EN.530.663: Robot Motion Planning
Final Project

by Stefan Hustrulid, Xiyuan Wang and Nathan Van Damme

Task 1: Planar Serial Manipulator

1.1 Forward kinematics

In Task 1, a 4-link planar serial manipulator is used, which usually has multiple solutions due to under-constraints. The forward kinematics of the end effector position can be expressed as:

$$\begin{cases} x_e = l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) + l_3\cos(\theta_1+\theta_2+\theta_3) + l_4\cos(\theta_1+\theta_2+\theta_3+\theta_4) \\ y_e = l_1\sin(\theta_1) + l_2\sin(\theta_1+\theta_2) + l_3\sin(\theta_1+\theta_2+\theta_3) + l_4\sin(\theta_1+\theta_2+\theta_3+\theta_4) \end{cases}$$

1.2 Construct configuration space

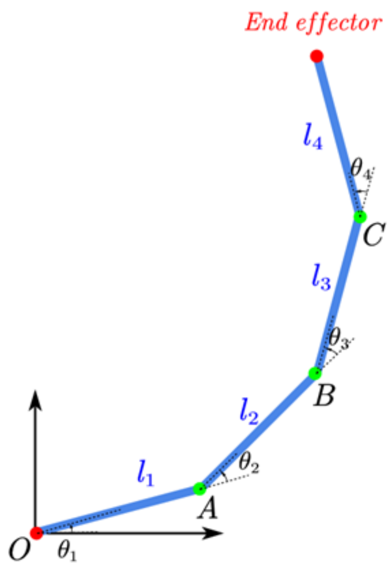

Figure 1: Structural Diagram

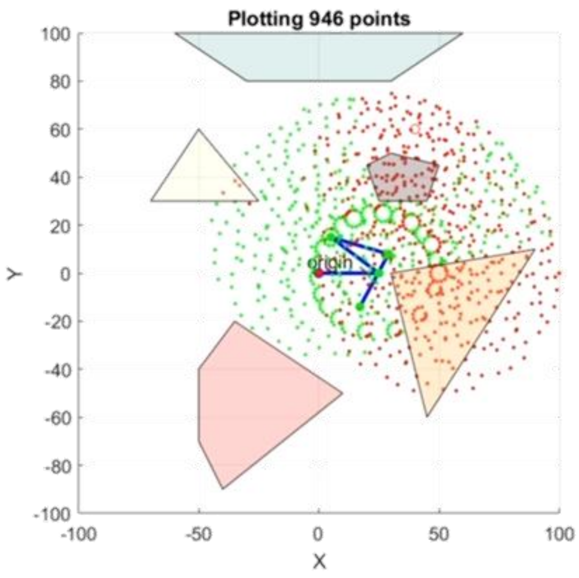

Figure 2: Constructing Configuration Space

Configuration space is constructed by iterating through the angles of each joint, as shown in the video work_space_pointcloud.mp4. The process involves setting a 0.5 rad step for each joint and sweeping through a [0, 2pi] range to collect collision data for each joint state. States where the robot collides with obstacles are marked as '1' in the collisionMap variable, indicating unusable points for path generation. Figure 2 illustrates this, with green points indicating collision-free states and red points indicating collisions. Every point in the figure corresponds to a specific joint state which is four-dimensional and cannot be plotted here.

1.3 Apply algorithms

Next, algorithms are applied to generate paths in the configuration space. The functions of each variable are described below:

1.3.1 PRM

[PRMPath, V, G] = PRM(N, K, linkLen, p_init, p_goal, pointSpace, jointState, B)

This function first uses pointSpace, jointState, and FK() to identify the joint states closest to p_init and p_goal, designated as initialState and goalState, respectively. It then applies the PRM algorithm to find a path between initialState and goalState in the configuration space.

1.3.2 RRT

[RRTPath, V, E] = RRT(NumNodes, delta_q, tolerance, bias, linkLen, p_init, p_goal, pointSpace, jointState, B)

This function operates similarly to PRM, utilizing the RRT algorithm.

1.4 Results

Results are available in the 'Demos' folder as videos. Attached are some final trajectory results.

