Motion planning and obstacle avoidance with barrier functions for autonomous car racing

Yifei Gao, Enrique Mallada

Abstract

Motivated by the need to simultaneously guarantee safety and maximize progress through path planning for autonomous systems, we demonstrate one application of autonomous driving on F1tenth [13] racing simulator that uses algebraic techniques to optimize the racing path and certifies safety of the car by calculating barrier function with SOS (sum-of-squares programming) [14, 15, 23] simultaneously. Inspired by the idea of Hierarchical control to optimize race track and perform real-time obstacle avoidance with barrier functions for dynamic systems, [10][1] we implement and guarantee both safety and track progress on F1tenth simulation by combining a high-level algorithm for hierarchical controller with real-time calculated barrier function as our lower-level algorithm.

1 Introduction

In this report we consider one application combining path planning method and polynomial optimization in F1tenth simulation [13] on ROS (Robot Operating System) [18]. While methods for generating the optimal path of dynamic system have found wide applications such as dynamic programming for automatic driving RC cars [2] [6]. In the field of polynomial optimization, which has shown rapid advancements, may still have unexplored applications. Many research papers have proved that polynomial optimization plays an important role of guaranteeing safety and stability of controlled dynamic systems by introducing barrier certificates. [16] [22] However, some of them require large computation and cannot simultaneously guarantee safety of the system. Here we demonstrate one application with sum-of-squares programming combined with integer programming to achieve safety and track progress for autonomous car along a certain track in real-time.

The report is organized as follows. In Section 2, we introduce the basic definition of barrier function which can guarantee safety of a dynamic system. We use polynomial optimization technique including sum-of-squares programming and SDP (semidefinite programming) described in Section 2.3 and 2.4 to generate a valid barrier certificate. In Section 3, we firstly demonstrate a simple car dynamic model that will be implemented through this application. The reason why we implement this model is that it has the same control input as car model in f1tenth simulation environment. We describe one feasible algorithm for vehicle path planning around the track by setting up the integer program and solve it in real-time. We call it higher-level path planning. In Section 3.4-3.5, we make use of barrier certificates and generate polynomial inequalities. We show the corresponding polynomial inequalities can be converted to sum-of-squares programming(SOS). And we also demonstrate SOS can be transferred into SDP problem and solved by standard solver SCS [12] in CVXPY package [5]. Optimization over non-negative polynomials procedure represents lower-level part of our algorithm.

In Section 4, we combine both higher-level with lower-level part and implement the algorithm described at the end of Section 3 in f1tenth simulation environment. Finally in the Section 5, we discuss the experimental results and how to improve the algorithm through parallel computing [8]. We also show how to calculate and manipulate convex obstacles constraints and bounded disturbances on control dynamic system.

2 Preliminaries

2.1 Barrier function

Consider a general differential equation:

$$\dot{x} = f(x), \tag{1}$$

where $x \in S$ is the state vector and $\dot{x}$ denotes the derivative of state x with respect to time. Here the set S denotes the set which contains all the possible states x and f is a polynomial function . The barrier certificate [16] $V(x)$ needs to satisfy

$$V(x) \geq 0 \quad \forall x \in S_0 \tag{2}$$

$$V(x) < 0 \quad \forall x \in S_{unsafe} \tag{3}$$

$$\frac{\partial V(x)}{\partial x} f(x) \geq 0 \quad \forall x \in S \tag{4}$$

so that the safety of the system is guaranteed. S_0 denotes the set contains all the initial states of x and S_{unsafe} denotes unsafe set of states x. $S_0, S_{unsafe} \subset S$. Equivalently, the definition of barrier certificates can be modified as the following, [1]

$$V(x) < 1 \quad \forall x \in S_{safe} \tag{5}$$

$$V(x) > 1 \quad \forall x \in S_{unsafe} \tag{6}$$

$$\frac{\partial V(x)}{\partial x} f(x) \leq 0 \quad \forall x \in S \tag{7}$$

Suppose we are given two semi-algebraic sets which are S_{safe} and S_{unsafe}, condition in (7) guarantees that any trajectory staring in safe set S_{safe} will never end up in unsafe set S_{unsafe} since the value of V is actually non-increasing along trajectories. Here note that $\frac{\partial V(x)}{\partial x} f(x)$ can also be simplified as $\dot{V}(x)$ indicating the chain rule. If there exists such $V(x)$ satisfying (5)-(7) conditions, we can guarantee the safety of the system.

