ECE 5330 Final Project
Full Dubin's Path Manager

Joshua Brock

I. Project Description

THIS project is to implement a full Dubin's path manager. The path manager previously implemented in class is incomplete. The previous implementation included the four arc-straight-arc configurations, which always includes a straight-line path between two arcs. A straight-line path is not always possible, nor is it always the most efficient solution. If the distance between two waypoints is less than $2R$, where R is the minimum radius the aircraft can fly, the aircraft must take an arc-arc-arc path. Additionally, there are cases where an arc-arc-arc solution is more efficient than an arc-straight-arc when both types of paths are possible. For this project we define four arc-straight-arc paths and two arc-arc-arc paths:

- right-straight-right (RSR)
- right-straight-left (RSL)
- left-straight-right (LSR)
- left-straight-left (LSL)
- right-left-right (RLR)
- left-right-left (LRL)

The first four paths were already implemented, so this project focuses on implementing the RLR and LRL paths. Implementing the remaining paths in the path manager involves determining expressions for the RLR and LRL distances and switching planes, modifying the `DubinsParameters` and `DubinsParamsStruct` classes to accept RLR and LRL path parameters, and adding functions to calculate the RLR and LRL distances and switching planes. The following sections detail how the RLR and LRL paths are derived and implemented.

II. Project Details

A. Math Derivations

To implement the RLR and LRL paths, we must define the following symbols:

$p_s :=$ the starting point
$p_e :=$ the ending point
$c_{ls} :=$ the center of the left starting circle
$c_{rs} :=$ the center of the right starting circle
$c_{le} :=$ the center of the left ending circle
$c_{re} :=$ the center of the right ending circle
$c_{li} :=$ the center of the left intermediary circle
$c_{ri} :=$ the center of the right intermediary circle
$\vartheta :=$ the angle between the vector from c_{*s} to c_{*e} and north
$\angle c_{*i} :=$ the angle between c_{*e} and c_{*s} at c_{*i}
$\theta_{rlr} := \vartheta - \angle c_{li} :=$ the angle between the vector from c_{rs} to c_{li} and north
$\theta_{lrl} := \vartheta + \angle c_{ri} :=$ the angle between the vector from c_{ls} to c_{ri} and north
$\lambda := ||p_s - p_e||$
$l := ||c_{*s} - c_{*e}||$
$R :=$ the minimum turning radius (at a given airspeed)
$\mathcal{H}(z, q) :=$ the half space defined by the point z, which lies on the half plane, and the unit vector q.
$z_1 :=$ the switching point from the starting circle to the intermediate circle
$z_2 :=$ the switching point from the intermediate circle to the ending circle
$z_3 := p_e$
$L_{RLR} :=$ the RLR path length
$L_{LRL} :=$ the LRL path length

To derive each of the above symbols, we will need look at the geometry of RLR and LRL paths shown in Figures (1) and (2) below. From the geometry, we are able to find the optimal switching points z_1 and z_2. We know that q_1 and q_2 will be tangential to both circles, so c_{*i} will be equidistant from c_{*s} and c_{*e}. Using this information and the geometry, we are able to find z_1, z_2, q_1, q_2, c_{*i}, and the path length L_{***}.

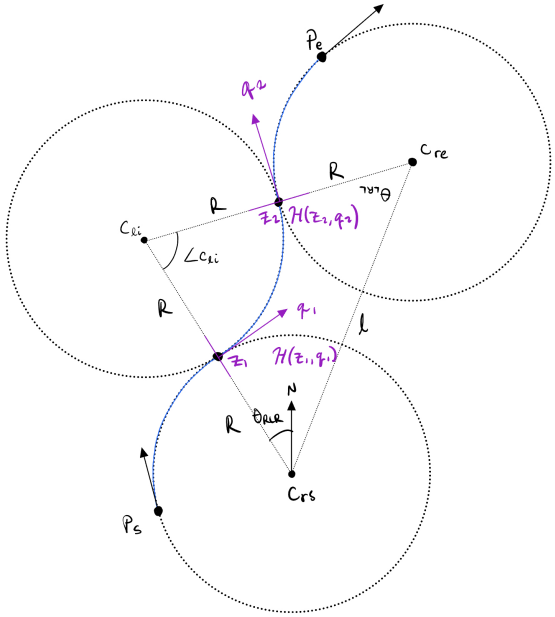

Fig. 1. Geometry of right-left-right Dubin's path.

