Ride Sharing Optimal Planner and Trajectory Tracking

Erica Tevere, Zidi Tao

Problem Introduction: Our project looks at finding a solution to a dynamic ride sharing problem. We define this problem with the following conditions:

1. A random start position is known
2. There are four randomly located waypoints that each need to be visited only once and in no specified order
3. There is one final goal that does not change which is centralized given the positional bounds of the problem
4. All dynamic constraints are known (including location of obstacles)
5. The goal is to minimize the control effort (the smoothness of the ride) and keep the controlled vehicle as close to the ideal path along the route

To be effective in our ability to solve this problem we break it into three separate subproblems:

1. Solving for the optimal route for the shuttle to take given know environmental constraints
2. Solving for the optimal trajectory of the entire known route given a dynamics model
3. Designing a trajectory tracking controller using feedback linearization

We will now go into further detail of how each of these subproblems is addressed and solved.

Subproblem 1

To solve for subproblem 1, we break the overall task of finding the optimal route given known constraints into a set of smaller tasks.

Graph Search: First, we set up the constraints and locate our desired points on our constrained graph. We use an implementation of Dijkstra's algorithm to generate a known cost and optimal path to traverse between each desired point.

Route Planner: We then use these known matrices storing cost and optimal path between each set of points to solve a version of the Traveling Salesman Problem using dynamic programming. The dynamic programming algorithm outputs the optimal route to visit each waypoint from the start to the goal.

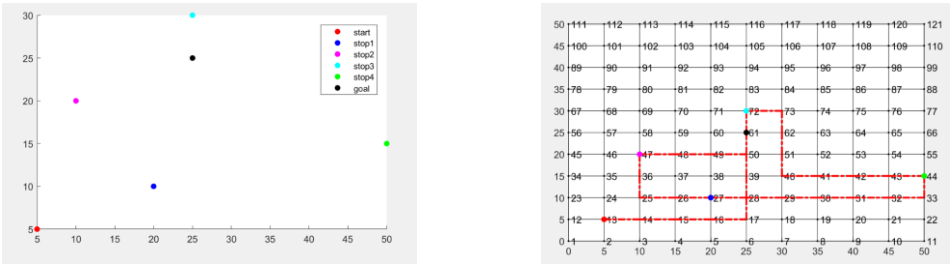

Figure 1-2: The solution to subproblem 1(Fig 2) with given inputs (Fig 1)

Subproblem 2

Trajectory Stitching/Continuity Conditions: It is important to note, after solving for the optimal route we complete some additional computational steps that review the overall trajectory to filter out undesirable behaviors and break the trajectory into segments and parameters that will be used in the dynamic trajectory generation. We search the total trajectory for movements that would correspond to a U-turn and recompute subproblem 1 if re-planning is necessary. We separate each path between two waypoints into a segment with initial and final velocities known to be zero. Furthermore, we separate each of these segments into a subset of segments where new segments are generated by the vehicle's need to make a turn. For this subset, a known velocity of zero is assigned to the beginning and final node of the subset path. In addition to these constraints on initial and final velocities, we also define a continuity condition. This sets the initial conditions of the following node to be the final conditions of the previous node. Then we use Acado toolkit to plan the trajectory for each segment.

Dynamic Trajectory Generation:

Control Model: For our dynamic trajectory optimization we used a simple car model with the following dynamics and constraints. Initial and final constraints are as described above and create continuity between each segment.

$$\begin{aligned} \dot{x} &= v * \cos(\theta) & 0 \le x \le 60 \\ \dot{y} &= v * \sin(\theta) & 0 \le y \le 60 \\ \dot{\theta} &= v * \tan(\omega) & 0 \le v \le 5 \\ \dot{v} &= u_1 & -\frac{\pi}{4} \le \omega \le \frac{\pi}{4} \\ \dot{\omega} &= u_2 & -2 \le \dot{\theta} \le 2 \\ & & -\frac{\pi}{6} \le \dot{\omega} \le \frac{\pi}{6} \end{aligned}$$

Figure 3. Dynamics and Constraints of the System

Cost Function: We chose to optimize our dynamics to minimize our least-squared term. Our choice of least-square term was u1,u2, and v which represent the linear and angular acceleration and the linear velocity. These terms had respective weights of 2,1, and 1.