2.2 Stability and region of attraction

Suppose we have a differential equation $\dot{x} = f(x)$, which has an equilibrium point at the origin satisfying $(f(0) = 0)$ and it is locally asymptotically stable. Let $\psi(t; x_0)$ denote the state trajectory of the system starting from x_0. According to [22], the RoA($\mathcal{O}$) of the origin is defined as the set of all initial states starting from x_0 which finally converges to the origin when time goes to infinity.

$$\mathcal{O} = \{x_0 \in S | \lim_{t \to \infty} \psi(t; x_0) = 0\}, \tag{8}$$

In fact, Lyapunov's stability theorem [7] indicates that if there exists a Lyapunov function V:$\mathbb{R}^n \to \mathbb{R}$ which satisfies

$$V(x) > 0 \quad \forall x \neq 0 \tag{9}$$

$$\dot{V}(x) < 0 \quad \forall x \in \{x \,|V(x) \leq \beta, x \neq 0\} \tag{10}$$

then conditions (9)-(10) can be used to estimate RoA(region of attraction). Here β is the sublevel of the Lyapunov function $V(x)$ and sublevel set $\{x \,|V(x) \leq \beta, x \neq 0\}$ is one part of RoA. Further research shows that such RoA can be expanded by barrier certificates [22] and proves that the expanded RoA obtained from barrier certificates is at least as large as the one obtained from Lyapunov theory. If we parameterize V as a polynomial function and convert dynamic model $f(x)$ to polynomials, then the search for such $V(x)$ satisfying the conditions (9)-(10) is an optimization problem over the set of non-negative polynomials. We use SOS programming [14] [15] [23] to deal with such inequalities.

2.3 Sum-of-squares(SOS)

Consider a polynomial function [19],

$$f(x) = x^2 + 8x^4 \tag{11}$$

and given a polynomial basis vector $b(x) = \begin{bmatrix} x, x^2 \end{bmatrix}^T$and a 2*2 matrix Q which is the coefficient of the SOS polynomial, $Q = \begin{bmatrix} q1 & q2 \\ q3 & q4 \end{bmatrix}$. Define a quadratic function of the basis $b^T Qb$ and let $f(x) = b^T Qb$. Clearly, if we let $q1 = 1$, $q2 = q3 = 0$ and $q4 = 8$, Q is positive definite and we get $f(x) = b^T Qb = \|(x, \sqrt{8}x^2)\|^2 \in SOS$. Therefore, after defining a polynomial basis $b(x)$, we are searching for a positive semidefinite (PSD) matrix Q such that $f(x) = b^T Qb \in SOS$.

2.4 Converting SOS to SDP (Semidefinite Programming)

Semidefinite programming [3, 19] is a convex optimization problem and it is defined as,

$$\min_{X \in S^n} \text{Tr}(CX), \tag{12}$$

$$\textbf{s.t. } X \succeq 0, \tag{13}$$

$$\text{Tr}(A_i X) = b_i, \ \forall i = 1, ..., N \tag{14}$$

where $X \in S^n$ is a symmetric matrix and also the optimization variable in the problem. $X \succeq 0$ denotes that X needs to be positive-semidefinite. $C, A_1, ..., A_N \in S^n$ and $b_1, ..., b_N \in \mathbb{R}$ are problem data. Finally, the trace operator Tr is the sum of diagonal elements in the square matrix. In the numerical results section, we use SCS solver [12] in CVXPY [5] package for solving SDP problem. Back to the section 2.3, a sufficient condition for $b^T Qb \in SOS$ is $Q \succeq 0$ [4], then we have $b^T Qb = b^T L^T Lb = \|Lb\|^2 \in SOS$ where the rows of L are the eigenvectors of Q multiplied by the square roots of eigenvalues of Q. We can actually form a SDP program to find Q such that $f(x) = b^T Qb \in SOS$,

