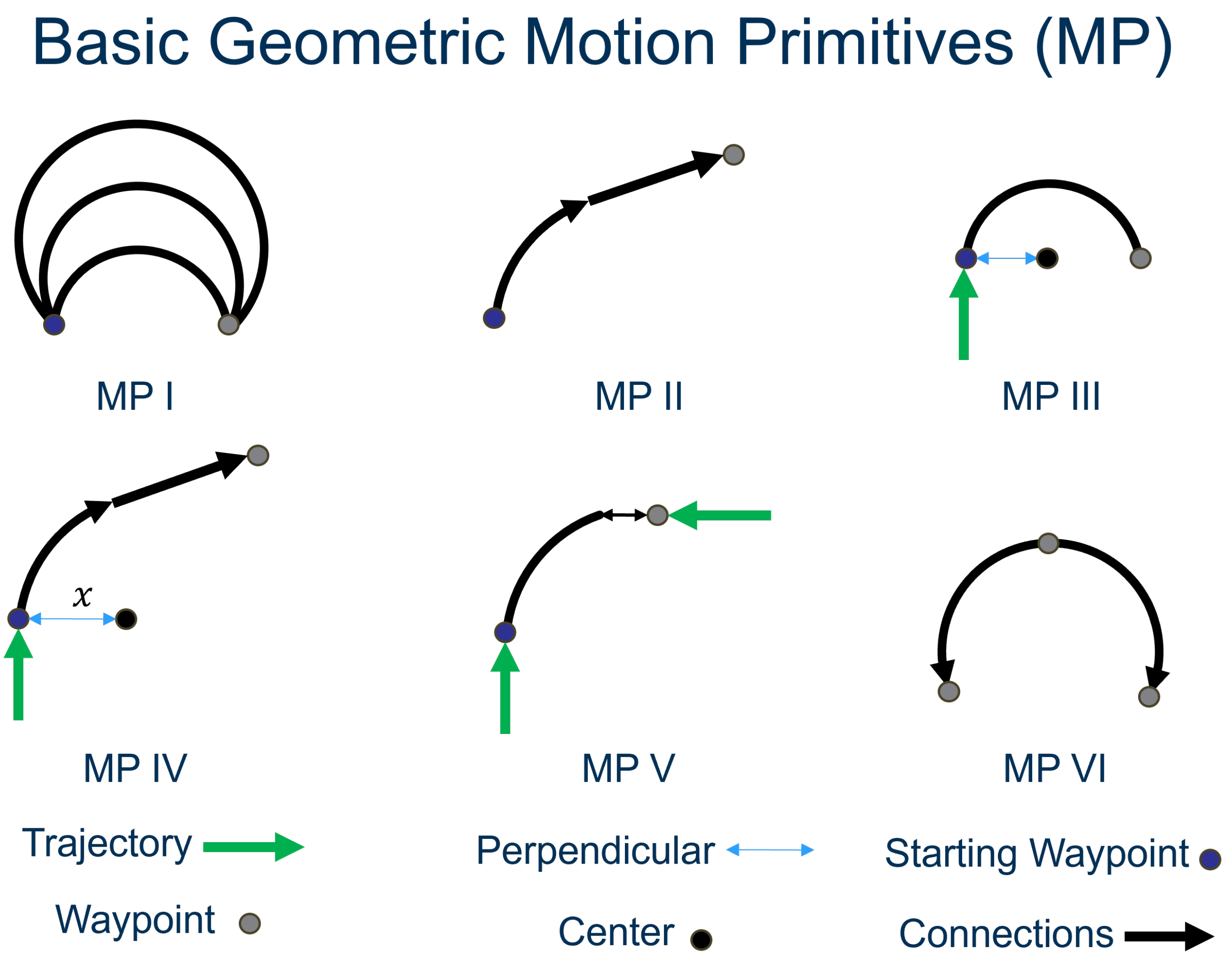

Fig. 1. All the motion primitives and how they are constructed.

IV which can be used to connect two different points. However, the variations of MP IV lead to a different output trajectory from the ending waypoint. I identified this as the key aspect to make the trajectory as it further constrains the problem. It is more suitable to pick a trajectory that is in line with a third waypoint, the waypoint you want to navigate to after connecting 2 points. This can be shown in Fig. 2. The correct terminology for the paper is we are conducting a dynamic programming approach by segmenting all the waypoints in the problem into problems of three waypoints. From there, we conduct a look ahead algorithm, where we observe three waypoints. We will change the path connecting waypoints 1 and 2 to make a more optimized path to get to the third waypoint.

This sounds good in theory but leaves much to be desired as you need to decide which MP to use in different situations. To do this, we can use the end trajectory of the different MP. We have one confirmed trajectory by utilizing MP I. This allows for one limit but the other limit is not well defined. For this algorithm, we make the other limit a modified version of MP II which utilizes the smallest radius curve possible paired with a straight line. This will minimize the line length while making the robot turn and realign with the next points. The result of these two limits makes a cone of decisions shown in Fig. 2. Anything third point inside of the cone we can find an exact solution. We can mathematically solve this trajectory using MP V. If the third point falls outside the cone, we use either MP I or the modified MP II depending on the side the third point falls on.