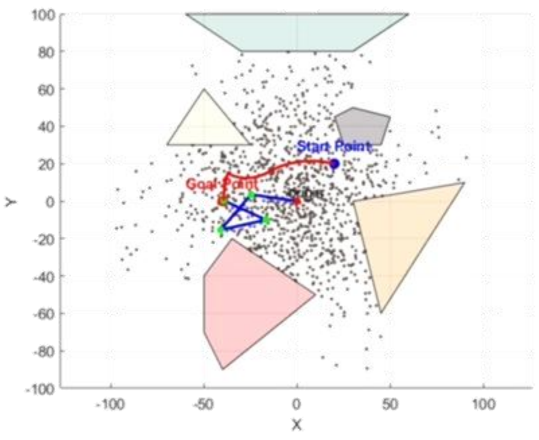

Figure 3: PRM Trajectory

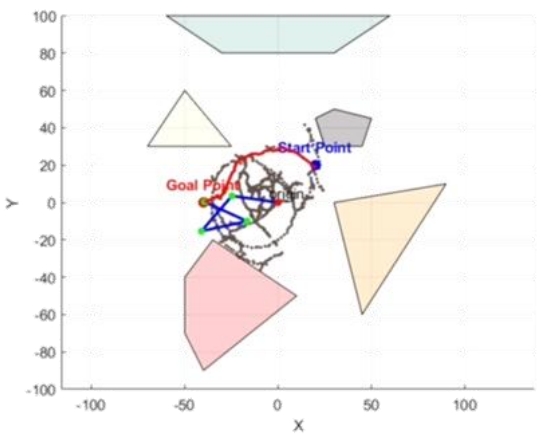

Figure 4: RRT Trajectory

Task 2: Planar Rigid Robot

2.1 Potential fields

The artificial potential field method requires both an attractive and a repulsive field to be applied on a manipulator. The attractive field guides the manipulator to the goal while the repulsive field is responsible to avoid contact with obstacles. The formulas are defined as:

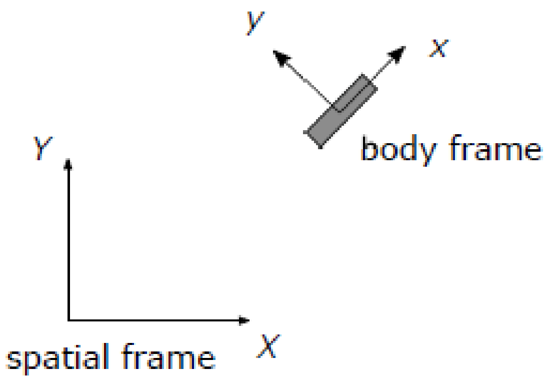

Figure 5: Manipulator Representation

$$U_{att,j}(q) = \begin{cases} \frac{1}{2}\zeta_j \rho^2\left(r_j(r), r_j(q_G)\right) & \rho\left(r_j(q), r_j(q_G)\right) \leq \rho_G^* \\ \rho_G^* \zeta_j \rho\left(r_j(q), r_j(q_G)\right) - \frac{1}{2}\zeta_j (\rho_G^*)^2 & otherwise \end{cases}$$

$$U_{rep,i,j}(q) \doteq \begin{cases} \frac{1}{2}\eta \left(\frac{1}{\rho_i\left(r_j(q)\right)} - \frac{1}{Q^*}\right)^2 & \rho_i\left(r_j(q)\right) \leq Q_i^* \\ 0 & otherwise \end{cases}$$

In our case, this method was applied to a rectangular shaped manipulator. Therefore, we can not just apply the fields to the center of the rectangle since this could cause the edges to still intersect with the obstacles. Therefore, we use control points on the corners of the rectangle. By summing the fields of these control points we steer the edges of the manipulator away from the obstacles. Since the control points are defined in a body frame, we need to adjust every change in position by using the jacobian. The final formula is given below:

$$u(q) = \sum_j u_{att,j}(q) + \sum_i \sum_j u_{rep,i,j}(q) = \sum_j J_j^T(q) f_{att,j}(q) + \sum_i \sum_j J_j^T(q) f_{rep,i,j}(q)$$

2.2 Model parameters

Our workspace was chosen to be 10 by 10 units while our manipulator was a 1 by 0.1 rectangle free to rotate around its center. Again, the corners of the rectangle were defined in the body frame, not in the spatial frame, as shown below. Only in the algorithm would they get transformed to the spatial frame. Obstacles were represented as a cell containing vertices and faces. Faces are arrays holding the indices of the vertices that make it up. The dimensions of the obstacles can be seen in the result section. For simplicity, all the obstacles were chosen to be rectangles.