$$\min_{Q \in S^n} 1, \tag{15}$$

$$\textbf{s.t. } Q \succeq 0, \tag{16}$$

$$q1 = 1, \tag{17}$$

$$q2 = q3 = 0, \tag{18}$$

$$q4 = 8 \tag{19}$$

CVXPY package in python [5]can be used to check if there exists such Q that satisfies the constraints.

3 Method

3.1 Modeling dynamics

Consider a general car model [9] which has three degrees of freedom and its velocity space at any configuration is on x-y dimension. A simple car is treated as a rigid body that moves in the plane. The configuration of car is denoted by $q = (x, y, \theta)$ which are x,y position and global heading of the car θ respectively. As indicated on Figure 1, the car's body frame is located at the rear axis of the vehicle and x axis is pointing along the main axis of the car. We assume car model is front-wheel drive and ϕ is the front wheel steering angle. The distance between front and rear wheel is denoted by L. We know if the steering angle is fixed, the car will travel in a circular motion and circle's radius is denoted by ρ. Let w denote the distance travelled by car which is the integral of speed s, we know $dw = \rho d\theta$ and $\rho = L/tan\phi$ from trigonometry. Two equations implies

$$d\theta = \frac{\tan\phi}{L} dw \tag{20}$$

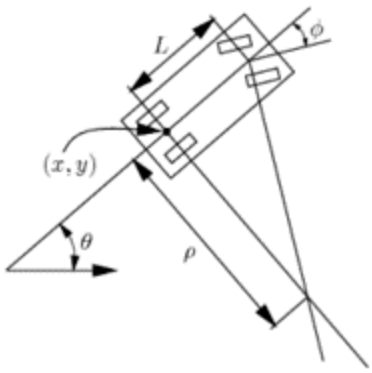


Figure 1: dynamic model of a simple car

Dividing both sides by dt and we have,

$$\dot{\theta} = \frac{s}{L} \tan \phi \tag{21}$$

Here we define an input vector $u = (u_s, u_\phi)$ of the car dynamic system and $u_s = s$. Let u_s denote input speed of the vehicle and u_ϕ denotes steering angle of the front wheel. The dynamic model of a simple car can be written as

$$\dot{x} = u_s \cos \theta, \tag{22}$$

$$\dot{y} = u_s \sin \theta, \tag{23}$$

$$\dot{\theta} = (u_s/L) \tan u_\phi \tag{24}$$

We use this dynamical model in the f1tenth simulation [13] and we only focus on steering angle input u_ϕ between $-\pi/2$ and $\pi/2$.

3.2 High-level path planning

Inspired by the Hierarchical Receding Horizon Controller [10], we take advantage of its higher level path planning part to optimize the race track. The general goal of high-level path planning is to find a nominal trajectory that maximizes the progress along the track. Based on the nonlinear dynamic system above, we can set u_s to be stationary velocity. For a certain stationary velocity u_s, we can choose multiple steering angle u_ϕ between $-\pi/2$ and $\pi/2$, then a trajectory table of zero acceleration can be generated. For example, if we fix velocity input u_s to be 1.0m/s, and we take five different steering angle u_ϕ, five nominal trajectories can be generated by integrating the simple car model. In the experiment, we simulate 21 stationary points for each nominal trajectory for a certain velocity u_s. Here we should notice that the trajectory table does not change during simulation so we can generate them offline to save computation time. After generating the nominal trajectories, we can proceed with finding the trajectory which maximizes the progress along the track by setting up the following integer program,