To find z_1 and z_2 for the RLR and LRL cases, we first need to derive c_{*i}. To find c_{*i}, we also need $\angle c_{*i}$ and ϑ. To derive $\angle c_{*i}$, we can use the fact that c_{*s}, c_{*e}, and c_{*i} form a triangle with a base of length l and two sides of length $2R$. We then

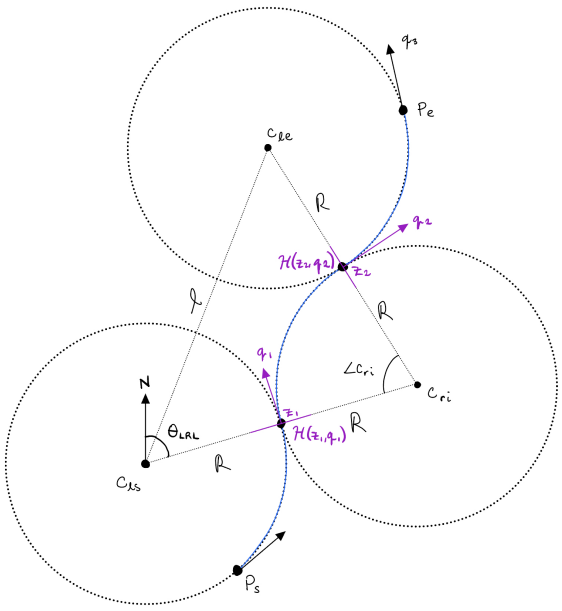

Fig. 2. Geometry of left-right-left Dubin's path.

use the law of cosines to solve for $\angle c_{*i}$:

$$\begin{aligned} l &= ||c_{*s} - c_{*e}|| \\ \cos\angle c_{*i} &= \frac{(2R)^2 + (2R)^2 - l^2}{2(2R)(2R)} \\ &= \frac{8R^2 - l^2}{8R^2} \\ &= 1 - \frac{l^2}{8R^2} \\ &= \frac{l}{4R} \\ \angle c_{*i} &= \cos^{-1}\left(\frac{l}{4R}\right) \end{aligned}$$

Since ϑ is the angle from the starting arc center to the ending arc center with respect to north, we know that

$$\vartheta = \tan^{-1}\left(\frac{c_{*e_{east}} - c_{*s_{east}}}{c_{*e_{north}} - c_{*s_{north}}}\right)$$

Now we need to find θ_{rlr} and θ_{lrl}. The intermediate circle in an RLR configuration will always be to the left of the vector pointing from c_{rs} to c_{re}. Likewise, the intermediate circle in an LRL configuration will always be to the right of the vector pointing from c_{rs} to c_{re}. Therefore, we can find θ_{rlr} and θ_{lrl} by

$$\begin{aligned} \theta_{rlr} &= \vartheta - \angle c_{li} \\ \theta_{lrl} &= \vartheta + \angle c_{ri} \end{aligned}$$

We now have all the information we need to find c_{*i}:

$$\begin{aligned} c_{li} &= \begin{bmatrix} c_{rs_{north}} + 2R\cos\theta_{rlr} \\ c_{rs_{east}} + 2R\sin\theta_{rlr} \\ c_{rsdown} \end{bmatrix} \\ c_{ri} &= \begin{bmatrix} c_{ls_{north}} - 2R\cos\theta_{lrl} \\ c_{ls_{east}} - 2R\sin\theta_{lrl} \\ c_{lsdown} \end{bmatrix} \end{aligned}$$