E. End Path Solution

The look ahead algorithm with the motion primitives produces the majority of the path. Based on the algorithm, we assumed the path is the best path possible as it will make the most amount of straight lines and decrease the path length. However,

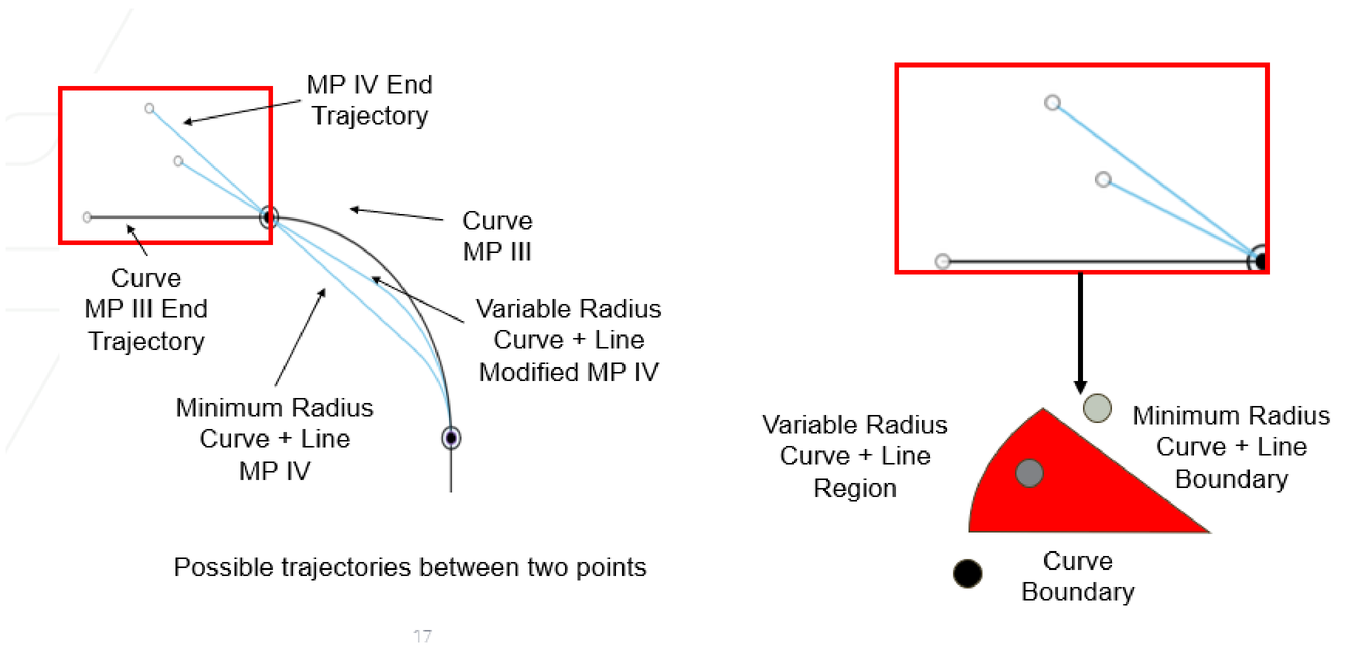
MP IV End Trajectory
Curve MP III
Curve MP III End Trajectory
Variable Radius Curve + Line Modified MP IV
Minimum Radius Curve + Line MP IV
Possible trajectories between two points
17
Variable Radius Curve + Line Region
Minimum Radius Curve + Line Boundary
Curve Boundary

Fig. 2. Visual representation of the algorithm and helps showcase the selection of which trajectory to pick.

the end path does not have a set trajectory. To solve this, a heuristic equation relating the motion primitives is needed to fully describe the path. The entire equation is based on the x, representing the center of the curve. The goal was to find an equation relating x to the work of the path. You can take the derivative of the function to minimize work based on x.

The first equation is to find the degrees of the curve. This is done by using the law of cosines between the end of the arc and beginning of the arc. This is represented in Fig. 7. We are assuming xFin and yFin are known values and the only thing that is changed is the position of x.

$$arcDegrees = (\arccos(\frac{xFin - |x|}{\sqrt{(xFin-x)^2 + yFin^2}}) - \arccos(\frac{|x|}{\sqrt{(xFin-x)^2 + yFin^2}})) * |x| \tag{2}$$

The second equation is to find the arc length based on the degrees previously found in the last equation. The way the code is setup is the line trajectory is on the y axis. As a result, we make a simplification and say that the absolute value of x can always be assumed to be the radius. As a result, we can simply multiply the arc degrees with the radius to find the actual arc length.

$$arcLength = arcDegrees * |x| \tag{3}$$

This is arguably the most important equation as it relates the curvature directly to force. Lorenzo, you can change the equation to whatever best represents the force data collected. The goal is to relate the curvature of the device to the force necessary to achieve certain curvatures. Currently, the equation written below is based on a quadratic trend line relating the mean of the previously collected ICRA data. The a,b, and c constants in the equation can be pulled from the ICRA paper and are showcased in Fig. 3. From my experimentation, 1/X should be the most dominant factor to determine the force.