The potential field parameters were carefully chosen in an iterative way. for U_att, zeta was chosen to be 0.05 and U_rep has an eta of 0.1. The distance at which the obstacles start repulsing the manipulator was 1.5 and the attraction threshold distance was 10.

2.3 Algorithms

2.3.1 RB_potential_field

[q, cp] = RB_potential_field(qi, qg, O, Cpoints)

Takes the goal and initial position of the manipulator center together with the obstacles and the control points. It returns an array with the consecutive states and a cell type structure howling the consecutive positions of all the control points in each step. It uses a function to determine the attractive field and repulsive field at each location as well as an algorithm to determine the closest point from an obstacle to a control point.

2.3.2 closestPointToEdge

[closest, distance] = closestEdgePointToPoint(edge, points)

Takes an edge and a control point as input and returns the closest point on the edge to the point as well as the distance. This function is used in the repulsive field calculation to determine its strength.

2.4 Results

The results show the manipulator is able to successfully reach the target in the desired orientation while at the same time maneuvering between relatively tight spaces.

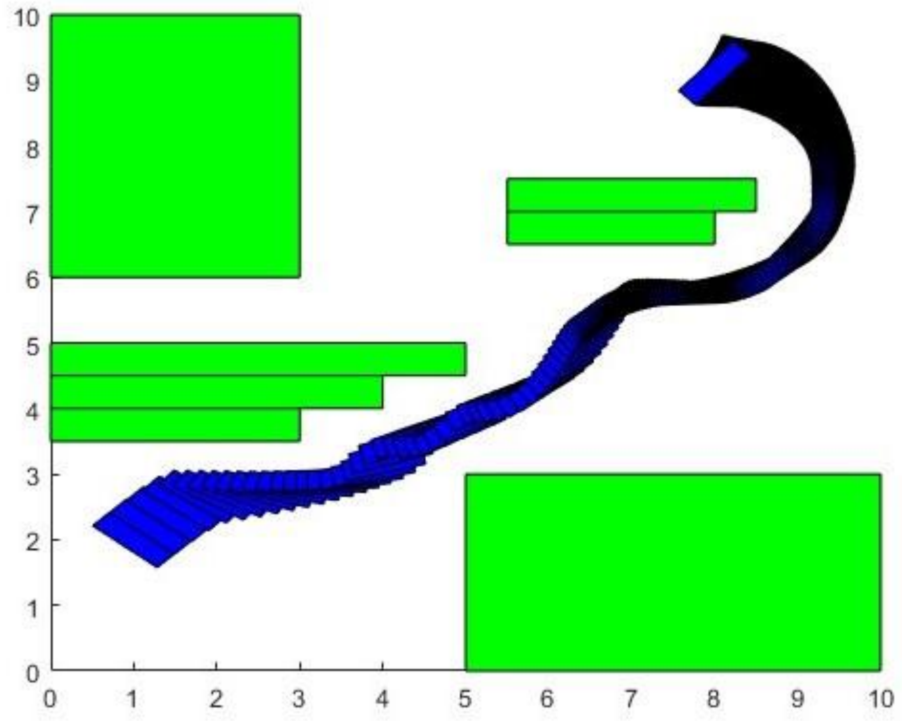

Figure 6: Artificial Potential Field Path

Task 3: Simple Model for Flexible Needle

3.1 Kinematics Equations

In Task 3, the path of a flexible needle is modeled with the trajectory of a unicycle robot. With a unicycle model, the state space $q = [x \quad y \quad \theta]^T$ is a function of $u = [u_\sigma \quad u_\omega]^T$ where u_σ and u_ω are the wheel rotation rate and $\dot{\theta}$ accordingly. As the velocity of the unicycle, $u_v = ru_\sigma$, is proportional to u_σ and there is direct control over the velocity of the needle, it can be assumed that $r = 1$ and $u = [u_v \quad u_\omega]^T$.
The kinematic equations are as follows:

$$\dot{x} = u_v \cos(\theta)$$
$$\dot{y} = u_v \sin(\theta)$$
$$\dot{\theta} = u_\omega$$