Now that we have the intermediate circle centers, we can use the fact that the intermediate circle center must always be tangent to both start and end circles to find z_1 and z_2. This means that z_1 lies on the vector between c_{*i} and c_{*s}, and z_2 lies on the vector between c_{*i} and c_{*e}. Thus

$$\begin{aligned} z_1 &= c_{*i} + \frac{c_{*s} - c_{*i}}{||c_{*s} - c_{*i}||}R \\ z_2 &= c_{*i} + \frac{c_{*s} - c_{*i}}{||c_{*s} - c_{*i}||}R \end{aligned}$$

We can now derive q_1 and q_2 for both RLR and LRL. To find q_1 for RLR, we need to define the plane $\mathcal{H}_1$ by finding two vectors on the plane. Since q_1 is tangential to the start and intermediate circles and normal to $\mathcal{H}_1$, we know that the unit vector pointing from c_{*i} to c_{*s} must lie on $\mathcal{H}_1$. Since we don't change altitudes, we know that the up vector $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ also lies on $\mathcal{H}_1$. We can then find q_1 by taking a cross product and we know it's pointing the right way by the right-hand rule. Similarly, we can find q_2 by defining the plane $\mathcal{H}_2$ with the unit vector pointing from c_{*i} to c_{*e} and the up vector.

$$\begin{aligned} q_{1_{RLR}} &= \left(\frac{c_{rs} - c_{li}}{||c_{rs} - c_{li}||}\right) \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ q_{2_{RLR}} &= \left(\frac{c_{re} - c_{li}}{||c_{re} - c_{li}||}\right) \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned}$$

To find q_1 and q_2 for LRL, we can do the same thing, except we need to negate the cross product result. This is because the vector pointing from c_{ri} to c_{ls} is pointing the opposite direction compared to the RLR case. So

$$\begin{aligned} q_{1_{LRL}} &= -\left(\frac{c_{ls} - c_{ri}}{||c_{ls} - c_{ri}||}\right) \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ q_{2_{LRL}} &= -\left(\frac{c_{le} - c_{ri}}{||c_{le} - c_{ri}||}\right) \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned}$$

Now, we can derive the RLR and LRL path lengths. We need to find the three arc lengths. We just need to find the angle between the switching points. We can do that by using the dot product formula on the vectors from the circle centers to the switching points. The angle between two vectors is given by:

$$\theta = \cos^{-1}\left(\frac{a^T b}{|a||b|}\right)$$

Now we can use the arc length formula ($l = R\theta$) to find the total length of the path. Define $\hat{i}_{ab}$ as the unit vector from a to b and the total path length is:

$$\begin{aligned} L_{***} = & R\cos^{-1}\left(\hat{i}_{c_s p_s}^T \hat{i}_{c_s z_1}\right) \\ & + R\cos^{-1}\left(\hat{i}_{c_i z_1}^T \hat{i}_{c_i z_2}\right) \\ & + R\cos^{-1}\left(\hat{i}_{c_e z_2}^T \hat{i}_{c_e p_e}\right) \end{aligned}$$

B. Algorithms

We need to modify Algorithm 9 to work with the RLR and LRL paths. In the original Algorithm 9, we compute four lengths, each corresponding to a Dubin's path. We need to modify Algorithm 9 to also compute lengths L_5 and L_6, which correspond to L_{RLR} and L_{LRL}, respectively. We also need to consider that for a RLR or LRL path to be possible, the distance between two waypoints λ must be less than or equal to $4R$, since the intermediate circle must touch both the starting and ending circles. We can therefore modify Algotithm 9 to read (starting at line 5):

5: Compute L_1, L_2, L_3, and L_4 using equations (11.9) through (11.12)
6: **if** $||p_s - p_e|| <= 4R$ **then**
7: Compute L_5, L_6 using equation derived above
8: **else**
9: $L_5, L_6 = \infty$
10: **endif**
11: $L \leftarrow \min\{L_1, L_2, L_3, L_4, L_5, L_6\}$

After those modifications, we continue on with the rest of the algorithm as before with the added cases to handle defining the paths for RLR and LRL. We next modify Algorithm 10 to accept RLR and LRL paths. For this, we need to create two extra states for the intermediate circle. Starting at line 5, Algorithm 10 is now:

5: **if** state = 1 **then**
6: flag $\leftarrow 2$, $c \leftarrow c_s$, $\rho \leftarrow R$, $\lambda \leftarrow \lambda_s$
7: **if** $p \in \mathcal{H}(z_1, -q_1)$ **then**
8: state $\leftarrow 2$
9: **end if**
10: **else if** state = 2 **then**
11: **if** $p \in \mathcal{H}(z_1, q_1)$ **then**
12: **if** path = RLR or LRL **then**
13: state $\leftarrow 4$
14: **else**
15: state $\leftarrow 3$
16: **end if**
17: **end if**
18: **else if** state = 3 **then**
19: flag $\leftarrow 1$, $r \leftarrow z_1$, $q \leftarrow q_1$
20: **if** $p \in \mathcal{H}(z_2, q_1)$ **then**
21: state $\leftarrow 6$
22: **end if**
23: **else if** state = 4 **then**
24: flat $\leftarrow 2$, $c \leftarrow c_i$, $\rho \leftarrow R$
25: **if** $p \in \mathcal{H}(z_2, -q_2)$ **then**
26: state $\leftarrow 5$
27: **end if**
28: **else if** state = 5 **then**
29: **if** $p \in \mathcal{H}(z_2, q_2)$ **then**
30: state $\leftarrow 6$
31: **end if**
32: **else if** state = 6 **then**
33: flag $\leftarrow 2$, $c \leftarrow c_e$, $\rho \leftarrow R$, $\lambda \leftarrow \lambda_e$
34: **if** $p \in \mathcal{H}(z_3, -q_3)$ **then**
35: state $\leftarrow 7$
36: **end if**
37: **else if** state = 7 **then**
38: **if** $p \in \mathcal{H}(z_3, q_3)$ **then**
39: state $\leftarrow 1$
40: $i \leftarrow i + 1$ until $i = N$
41: findDubinsParameters($w_{i-1}, \chi_{i-1}, w_i, \chi_i, R$)
42: **end if**
43: **end if**

III. Code Structure

The code is located on GitLab at this link: https://gitlab.com/utahstate/courses/5330_suas/student-groups/josh-brock/full_dubins_path

Outside of implementing the RLR and LRL path calculations and modifying Algorithms 9 and 10, there were two more things that needed to be done:

1) Adding member variables to the Dubin's parameter classes in `dubins_parameters.py`
2) Modifying the path display functions in `draw_waypoints.py`

For the `DubinsParameters` class to handle an intermediate circle, I needed to add members `self.center_i`, `self.dir_i`, and `self.n2`. For `DubinsParamsStruct`, I added `self.c_i`, `self.lam_i`, and `self.q2`.

To display different colors for each part of the path, I modified the `dubins_points` function to return colors associated with each of the generated points. I then use those colors in the call to `waypoint_plot_object.setData(pos=points, color=colors)`. If the path is not a Dubin's path, the colors display as normal. I assigned the following colors to the path components:

- green: the starting arc
- blue: the intermediate arc
- pink: the intermediate straight line
- orange: the ending arc

A basic diagram of the code architecture is provided in Figure (3). Note that the diagram only includes material involving the RLR and LRL paths. The other paths are still be in the structure where they previously were.

IV. Results

I now have a fully functional Dubin's path manager. Waypoints can be arbitrarily close to or far from each other without any issues. Figure (4) shows a sample path showing both the RLR and LRL path types. Figure (5) shows the states while the aircraft flies that path.