$$forceBasedOnCurvature = \frac{-b + \sqrt{b^2 - 4 * a * (c - \frac{1}{|x|})}}{2 * a} \tag{4}$$

The straight line equation just utilizes the Pythagorean theorem to find the distance from the end of the produced curve to the point in space.

$$straightLine = \sqrt{((xFin - \cos(arcDegrees) * |x| + x))^2 + ((yFin - \sin(arcDegrees) * |x| + x))^2} \tag{5}$$

The final equation is utilized to equate the total work for one movement primitive. The equation combines a natural movement force needed to progress the device (tendonForce) which is added to the force needed to pull the device for a certain length of time.

$$totalWork = (straightLine + arcLength) * tendonForce + arcLength * forceBasedOnCurvature \tag{6}$$

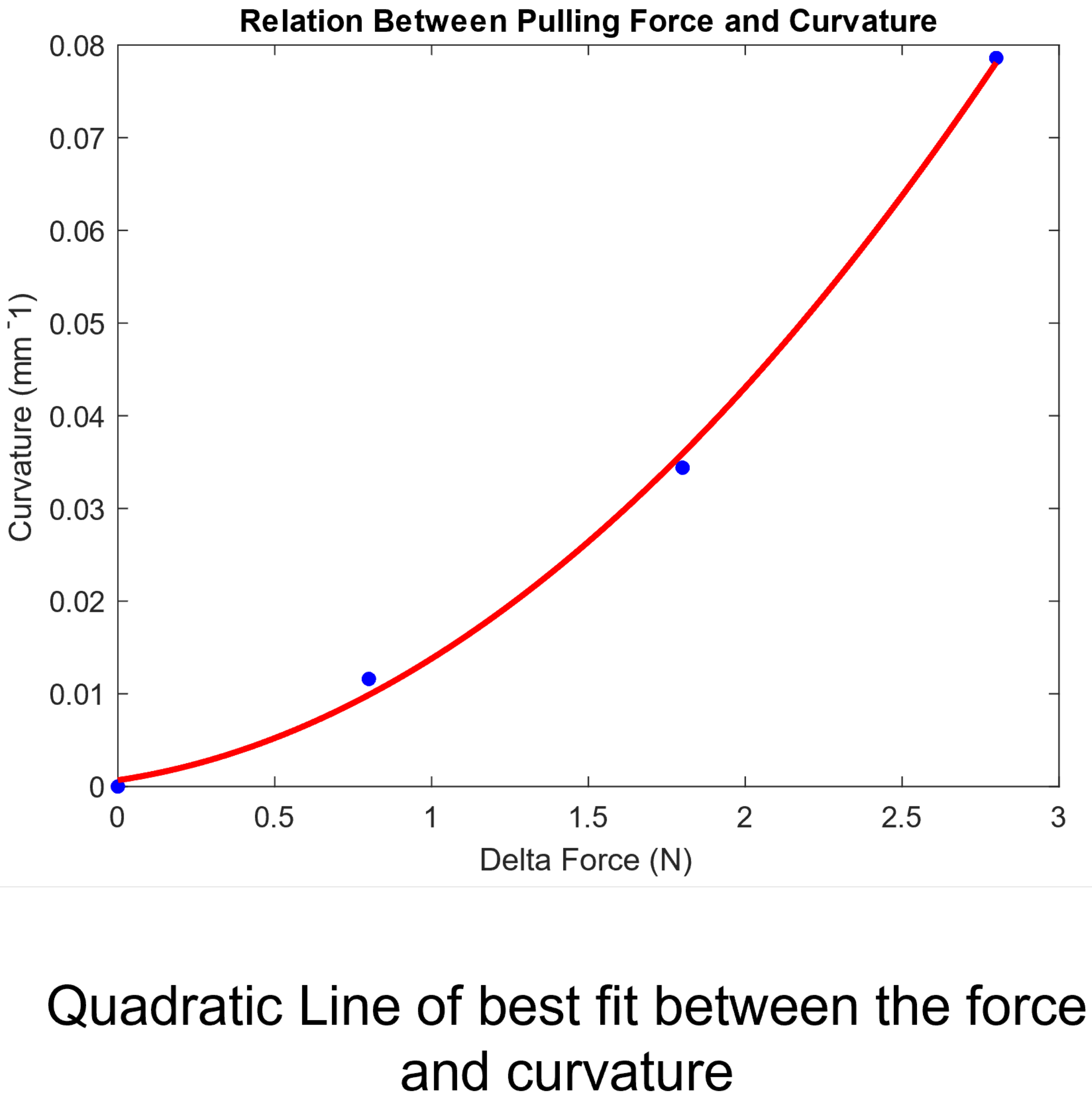

Fig. 3. The trend between force and curvature derived from the ICRA paper.

F. Closing Thoughts on 2D

A 2D trajectory algorithm has been developed based on the look-ahead algorithm paired with basic geometric MPs. There is a novelty within the needle path planning field with this type of methodology as it does not utilize a tree to make the path. Typically, RRT is resolution-reliant, however, this particular path planning is an algorithm capable of fully utilizing the full continuous range of curvatures provided by the flexible nature of the device. This algorithm can be extrapolated to other needle steering devices that are capable of variable curvatures when conducting surgery.