$$\max_{j \in 1,\ldots,M_0} P^*(X_N^j, Y_N^j), \tag{25}$$

$$\textbf{s.t. } x_0^j = x, \tag{26}$$

$$x_{k+1}^j = f_{km}(x_k^j), \tag{27}$$

$$x_k^j \in \mathcal{X}_{track} \tag{28}$$

where $x_k = (X_k, Y_k, \theta_k)$ is the position and vehicle heading by integrating the discrete version of dynamic model f_{km} of a simple car. Here we let k = 1,...,N and N denote as the total number of points integrated to

form one nominal trajectory. M_0 denotes total number of nominal trajectories on the trajectory table and x_0^j denotes the initial position of the car. The objective (16) is trying to find the trajectory with the largest progress on the track denoted by progress operator $P^* : \mathbb{R}^2 \to [0, L]$ that calculates the track progress of position X_N^j, Y_N^j onto the blue centerline on Figure 2. Here L represents the total length of the centerline.

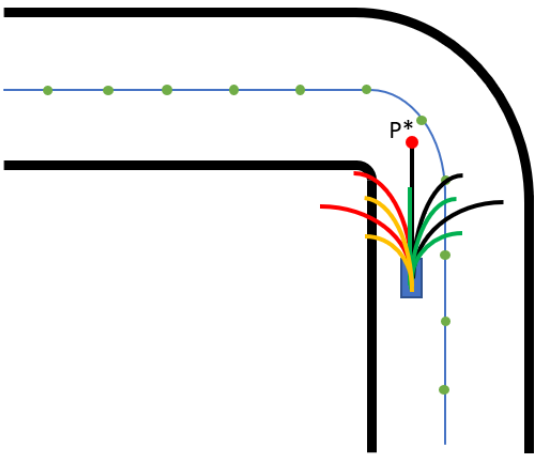


Figure 2: Nominal trajectories and progress

3.3 Choose a trajectory

We generate nominal trajectories on different velocity input u_s based on the car's body frame and pick candidate trajectories that stay inside the track boundary to satisfy (19) in the optimization problem. Here collided trajectories are indicated as yellow and red on Figure 2. Green points are equally spaced points on the centerline and each of them has a data structure (x, y, p) representing position and progress of along the centerline. Assume car is going counter-clockwise, we need to choose a starting green point and set its corresponding progress p to zero. And for each green point on the counter-clockwise direction, its progress p will be the previous green point's p plus the distance between these two green points. To compare the progress, we also need to find the closest line segment formed by two green points on the centerline with respect to each end point on the nominal trajectories.

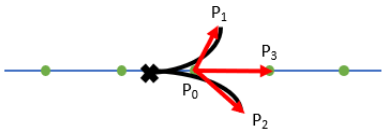


Figure 3: end points have same closest line segment

Figure 3 denotes two candidate nominal trajectories that share the same closest line segment (P_0, P_3) for end points P_1 and P_2. We let projection of vector $\overrightarrow{P_0P_1}$ over line segment (P_0, P_3) be $Pr = |\frac{\langle P_1 - P_0, P_3 - P_0 \rangle}{\|P_3 - P_0\|}|$ where $\langle , \rangle$ denotes as inner product. Similarly, we can calculate end point P_2 and choose the nominal trajectory with the largest magnitude of projection value.

3.4 Obstacle avoidance with Sum-of-squares programming

On the lower-level of the algorithm, our goal is to make our autonomous car avoid obstacles by calculating barrier function for each control primitives. Since we know input vector u is denoted as $u = (u_s, u_\phi)$ which u_s, u_ϕ represents input speed and desired steering angle of the vehicle respectively. Here we let $u_s = 1.0m/s$ and specify five control primitives $u_i = (1.0, u_{\phi_{des},i})$, and we choose $u_{\phi_{des},1} = 0$ rad , $u_{\phi_{des},2} = -20\pi/180$ rad, $u_{\phi_{des},3} = 20\pi/180$ rad, $u_{\phi_{des},4} = -45\pi/180$ rad and $u_{\phi_{des},5} = 45\pi/180$ rad. We Taylor expand the dynamics of the simple car to degree 3 to obtain the polynomial dynamics, then we can formulate the following inequalities to search for a polynomial function $V(\mathbf{x})$ of degree 4.