Which can also be represented as:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_v \\ u_\omega \end{bmatrix}$$

When calculating the state as a function of time:

$$q_t = q_0 + \int_0^t \dot{q}\,dt$$

As $\dot{x}_t$ and $\dot{y}_t$ are both functions of θ_t:

$$\theta_t = \theta_0 + \int_0^t u_\omega dt = \theta_0 + u_\omega t$$

$$x_t = x_0 + \int_0^t u_v \cos(\theta_t)\,dt = x_0 + \begin{cases} \frac{u_v}{u_\omega}(\sin(\theta_t) - \sin(\theta_0)) & u_\omega \neq 0 \\ u_v t \cos(\theta_0) & u_\omega = 0 \end{cases}$$

$$y_t = y_0 + \int_0^t u_v \sin(\theta_t)\,dt = y_0 + \begin{cases} \frac{u_v}{u_\omega}(\cos(\theta_t) - \cos(\theta_0)) & u_\omega \neq 0 \\ u_v t \sin(\theta_0) & u_\omega = 0 \end{cases}$$

3.2 Lie-Group-Theoretic Kinematic Equations

Given the transformation g, the body Jacobian J^b can be calculated using the following:

$$g = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix}$$

$$g^{-1} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & -x\cos(\theta) - y\sin(\theta) \\ -\sin(\theta) & \cos(\theta) & x\sin(\theta) - y\cos(\theta) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\frac{dg}{dx} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\frac{dg}{dy} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\frac{dg}{d\theta} = \begin{bmatrix} -\sin(\theta) & -\cos(\theta) & 0 \\ \cos(\theta) & -\sin(\theta) & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$J^b = \begin{bmatrix} \widetilde{g^{-1}\frac{dg}{dx}} & \widetilde{g^{-1}\frac{dg}{dy}} & \widetilde{g^{-1}\frac{dg}{d\theta}} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The twist ζ and transform derivative $\dot{g}$ can then be calculated using J^b and $\dot{q}$

$$\widetilde{g^{-1}\dot{g}} = \vec{\zeta} = J^b\dot{q} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_v \\ u_\omega \end{bmatrix} = \begin{bmatrix} u_v \\ 0 \\ u_\omega \end{bmatrix}$$

$$\dot{g} = g\hat{\xi} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -u_\omega & u_v \\ u_\omega & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -u_\omega \sin(\theta) & -u_\omega \cos(\theta) & u_v \cos(\theta) \\ u_\omega \cos(\theta) & -u_\omega \sin(\theta) & u_v \sin(\theta) \\ 0 & 0 & 0 \end{bmatrix}$$

g_t can also be calculated as a function of g_0, $\hat{\xi}$, and t such that:

$$g_t = g_0 e^{\hat{\xi}t} = \begin{bmatrix} \cos(\theta_0) & -\sin(\theta_0) & x_0 \\ \sin(\theta_0) & \cos(\theta_0) & y_0 \\ 0 & 0 & 1 \end{bmatrix} e^{\begin{bmatrix} 0 & -u_\omega & u_v \\ u_\omega & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} t}$$

q_t can then be derived from g_t:

$$q_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} g_{t_{1,3}} \\ g_{t_{2,3}} \\ atan2\left(g_{t_{2,1}}, g_{t_{1,1}}\right) \end{bmatrix}$$

As it can be applied without a conditional if-else statement, this method was used to calculate the nonholonomic path of the needle.