G. Obstacle Avoidance

This section speaks on the obstacle avoidance strategy. We make the assumption we are navigating around convex polygons. To find a path around the obstacles, a visibility graph between the beginning and end points is utilized. This can be showcased in Fig. A line is drawn between the starting waypoint and the end waypoint. The line sees if it interacts with any of the

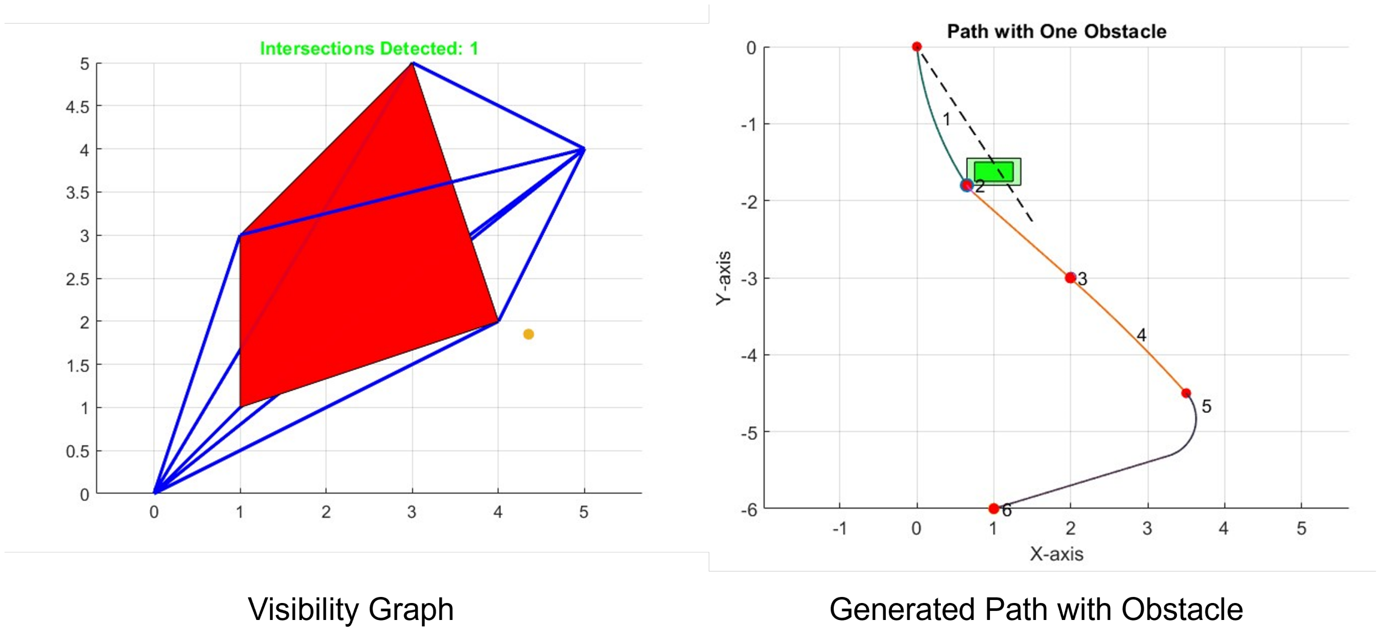

Fig. 4. Obstacle Avoidance Strategy.

existing polygons. If it does not, it progresses as usual. However, if there is an intersection, a visibility graph is made. Each corner of the obstacle becomes a new fake waypoint. Fake is classified as a choice where the path can utilize the position to avoid the obstacle but it is not necessary to go through the point. As a result, when the path goes through different obstacles, all the combinations of the corners the robot can go through are recorded. The algorithm tests all the permutations of the combinations and selects the best possible path around the obstacle based on the fake waypoint. This is just an A* algorithm as the algorithm looks to minimize the number of waypoints it goes through. The current method of selection is based on the heuristics with an additional cost associated with the additional change in curvature. As a result, the best result will typically be of low waypoint changes. The visibility graph and waypoint selection are showcased in Fig. 4.

H. 3D Path Planning with Obstacle Avoidance

The goal of this is to find a way to utilize the twisting mechanics of the ribbon. The current strategy is to turn the ribbon so it directly aligns with the 3 closest points in a plane. The ribbon will conduct 2D path planning. After going through the 3 points, the ribbon will conduct a turn a find a new plane to travel in for the rest of the points.

III. Code Segments

Here are the main scripts used to make the path planning. The main suggestion is just to recode everything for robustness. This is the task I will be completing but the goal of these segments is to help you better understand the process.

A. lookAheadAlgorithm.m

The lookAheadAlgorithm.m script provides the main functionality of the algorithm. It will make different MP choices as described in Section II E. This function takes in the following variables. The first is the initialPoint, the variable representing your starting waypoint. The next variable is the trajectory, a vector representing where the trajectory is starting from, the next variable called second point is the ending waypoint. Thirdpoint is the next waypoint you want to aim towards with the algorithm. Color1, Color2, and count are just variables to make the color of the lines whatever you want them to be. Count will take note of the line segments. lookAheadAlgorithm.m will try to test out all the different MP available. So it calls the MP code for the curve, the minimum radius curve with line MP, and sees where the third point lies. The location is checked with checkReferenceFrame to determine where the point lies. It will ultimately just pick the best MP with the algorithm.