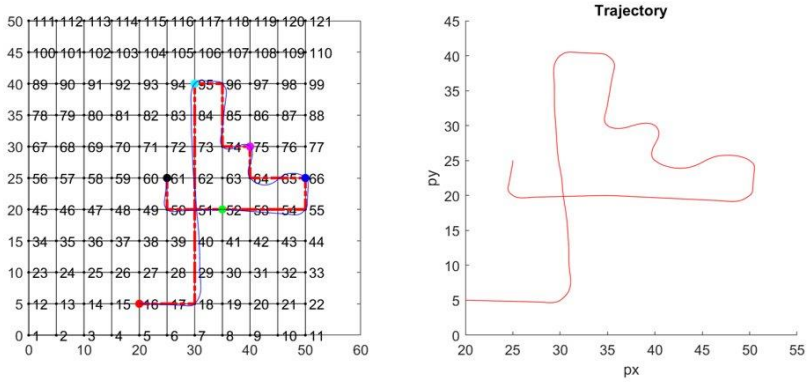

Figure 4-5: The output of Optimal Trajectory with dynamics (Fig 5) which is overlaid onto the solution to subproblem 1 (Fig 4) to show closeness to the ideal straight-lined path

Results: To understand the performance of the trajectory optimization we look at identifying the different patterns in output parameters. The closeness to the straight-line trajectory can be observed in Fig. 4. In addition, the error or deviation from the final position can be seen along the route in Fig. 7. As we reach the end of the trajectory, this goes down to zero. We also look at a comparison of the control effort over the trajectory. A notable pattern of tangent like curves can be seen for u1 and similar peaks for u2 is seem in many versions of the output. In addition, we also look at the comparison of the velocities, both angular and linear (Fig. 8.). A pattern of quick ramping and deramping periods to a steady value peak is seen in many versions of the output for v and w.

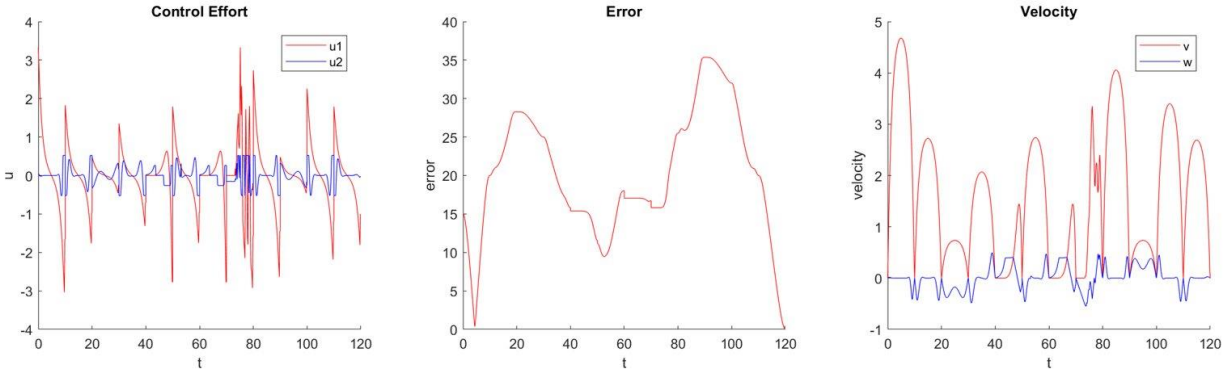

Figure 6-8: The control output vs. time (Fig. 6.), the deviation from the of the path from the final position (Fig. 7.), and velocity (angular and linear) vs. time (Fig. 8.)

Subproblem 3

Controller Design: After calculation of the optimal trajectories and control sequences, a nonlinear controller is designed to track the desired trajectory. We choose to use feedback linearization by taking the third derivative of the desired output Y = [x y].