3.3 Construct Workspace

The workspace is set to be within the bounds of x_{min}, x_{max}, y_{min}, and y_{max}. For the purpose of testing q_I, q_G, and the obstacles are randomly determined with $\theta = [-\pi, \pi]$.
The shape of the needle tip is a symmetrical isosceles triangle defined by the width and length dimensions $[w \quad l]^T$. The velocity and curvature of the needle are also defined as $u = [u_v \quad u_\omega]^T$.
For the RRT algorithm, a maximum step size and maximum number of branches is also defined.
Given these parameters, the following algorithms can be used to solve the nonholonimic pathfinding problem.

3.4 Algorithms

3.4.1 Dubin's Curve

The first problem to be solved is calculating the shortest path between two points using only Dubin's curves. A Dubin's curve can be classified into the falls categories of RSR, LSL, RSL, LSR, RLR, and LRL which are demonstrated in Figure 7. Each solution can be calculated using the methods found in [1]. When calculating the Dubin's curve, each of the viable solutions is calculated and the shortest path is selected.

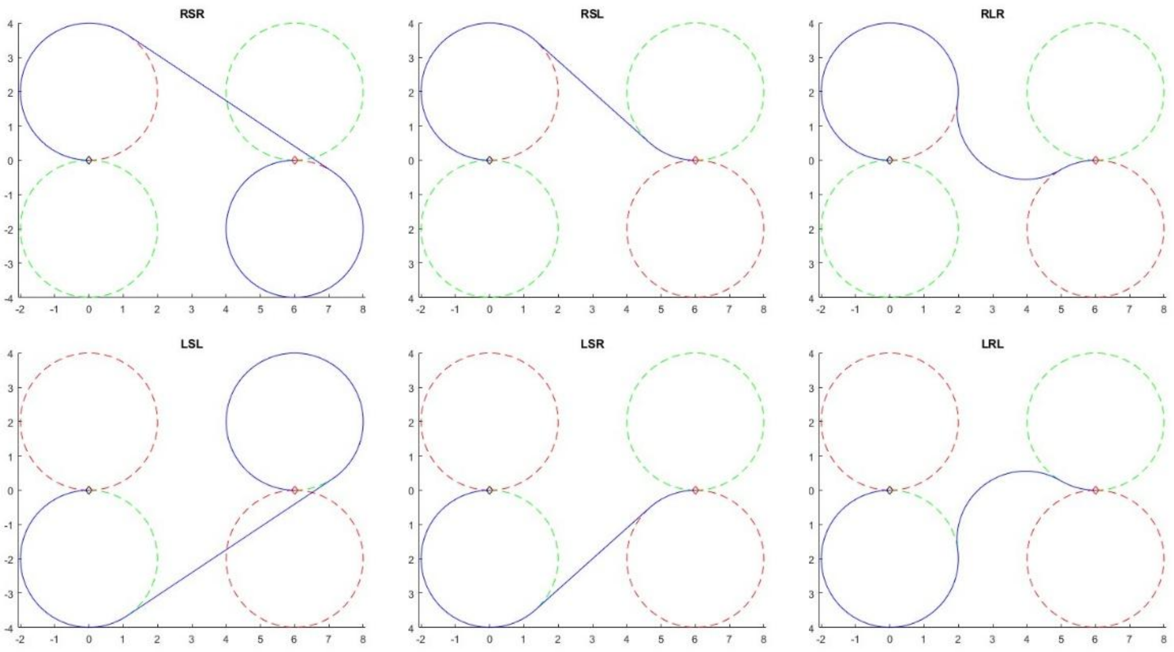

Figure 7: Dubin's Curves

3.4.2 Continuous Collision Detection Algorithm (CCD)

With a Dubin's path, the next step is to check for collisions with obstacles. As the two possible movements were traveling in a straight line and moving along a circular path. It was best to consider a different algorithm for either case. The CCD Algorithm in [2] was chosen to perform collision checks on the curved path. The algorithm creates a ring segment of the area covered by the object following Dubin's Curve, as shown in Figure 8. r_{in} and r_{out} are determined by the geometry of the object and the arc radius. θ_1 and θ_2 are determined by the object geometry, arc radius, q_I, q_G, and direction of travel. The CCD Algorithm only works when $|\theta_2 - \theta_1| \leq \pi$. Obstacles are then checked for intersections with the arc shape.