B. mp1Traj2Points.m

This represents the first motion primitive where we are trying to connect 2 points with 1 line/curve. The function takes in the starting point, the previous trajectory, the end waypoint, a draw boolean to tell the function to output the lines, color for the

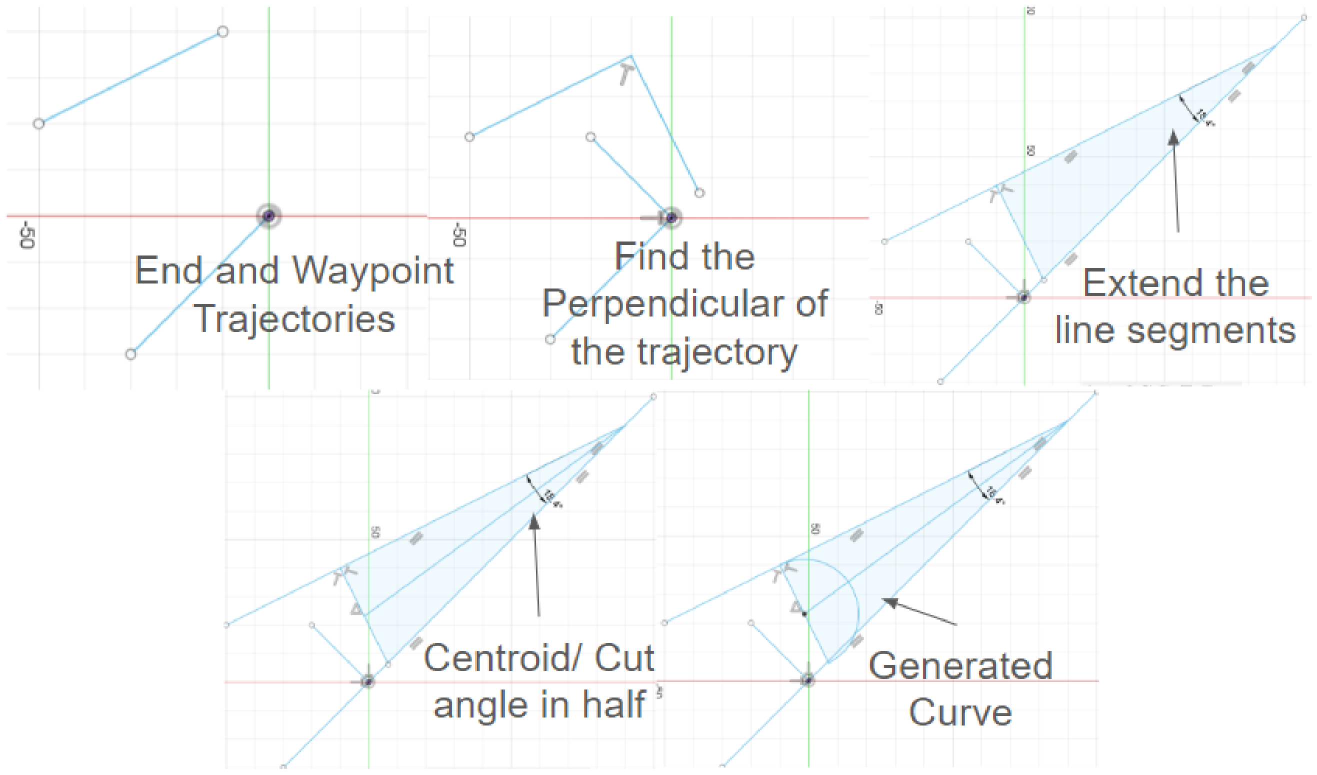
End and Waypoint Trajectories
Find the Perpendicular of the trajectory
Extend the line segments
Centroid/ Cut angle in half
Generated Curve
-50
50

Fig. 5. Two Tangent Trajectory Example to Solve.

line, and the number of lines already to keep track of curvature changes. The code for this section was chatGPT but the logic is as follows: The first part is to find the midpoint between the starting and ending waypoint. Afterwards, the perpendicular line of the trajectory is calculated. The position of the center will lie on the intersection between the two lines. Radius and center position are found from the intersection and the Pythagorean theorem is utilized to find the exact radius. However, while we have the center, the direction of the arc is unknown. We utilize another function checkClockwiseDirection.m, which takes the starting and ending waypoint, trajectory, center, and radius. A change in the reference frame is conducted to determine which side, +x or -x, the second waypoint is on. An arc is drawn depending on the position, reoriented, and output with the x and y coordinates. We now have the arc but we need to output the trajectory of the curve. To find the trajectory, it can be mathematically defined as the perpendicular of the endpoint to the center which is shown in Fig. .Based on the geometric relationship we can extrapolate the trajectory. There is one last problem, we need to figure out the direction of the trajectory, which is the start and end of the trajectory as this will define the next path. To do this, we search for the trajectory line closest to the created arc as we assume the trajectory line should be tangent to that arc. Ultimately the logic for this method can be summarized as the following: Find the placement of the center, find the direction of the arc (clockwise or counterclockwise), and determine the trajectory (pick the trajectory closest to the developed arc).