$$\begin{bmatrix} x''' \\ y''' \end{bmatrix} = \underbrace{\begin{bmatrix} \mathrm{Cos}[\theta[t]] & -\mathrm{Sec}[\delta[t]]^2 \mathrm{Sin}[\theta[t]] V[t]^2 \\ \mathrm{Sin}[\theta[t]] & \mathrm{Cos}[\theta[t]] \mathrm{Sec}[\delta[t]]^2 V[t]^2 \end{bmatrix}}_{A} \begin{bmatrix} u1'[t] \\ u2[t] \end{bmatrix} +$$

$$\underbrace{\begin{bmatrix} -2\mathrm{Sin}[\theta[t]]\mathrm{Tan}[\delta[t]]u1[t]V[t] + V[t]\left(-\mathrm{Sin}[\theta[t]]\mathrm{Tan}[\delta[t]]u1[t] - \mathrm{Cos}[\theta[t]]\mathrm{Tan}[\delta[t]]^2 V[t]^2\right) \\ 2\mathrm{Cos}[\theta[t]]\mathrm{Tan}[\delta[t]]u1[t]V[t] + V[t]\left(\mathrm{Cos}[\theta[t]]\mathrm{Tan}[\delta[t]]u1[t] - \mathrm{Sin}[\theta[t]]\mathrm{Tan}[\delta[t]]^2 V[t]^2\right) \end{bmatrix}}_{B}$$

Figure 9. Third Derivative of desired output Y

For this system, we choose the desired output v as x’’’ and y’’’. By setting

$$V = Yd''' - 5(Y-Yd) - 5(Y'-Yd') - 8(Y''-Yd'')$$

The error dynamics of the linear part is stable. Then control is calculated by:

$$[u1';u2] = inv(A) * (V- B)$$

To complete the dynamics, we use dynamic extension of the original system by adding u1 as a state. Now we can drive the vehicle to track a desired trajectory.

Here are the results of tracking a fifth polynomial trajectory. The error is in term of distance, equivalently the norm of errors.

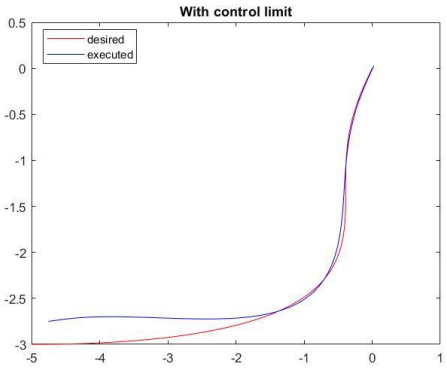

Figure 10. Trajectory

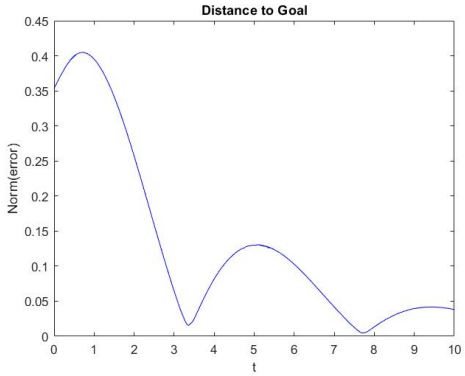

Figure 11. Error

Singularity: Because the control is computed by inverse (A) * (V-B), at some states the matrix A can be singular and not able to invert. To handle, these cases, we enforce the velocity to be at least 0.1 so matrix A will not lose rank at the end of each segment and if the determinant of A is smaller than 10^-3, we add a small offset of [1 1;1 1]*10^3 to make the matrix invertible.

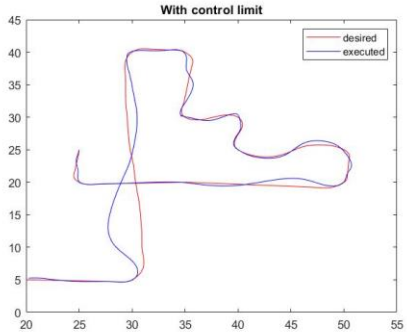

Figure 12. Following Desired Trajectory

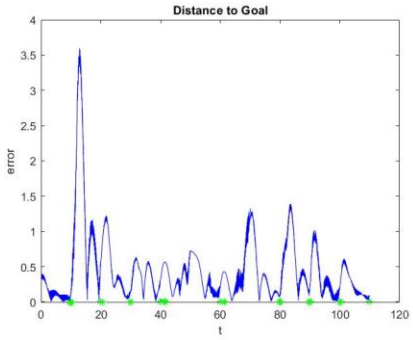

Figure 13. Error

The green dot in Figure 9 is the end point of each segment and error is close to zero, the vehicle can stabilize and follow the desired trajectory.