$$V(\mathbf{x}_0) = 0, \tag{29}$$

$$V(\mathbf{x}) > 1, \ \ \forall (x, y) \in X_{obs} \setminus (x_0, y_0), \tag{30}$$

$$\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}, u_i) < 0, \ \ \forall \mathbf{x} \in X \tag{31}$$

Here we parameterize $V(\mathbf{x})$ to degree 4 so its corresponding basis b has degree 2 and $V(\mathbf{x}) = b^T Q b$. Denoting the initial state of the vehicle as $\mathbf{x}_0 = (x_0, y_0, \theta_0)$ and state of the vehicle as $\mathbf{x} = (x, y, \theta)$ in the global coordinate system. The origin and x-y axis pointing direction of the global coordinate system mentioned above are same as the loading map's origin and x-y direction in the f1tenth simulation environment. Obstacle set is denoted as $X_{obs} \subset \mathbb{R}^2$ and it does not include initial x-y position of the vehicle in the global coordinate system. During the simulation, we can simultaneously localize the vehicle in global coordinate system by accessing the odometry topic. However, we need to use localization algorithm such as particle filter localization [20] to localize our vehicle globally when we implement in the physical car. X is a large set that dynamic system is guaranteed to remain in. The conditions above imply that the state vector $\mathbf{x}$ is only allowed to evolve within the 1-sublevel set of the polynomial function $V(\mathbf{x})$. For any current state $\mathbf{x}_0$, we need to solve at most five optimization problems to search for polynomial function $V(\mathbf{x})$ since we have five control primitives. We want to see if executing such control primitive u_i will result in colliding with the obstacles in a desired steering direction $u_{\phi_{des},i}$. Conditions in (29)-(31) can be transferred into the following SOS programming.

$$V(\mathbf{x}_0) = b_0^T Q b_0 = 0, \tag{32}$$

$$(V(\mathbf{x}) - (1 + \epsilon)) - J g_{obs} \in SOS, \tag{33}$$

$$-\dot{V}(\mathbf{x}, u_i) \in SOS, \tag{34}$$

$$J \in SOS \tag{35}$$

Here $\epsilon > 0$ is a constant and $g_{obs} = \{(x, y) : g_{obs}(x, y) \geq 0\}$ denotes a semi-algebraic set. For instance, we can simulate 2-D circle obstacles during implementation which can be easily converted as $g_{obs} > 0$. Suppose there is a circle obstacle with radius $r = 1.0$ located at $(4.0, 2.0)$ in the global coordinate system. Then we have $X_{obs} = \{(x, y) \in X | 1.0 - (x - 4.0)^2 - (y - 2.0)^2 \geq 0\}$. J is a multiplier and we need to parameterize it with a specific degree such as 2 with a polynomial basis b. If there exists a polynomial function $V(\mathbf{x})$ satisfies the conditions (32)-(35), system is guaranteed to be safe in the current state. These conditions can be converted to SDP with coefficient matching similar to section 2.4.

3.5 Convert to SDP via coefficient matching

We parameterize $V(\mathbf{x}) = b_1^T Q b_1$ where vector $b_1 = \left[1, x, y, \theta, x^2, xy, y^2, x\theta, y\theta, \theta^2\right]^T$, then Q is a 10×10 coefficient matrix for V and (x, y, θ) represents vehicle state vector. For multiplier J, we let $J = b_2^T N b_2$ where $b_1 = [1, x, y, \theta]^T$ and we force $N \succeq 0$. Since a sufficient condition for checking if polynomial function is sum-of-squares is determined by whether its coefficient matrix is symmetric and positive semi-definite or not. Therefore, on condition (33) above, we parameterize the left side $(V(\mathbf{x}) - (1 + \epsilon)) - J g_{obs} = b_1^T M b_1$ where M is the coefficient matrix and we let $M \succeq 0$. Similarly, we let $-\dot{V}(\mathbf{x}, u_i) = b_1^T G b_1$ and force $G \succeq 0$. Then, we can setup the following SDP problem and search for $V(\mathbf{x})$ satisfies the conditions above,