C. *twoTangentsTwoPointsFunction.m*

This code is the MP that mathematically finds the one solution if there are 2 points and 2 trajectories. The function is only used if the 3rd waypoint is in the cone or area of accessibility outlined by the When looking at the function the inputs are the following: point1 is the starting waypoint, end1 is the start of the trajectory, point2 is the end waypoint, and end2 is the most important variable here. end2 is the third waypoint in the look-ahead algorithm. Based on the third and second waypoint, this will form a straight line which is the lowest cost trajectory. The rest of the variables are for mapping out the trajectory. The beginning of the code will draw and label the trajectories to count the curvature. This should not be changed unless you would like to turn off the label. Afterward, line 36 starts the geometric solution to find the correct arc to join the two segments. The idea to connect the two segments is to extend one until a common radius curve can join the segments. The steps to do this are as follows: Find the equations of the lines between point1 to end1 and point2 to end2. Afterward, find the intersection between the two lines. Calculate the angle at the intersection and draw a line segment dividing the angle in half. When the line intersects a perpendicular line, half the length of the line will be the radius for the curve. You can extend the line which does not intersect the perpendicular. This can be showcased in Fig. 5. In the code I solved it poorly, but it follows a similar fashion. I utilized the perpendicular and drew both circles. I made the circles slightly bigger, so I can guarantee it crosses the line than the calculated radius, and found the intersection with the line using a polyxpolyy function that outputs

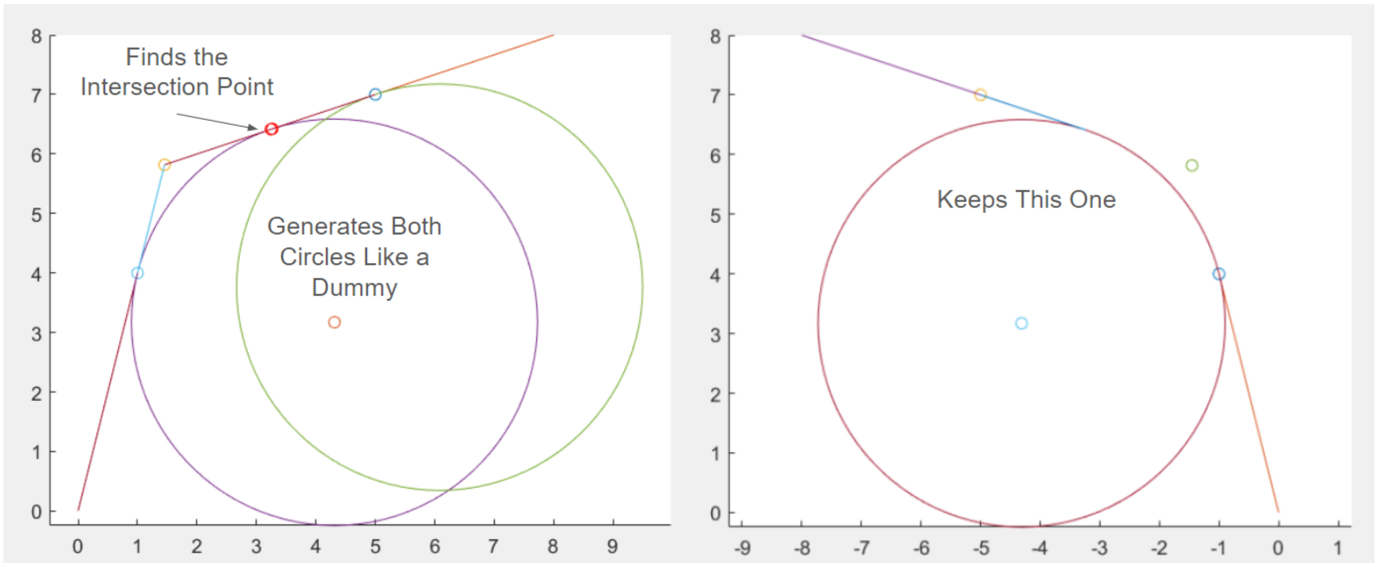

Fig. 6. Showcases the polyxpolyy marking and leaves the remaining circle it picks.

the intersections. The input of this function takes into the line segments and sees where they cross. This is a pretty bad way to do it and is highly inefficient. If I have the time, I will redo the script and have a better method to guarantee finding the solution between 2 tangents. But overall, if one circle intersects, we keep that circle which is shown in Fig. 6. There are places with zero solutions though, if the line segments never intersect or the extension of one line intercepts the other.

D. *minLineCirclePointGeneratorWithBaselineExactFixed.m*

There is a guaranteed solution which can be solved as follows. You must have the endpoint, the minimum radius, the start point, and the trajectory. Conduct a change of the reference frame to have the trajectory line up with the y-axis and the start point lies at the origin. In this position, the center of the curve will always lie on the x-axis at either the position or negative radius value (radius,0) or (-radius, 0). To find the center, check the position of the endpoint in the reference frame change determine if the x position is negative or positive, and pick the corresponding center. Now we solve for the internal arc angle through the following methodology as shown in Fig. 7: A line segment is drawn to connect the center and the external point, this will be the hypotenuse. A radius length segment forms the other part of the triangle. Based on this information, the arc cosine of the radius divided by the hypotenuse is used to find the internal angle of the triangle. This is the first angle that is needed to find the arc degree. The other arc angle needed can be derived by calculating the angle between the segment origin-center and center-endpoint. Subtract the two angles to find the arc degrees which gives all the information to find the solution.