Results: The results of our testing demonstrated to us that dynamic planning over many segments can be very difficult. Finding a cost that achieves desired performance and allows Acado to converge on each segment takes some experimentation. The parameters and cost options we explored are detailed in Appendix A. In addition, the dynamic planning portion of the project took a large amount of computation effort and time.

When comparing the desired trajectory (derived from the subproblem 1 and 2) and executed trajectory (derived from subproblem 3) the controller is able to track the trajectory fairly well. Running the code

with a variety of different trajectories we do sometimes find that the controller runs into errors with output choices for velocity. In the trajectory generation we specify a constraint on the linear velocity directly as we want only forward motion of the vehicle. However, because linear velocity is not one of the control variables (control variables are angular and linear acceleration), we are not able to constrain the controller to this desired behavior, which can in some cases cause deviation from the original trajectory.

There is also a spot that can be improved on right after the first segment where the vehicle makes the first turn. The vehicle will lose track of the desired trajectory for a short period of time and the maximum error in distance is around 3.5 meters. The most likely reason is that the steering angle theta is greater than theta/2 at some moment so the dynamic is not the authentic movement of the model. Currently we don't have a good solution to this issue.

Conclusion: This project focuses on solving the dynamic ride sharing problem. Solving the problem with random orientations of stopping position is too complex so the states of each stops are simplified. Dijkstra's algorithm and dynamic programming together provide a valid solution to minimize the distance between all the way points. Multiple shooting solves the minimal control effort and linear velocity between each stop. A nonlinear controller designed by feedback linearization can track the desired trajectory. Combining these three subproblems, we optimize the dynamic ride sharing problem with minimum distance and minimum control effort.

Extension of Project: If given additional time the goal would be to more directly tie the three subproblems together to make one solution. Doing this would not require as much simplification in the problem. It would be ideal to combine the dynamic optimization in with the route planning as we could optimize given a cost of the full trajectory with dynamics considered rather than a cost purely based on distance which is what our current solution implements. However, solving dynamic optimization using multiple shooting with Acado toolkit requires much computational resources; using it to solve for the current trajectory already requires a large amount of time, so it would be significantly more computational complex if we were to try and solve for each node to node connection to implement the dynamic cost into our project. Further exploration into different control methods, such as backstepping and Lyapunov redesign, would also be beneficial and a comparison between the outputs would allow us to understand which controller design method is best suited for the diverse problem presented in this project.

References:

Faiq Izzuddin Kamarudin (2019). Travelling Salesman Problem by Dynamic Programming (https://www.mathworks.com/matlabcentral/fileexchange/54744-travelling-salesman-problem-by-dynamic-programming), MATLAB Central File Exchange. Retrieved May 18, 2020.

Joseph Kirk (2019). Dijkstra's Shortest Path Algorithm (https://www.mathworks.com/matlabcentral/fileexchange/12850-dijkstra-s-shortest-path-algorithm), MATLAB Central File Exchange. Retrieved May 18, 2020.

Kobilarov, M(2019). Hw6_car_template (https://asco.lcsr.jhu.edu/en530-603-f2019-applied-optimal-control). Retrieved May 18, 2020.

Kobilarov, M(2020). Uni_flat_fl https://asco.lcsr.jhu.edu/en530-678-s2020/). Retrieved May 18, 2020.

Appendix A: Evaluation and comparison of different cost choices

In selecting the cost to be used for the trajectory generation algorithm, there were several parameters to consider. Ultimately, we narrowed down the parameters to the two most important qualities of the final cost. First, the ability for Acado to converge given the cost equation and weighting. Second, the ability to keep the trajectory as close as possible to the "ideal" straight lined path shown in Fig. 2. The second parameter represents the ability of the car to follow the straight-line path down the desired route; thus, including it in the cost will aim to minimize the drift from this ideal path. The weighting of each of these parameters was set to equal importance. The selection of the cost equation came from attempting different costs listed in Table 1 and evaluating their performance to meet the two parameters mentioned above. Ultimately a cost function based on u1,u2 and v performed the best and was selected to be used in the final dynamic optimization.

	u1+u2	**u1+u2+v**	**u1+u2+v*w**	**u1+u2+err**	**err**
Ability to converge in Acado	5	5	5	1	1
Drift from path	2	3	2	5	4
Total	7	8	7	6	5