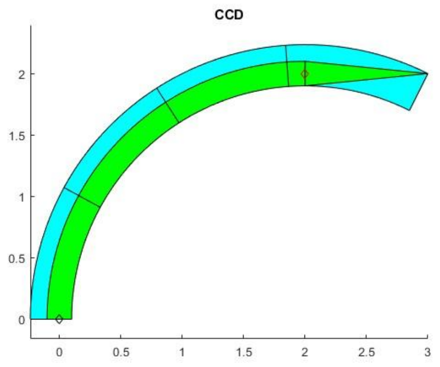

Figure 8: CCD Path Arc

3.4.3 Gilbert-Johnson-Keerthi Distance Algorithm (GJK)

When moving in a straight line the needle tip is maintaining a constant orientation, thus the shape if the needle tip path can be constructed by connecting the extreme vertices of the initial and final configurations to form a convex polygon, as shown in Figure 9. To check for collisions, the vertices of the initial and final positions can be concatenated, and the convex hull method can be applied to create the needle tip path polygon. The GJK Algorithm can then be applied to check for collisions with any obstacles.

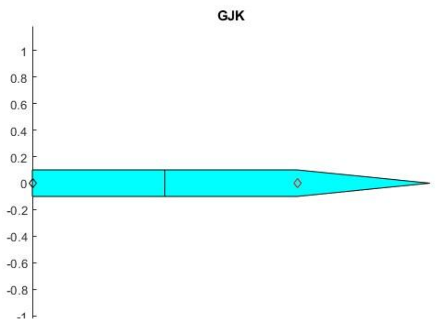

Figure 9: Straight Path Outline

3.4.4 Dubins Path RRT

The RRT Algorthm works as follows:

1. Find $q_{closest}$ in V with shortest Dubins path to q_G.
2. Split Dubins path into segments with $len \leq step\_size$.
3. For each segment, check for collision using CCD and GJK appropriately.
4. If no collision, add $q_{segment}$ to V and record q_{prev} and $u_{prev} = \begin{bmatrix} [-1,0,1] \\ dt \end{bmatrix}$.
5. If no collision, proceed to next segment, else stop exploring path.
6. If Dubins path was completed, q_G was reached, return path.
7. If V is unchanged between steps 1 and 5 repeat 1-5 with q_{rand}.
8. Repeat 1-7 N times, or until q_G is reached.