E. *minLineCirclePointGeneratorWithFuture.m*

This represents a minimum radius MP IV. The goal of this function is to connect the starting waypoint and the ending waypoint with a curve that has the smallest radius. This function takes in a waypoint, the trajectory from a point, and the ending waypoint. It outputs the waypoint that was achieved, the next trajectory, and some other extraneous functions. The algorithm's first part defines the line between the start point and the trajectory. After a change in coordinate frame is conducted, this is used to find the position of the point that the algorithm wants to connect to. A simple check on the position of the end waypoint is found based on the x position. If positive, the center of the curve is on the right half while if negative the center is on the left. There is a section of old code that would randomly generate a center point for the trajectory. This left some remaining variables that can be removed and will be removed when I rewrite all the code. With a center and a set radius, the trajectory connecting the curve with a straight line with the point must be found. The circular arc is discretized into 10000 parts, this can be changed to have a finer resolution. The equation of a line is found by taking the position of the arc and then finding the line connecting the arc position to the center. If the perpendicular of this line is found, that is the trajectory from the arc that can be used to connect to the end waypoint. By discretizing the arc, the algorithm tries multiple line segments to connect the arc with the end waypoint. In this code, we have a threshold or range accepted to quantify a stable connection. This is the non-geometric way of conducting this planning as it relies on a guess and check method along with a resolution. This algorithm can miss a connection that should be possible. Another section of the code is finding the starting position of the angle, an angle calculation between vectors is used here. The rest of the code rotates the points around the reference frame to receive the original path.