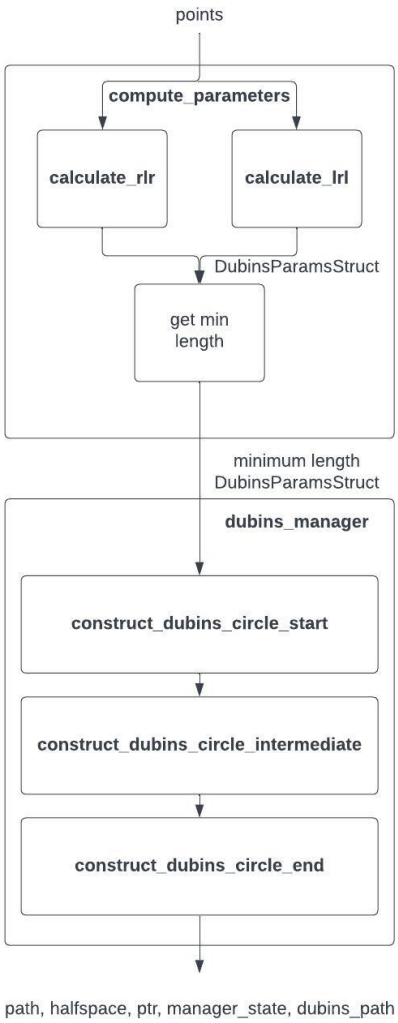

Fig. 3. Overview of modified code architecture.

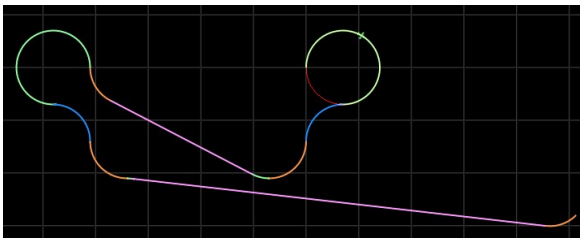

Fig. 4. Sample path showing RLR (middle) and LRL (left), as well as some arc-straight-arc paths. Note the different colors for each component of the path.

V. Difficulties

I ran into a few issues while working on the project. One difficulty I had was determining q_1 and q_2 for both the RLR and LRL paths. I initially had the RLR and LRL half-space vectors swapped. That was mostly due to my lacking attention to detail and I propagated that mistake all the way through to my code. This took a while to debug, but I ended up going back to the math to fix it. I fixed this issue by physically doing the cross product with my right hand and realizing I had the signs flipped.

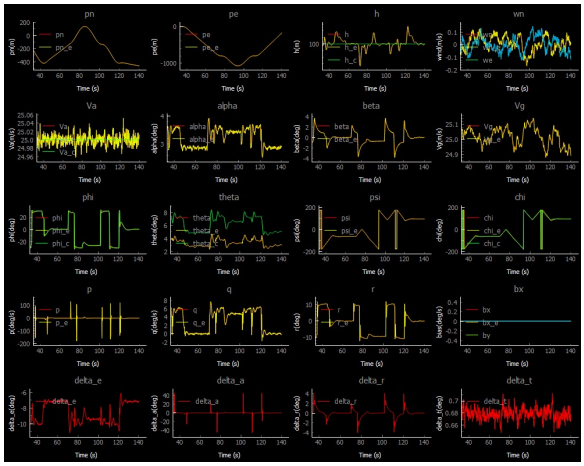

Fig. 5. Plots showing the states of the aircraft as it flies through the path shown in Figure (3).

Another difficulty I had was dealing with invalid values in the `arccos` and `sqrt` functions in the arc-straight-arc paths. I had commented out the section of code where it throws an error if the distance between two waypoints was greater than $2R$, since the RLR and LRL implementations allow for shorter distance between waypoints. Distances between arc centers less than $2R$ ended up being problematic for the arc-straight-arc paths since the values used by `arccos` are larger than 1, and the values used in `sqrt` are negative. I fixed this by adding an `if` statement checking that the distance between start and end arc centers is greater than $2R$. If it's greater, the parameters and length are computed normally. If the distance is less than $2R$, a length of ∞ is returned for the path, and the algorithm then has to choose an RLR or LRL path. Technically, the arc-straight-arc paths should be able to handle shorter distances, but with the way we've derived the equations and implemented them, it is not possible.

VI. Conclusion

In conclusion, I derived and implemented a full Dubin's path manager, including every arc-straight-arc and arc-arc-arc path configuration. This path manager can handle any combination of waypoints, regardless of distance between them.