Fast and Safe Tracking

Nathaniel Nyberg

In this project, I was interested in better understanding HJ Reachability and how it can be used to help solve autonomous navigation problems. I decided to try to implement FaSTrack: a Modular Framework for Real-Time Motion Planning and Guaranteed Safe Tracking developed by the hybrid systems lab at Berkeley[1].

Fastrack guarantees collision free planning and tracking by rigorously computing a maximum tracking error, allowing for a safe trajectory to be generated that takes this error bound into account by either augmenting obstacles or the size of the planning model itself. In order to do this, an error bound and control strategy are computed offline. The computed error bound and control strategy can then be used to both plan and track trajectories with guaranteed safety.

1 Offline Precomputations

In order to compute the maximum error bound, we must choose a planning model and algorithm that are simple enough to allow for real time planning. Furthermore, the planning algorithm must be constrained by the dynamics of the planning model. For explanation purposes, the planning model is chosen to be a 2D single integrator with the following dynamics:

$$\dot{p} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{1}$$

The tracker model must appropriately capture the dynamics and constraints of our real life/simulated system. In this case, we will use a Dubins car tracking model with the following dynamics:

$$\dot{s} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{2}$$

What we care about, is the relative system between the tracker and planner $r = s - Qp$, where Q is a matrix that maps p to the state space of s. We also can also add disturbances to our relative model, resulting in the following relative dynamic system:

$$\dot{r} = \begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \end{bmatrix} = \begin{bmatrix} v\cos\theta - u_x + d_x \\ v\sin\theta - u_y + d_y \\ \omega \end{bmatrix} \tag{3}$$

Now that we have the relative system dynamics, we can can calculate the maximum tracking error bound by setting up a pursuit evasion game, where the tracker acts as the pursuing agent and the disturbance plus the planning model act as an evading agent. We set up a value function $V(r)$ for which we wish to capture the deviation from our plan. This can be as simple as the distance from the origin of the relative dynamic system:

$$V(r) = \sqrt{x_r^2 + y_r^2} \tag{4}$$

The pursuit evasion game can be set up such that the disturbance plus planner model maximizes the value function and the tracking model minimizes it. Using HJ Reachability analysis, we can solve this value function over time according to the pursuit evasion game. If the control authority of the tracking system is powerful enough to always reach the planner, the value function will eventually converge to a value function $V_\infty(r)$. Any level set of this value function is then an acceptable error bound, however we generally wish to take the minimum error level set as it is the minimum possible safe error bound, given the planner model and maximum disturbance values.

At the same time, a safety controller can be generated from the $\nabla V_\infty(r)$ according to equation 5, below. The safety controller guarantees that the tracker will stay within the error bound level set, given that the tracker starts within the error bound level set. It is also possible to implement some other performance controller and only use the safety controller when near the edge of the error bound.

$$u_s^* = \arg\min_{u_s} \max_{u_p,d} = \nabla V_\infty(r) \cdot \dot{r} \tag{5}$$

2 Online Planning and Tracking

The online trajectory planning and tracking algorithm is as follows in algorithm 1.

```
Algorithm 1: Online Trajectory Planning
1: Initialization:
2: $p = s = r = 0$
3: $\mathcal{B}_p(0) = \{p : V_\infty(0) \leq \underline{V}\}$
4: while planning goal is not reached do
5:     Tracking Error Bound Block:
6:     $\mathcal{O}_{aug} \leftarrow \mathcal{O}_{sense} + \mathcal{B}_p(0)$
7:     Path Planner Block:
8:     $p_{next} \leftarrow j(p, \mathcal{O}_{aug})$
9:     Hybrid Tracking Controller Block:
10:    $r_{next} = s - Qp_{next}$
11:    if $r_{next}$ is on boundary $\mathcal{B}_p(0)$ then
12:        use safety controller: $u_s \leftarrow u_s^*$ in (18)
13:    else
14:        use performance controller:
15:        $u_s \leftarrow$ desired controller
16:    end if
17:    Tracking Model Block:
18:    apply control $u_s$ to vehicle for a time step of $\Delta t$,
       save next state as $s_{next}$
19:    Planning Model Block:
20:    $p = Q^T s_{next}$
21:    check if $p$ is at planning goal
22:    reset states $s = s_{next}, r = 0$
23: end while
```

Note that Algorithm 1 is taken directly from M. Chen et al [1] and that the reference to equation 18 in line 12 actually refers to equation 5 in this document. The Tracking Error bound can be applied to either the the obstacles, as indicated in line 5 of Algorithm 1, or the planning model itself. Applying it to the obstacles requires the augmentation to be applied whenever a new obstacles is observed, which can be computationally expensive. This method does, however, enable easier collision checking in the planning step.

In this implementation, I used an RRT based algorithm called RRT connect for trajectory generation as proposed in [2]. The algorithm uses two trees that alternate between randomly exploring as in standard RRT and attempting to connect to the most recently generated node in the other tree. This approach is generally able to find paths faster than the base RRT algorithm as long as the environment is relatively open (obstacles are relatively sparse). The path is then smoothed using a line of sight algorithm from the start node to remove unnecessary nodes from the path. A trajectory can then be generated from the resulting path according to the constraints of the planner.

Note that the straightline, piecewise path that generally outputs from simple planners, such as RRT and A^* based planners, can be used to make valid trajectories as the precomputed error bound accounts for maximally suboptimal planning movement. No splining to make the curve continuously differentiable is necessary. This is one of the great advantages of this framework and algorithm– very fast, simple planners can be used with safety guarantees.

3 Cited Resources

1. M. Chen et al., "FaSTrack: a Modular Framework for Real-Time Motion Planning and Guaranteed Safe Tracking," in IEEE Transactions on Automatic Control, doi: 10.1109/TAC.2021.3059838.

2. J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), 2000, pp. 995-1001 vol.2, doi: 10.1109/ROBOT.2000.844730.