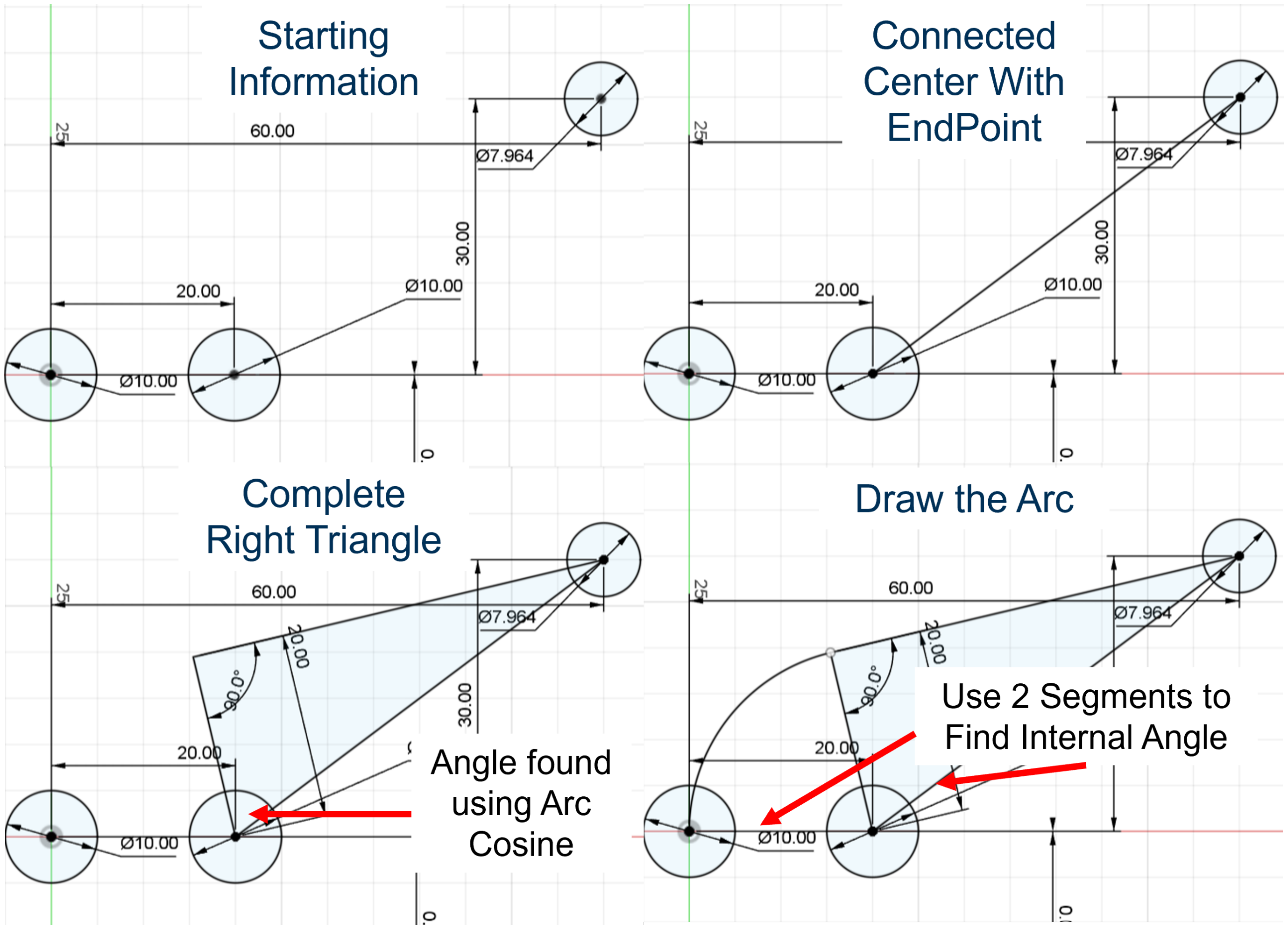

Fig. 7. Methodology to find the exact solution using a curve and a straight line.

F. *endPath.m*

This is a basic function that conducts a change in the coordinate frame to check the position of the final waypoint compared to the initial waypoint. It connects to another function called checkReferenceFrameOther.m that checks if the next waypoint is in the positive y or negative y. If negative y endPath.m will pick the minimum radius curve and straight line approach. If positive, a singular curve will connect the last points. There is probably a better way of doing this, but we originally agreed on this aspect so I am currently keeping it. The heuristic needs to be added but based on the current equations, the best path will be related to path length.

G. *waypointAdderUpdated.m*

We are trying to solve obstacle avoidance in this function. This function takes in all the obstacles vertices in one input with cell data structure formatting along with inputs for the start, end, trajectory, and a third point for the lookAheadAlgorithm. Another function is used called linePolygonIntersectionMultiObstacles.m which takes in the obstacles, the start and end positions, and finds if there is an intersection. The intersection is found by discretizing the line and determining if the line lies within the region specified by the vertices. This is a very efficient way to find intersections and resulted in the fewest issues. Based on the function output of intersectionPoints, numIntersections, firstObstacleIndex, and firstIntersectionPOint, we find the first obstacle collision. We scale the size of the obstacle using the scaling centroid function to make artificial waypoints and then we move clockwise or counterclockwise depending on the most efficient distance. The algorithm moves between the obstacles looking to see if there are any intersections or if there is an intersection with a new obstacle. If there is a new intersection with a different object, the algorithm moves forward to the next set of waypoints. The path traveled is recorded and saved as waypoints. The lookAheadAlgorithm.m is then conducted on all combinations of the false waypoints to find the most efficient algorithm which is defined as a cost function of distance with an added factor for the number of curvature changes. As a result, this algorithm represents a combined visibility graph algorithm with a version of A* I believe. Note you will need to fact-check this. There is a huge problem with this as the number of obstacles will increase the computation speed by a factorial depending on the number of point combinations that need to be sorted through.

IV. Future Work

A. Updated Algorithm

MP VI which connects three points with one curve. If an algorithm can be used to search the space to find three points capable of being connected this may lead to a more elegant solution. My original solution was to find the closest 3 waypoints to each other to form different trajectories. But this is not the best solution to find the connections. Find another student to study this as I think there is a very nice geometrical proof. The solution may lie in a certain range of radii that is suitable to use this MP.

The final thing is code transfer into Python. I do not use Python that much so this will be fun. But I will complete this. I will finish by January 1st.

V. Conclusion

Overall, I developed a novel algorithm that can be applied to needles capable of variable curvature constraints. Devices such as the Eden 2020 and the current tendon-driven ribbon robot. There will need to be work further done on the 3D implementation of the device but as Dr. Sakar said, this is a baseline algorithm that can be improved upon.

Acknowledgment

Thank you for being my mentor Lorenzo, I appreciate your effort and belief in my capabilities because I did not believe I could create something novel. The problem you gave me was completely out of my background and skill set but I tried my best to figure out a plausible and novel solution. I hope I gave you something you truly like and find proper to your taste. Cheers!

References

[1] Webster, R.J., Cowan, N.J., Chirikjian, G., Okamura, A.M. (2006). Nonholonomic Modeling of Needle Steering. In: Ang, M.H., Khatib, O. (eds) Experimental Robotics IX. Springer Tracts in Advanced Robotics, vol 21. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11552246$_4$

[2] Matheson, E.; Rodriguez y Baena, F. Biologically Inspired Surgical Needle Steering: Technology and Application of the Programmable Bevel-Tip Needle. Biomimetics 2020, 5, 68. https://doi.org/10.3390/biomimetics5040068

[3] Hoelscher, Janine. Multi-stage Motion Planning for Medical Steerable Needle Robots. 2024. https://doi.org/10.17615/8w55-zy13

[4] Pinzi, M., Galvan, S. Rodriguez y Baena, F. The Adaptive Hermite Fractal Tree (AHFT): a novel surgical 3D path planning approach with curvature and heading constraints. Int J CARS 14, 659–670 (2019). https://doi.org/10.1007/s11548-019-01923-3

[5] L. Noseda, A. Liu, L. Pancaldi and M. S. Sakar, ”A Flat Tendon-Driven Continuum Microrobot for Brain Interventions,” 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 2024, pp. 13466-13471, doi: 10.1109/ICRA57147.2024.10611399.