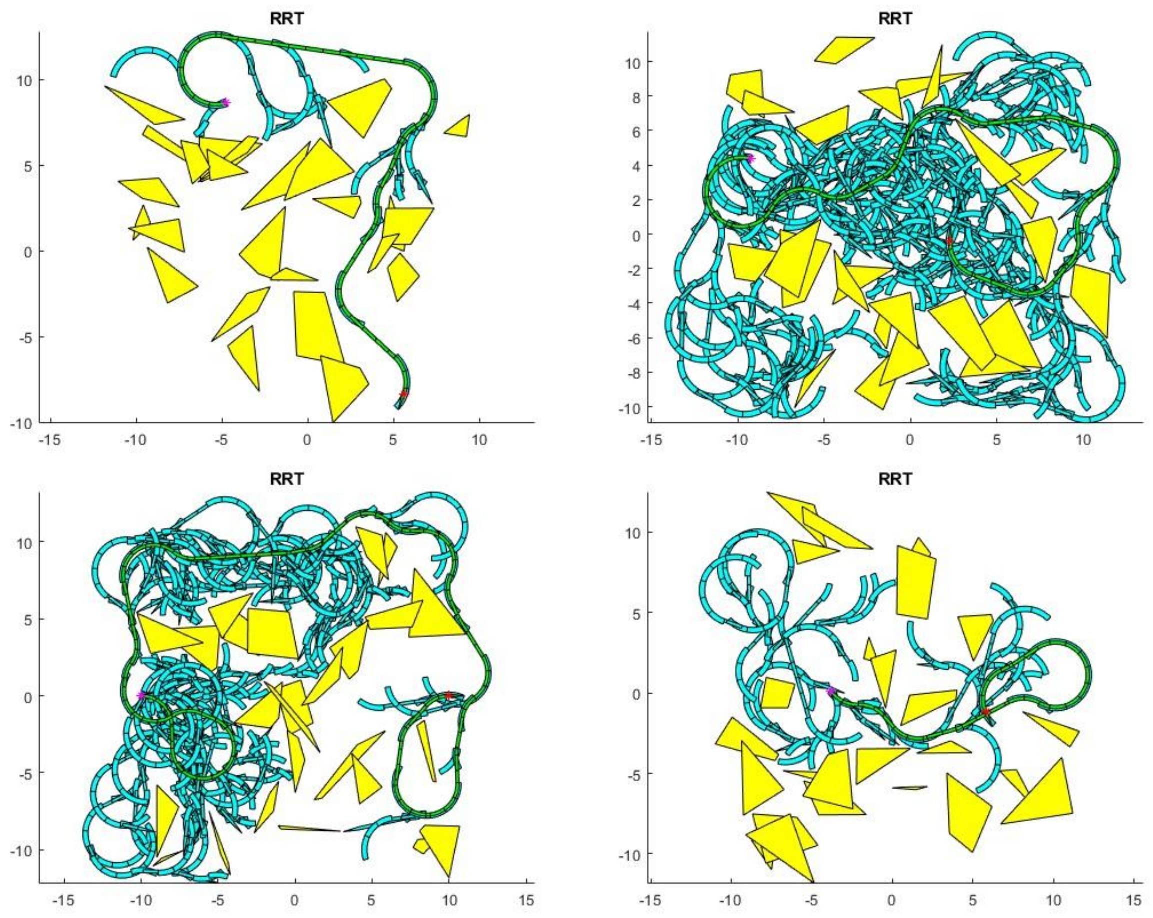

Figure 10: Nonholonomic RRT Examples

Task 4: 3D Cylindircal Cell Decomposition

4.1 Approach

Our project of choice was creating a 3D Cylindrical Cell Decomposition. We approached this problem by subdividing it in 3 smaller tasks to keep things organized:

1. Slice obstacle polyhedrons into polygons at specified planes
2. If given 2 points in 3D space, determine whether or not the line connecting them intersects the polyhedron
3. Combining the previous functions, connect points in different slices that do not cause intersection with the polyhedron.

4.2 Obstacle slicing

Every polygon, which is defined as a shape comprised entirely of straight-line sides, can essentially be broken down into a set of triangles. Initially, each face of an obstacle is decomposed into multiple triangles as Figure 12:

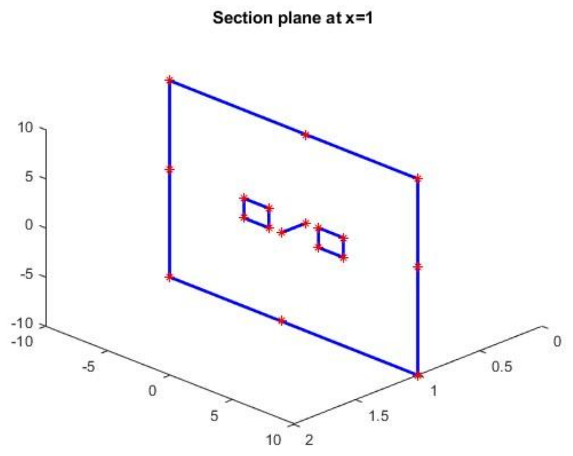

Figure 12: Section Planc Slice

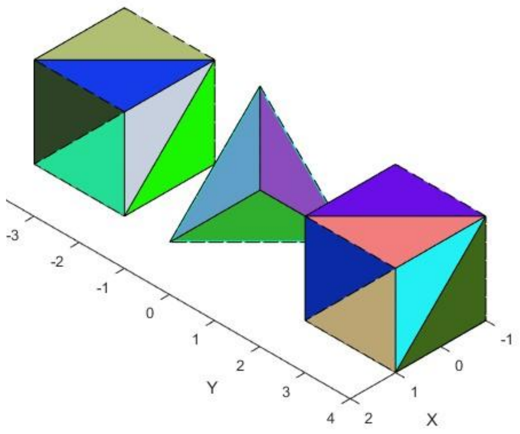

Figure 12: Triangular Decomposition

Points intersecting with the plane on each triangle can be seamlessly connected. This allows for the construction of a sectional view even when the plane intersects with a non-convex polyhedron.