Table 1. Evaluation of several different cost selections on important parameters, the cost with the best performance was selected to be used in the final computation of desired trajectory

Appendix B: Derivation of Feedback Linearization Controller

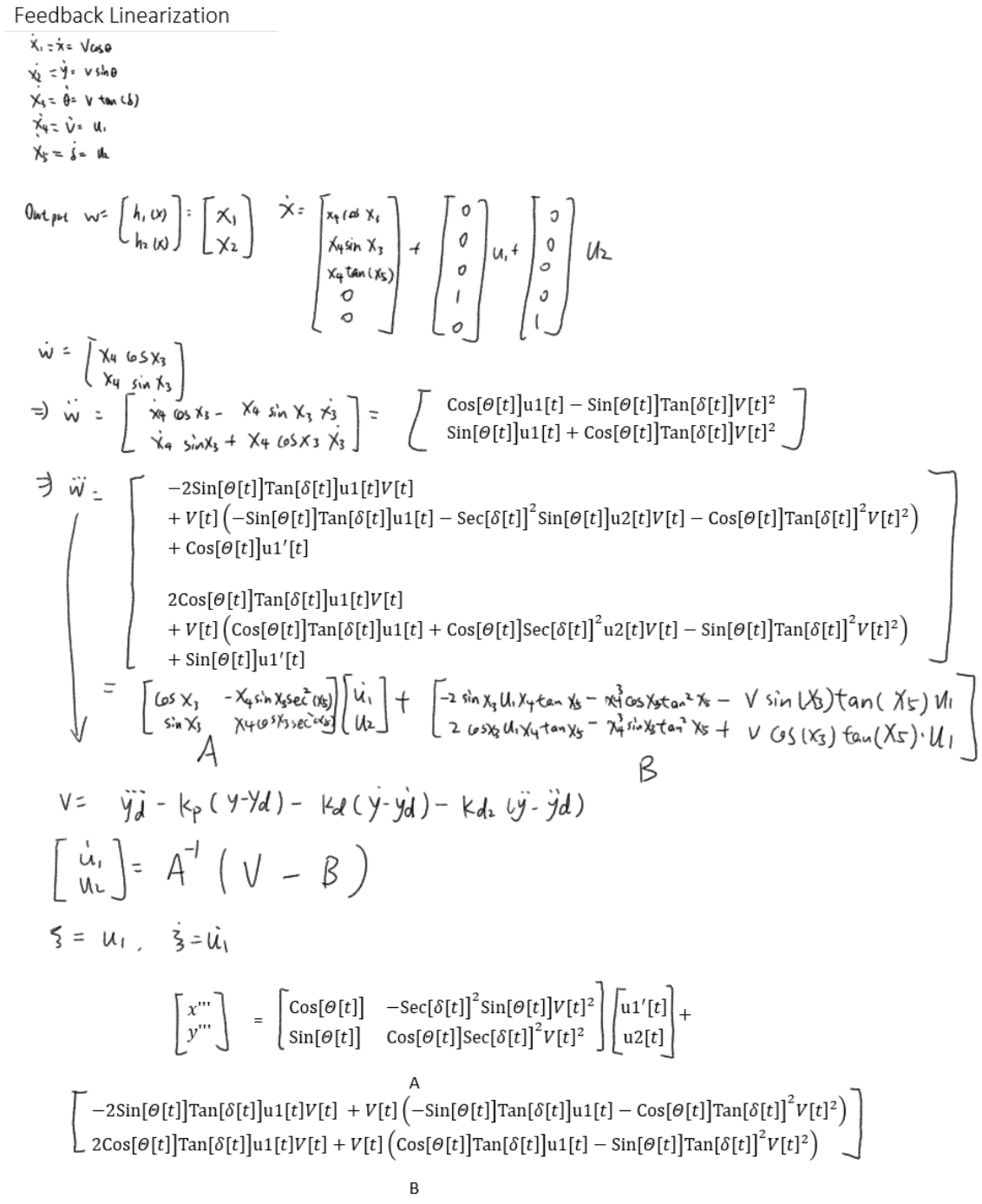

Figure 14: Derivation of controller using feedback linearization method