4.3 Interference detection

This section of the code used 2 geometrical principles to determine whether or not an intersection took place and what the intersecting point(s) would be.
First, given 2 points connected by a line, determine if it intersects with a given plane. Second, given the intersection point, determine if it falls within the face. In order to simplify this last step, we triangulated all the faces. This allows us to simplify the math by using the following formula reported by [3]:

$$\vec{u} = P_2 - P_1, \vec{v} = P_3 - P_1, \vec{n} = \vec{u} \times \vec{v}, \vec{w} = P - P_1$$

and,

$$\gamma = [(\vec{u} \times \vec{w}) \cdot \vec{n}]/\vec{n}^2$$
$$\beta = [(\vec{w} \times \vec{v}) \cdot \vec{n}]/\vec{n}^2$$
$$\alpha = 1 - \gamma - \beta$$

The point lies inside the triangle if $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$, and $0 \leq \gamma \leq 1$.
This proved to be a robust method of determining an intersection and was tested for many cases never resulting in a false detection or a missed detection.

4.4 Graph connection

A vertical plane is swept along the X-axis with each unique x coordinate of the object and boundary vertices being recorded as an event. The first and last events are discarded as they are the edge cases of the boundary. At each event the obstacle slicing algorithm is used to create a 2D slice of the object and boundary intersections. This slice is then fed into a 2D Cylindrical Cell Decomposition Algorithm which returns the nodes and edges at that event. Using the Interference Detection Algorithm, edges are drawn between each node of the event and all visible nodes of the previous event. The Interference Detection Algorithm is also used to connect qI and qG to the closest visible node of the events they fall between. Because the 2D CCD algorithm connects cell edge nodes directly and does not compute cell center nodes, there is an edge case of a square hole/tunnel where no node is able to be placed.

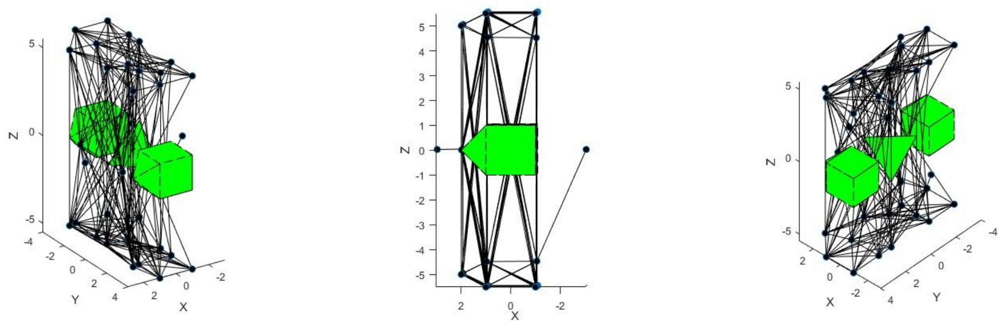

Figure 13: 3D CCD Example

Team Workload distribution

The workload for this final project wa divided fair and evenly with all members contributing an even amount to this final report. Stefan was responsible for implementing Task 3 and combining Xiyuan's and Nathan's functions into the 3D CCD function. Xiyuan was responsible for Task 1 and writing the algorithm for obstacle slicing. Nathan was responsible for the implementation of Task 2 and writing the algorithm for the interference detection in the 3D vertical cell decomposition.

References

[1] A. G, "A Comprehensive Step-by-Step Tutorial to Computing Dubin's Path," Andy G's Blog, 21 October 2012. [Online]. Available: https://gieseanw.wordpress.com/2012/10/21/a-comprehensive-step-by-step-tutorial-to-computing-dubins-paths/. [Accessed May 2024].

[2] G. Schildback, "A Continuous Collision Detection Algorithm for Dubons Paths," *IEEE Intelligent Vehicles Symposium (IV),* 2023.

[3] W. Heidrich, "Computing the Barycentric Coordinates of a Projected Point," *Journal of Graphics GPU and Game Tools ,* vol. 10, no. 3, pp. 9-12, 2005.