$$\min_{N,M,G\in S^n,Q} 1, \tag{36}$$

$$\textbf{s.t. } V(\mathbf{x}) = b_1^T Q b_1, \tag{37}$$

$$J = b_2^T N b_2, \tag{38}$$

$$(V(\mathbf{x}) - (1+\epsilon)) - J g_{obs} = b_1^T M b_1, \tag{39}$$

$$-\dot{V}(\mathbf{x}, u_i) = b_1^T G b_1, \tag{40}$$

$$N, M, G \succeq 0 \tag{41}$$

Here we need to notice that $V(\mathbf{x})$ might be negative from condition (29)-(31) so we cannot enforce $Q \succeq 0$. For coefficient matching in equation (39) and (40), we search for the corresponding coefficient associated with the same monomial on both sides of the equation. If the coefficient does not exist on one side for a certain monomial, we force the coefficient to be zero on the other side of equation of that monomial. The optimization problem can be solved with SCS solver [12] in CVXPY package [5]. The solving process is similar to primal-dual interior point method [3]. Detailed solving algorithm can be seen on the original paper. [11]

We summarize the method in Algorithm 1 by combining both higher-level and lower-level parts. The algorithm accepts data inputs including dynamic model $f(\mathbf{x})$, constant speed, current state of the vehicle, desired steering angles and threshold for solving time. The Algorithm 1 returns a specific control input $u = (u_s, u_\phi)$ where u_s is the constant speed and u_ϕ is the constant steering angle returned by either integer program or SOS program on Algorithm 1. $\psi_{candidates}$ stores the candidate trajectories which do not go out of the track (excluded best trajectory obtained from integer program). At any moment, we also need to check if vehicle is close to any obstacles. If vehicle is far away from any obstacle, we set steering angle associated with the best trajectory. Otherwise, we determine whether vehicle should execute that steering angle by solving SOS program and calculate its solving time. If solving time t exceeds the pre-defined threshold, we need to find out a candidate direction ϕ and make vehicle steer to that direction.

Algorithm 1: Hierarchical obstacle avoidance with BC1

```
Data: f(x), speed, x_0, φ_{des,i}, threshold
Result: return control input u = (u_s, u_φ)
while True do
    u_s ← speed;
    ψ_candidates ← empty list;
    Solve integer program and get φ_best;
    Store candidate trajectories into ψ_candidates;
    Check if vehicle is close to any obstacle (x, y) ∈ X_obs;
    if not close then
        u ← (u_s, φ_best)
    end
    else
        Solve SOS program on φ_best and get solving time t;
        if t < threshold then
            u ← (u_s, φ_best)
        end
        else
            Solve first available φ in ψ_candidates with SOS;
            u ← (u_s, φ)
        end
    end
    execute control u
end
```

Algorithm 2: Hierarchical obstacle avoidance with BC2

```
Data: f(x), speed, x_0, φ_{des,i}
Result: return control input u = (u_s, u_φ)
while True do
    u_s ← speed;
    ψ_candidates ← empty list;
    Solve integer program and get φ_best;
    Store candidate trajectories into ψ_candidates;
    Check if vehicle is close to any obstacle (x, y) ∈ X_obs;
    if not close then
        u ← (u_s, φ_best)
    end
    else
        Solve SOS program on φ_best;
        if available then
            u ← (u_s, φ_best)
        end
        else
            Solve first available φ in ψ_candidates with SOS;
            u ← (u_s, φ)
        end
    end
    execute control u
end
```

4 Numerical Results

We generate five nominal trajectories based on mentioned simple car dynamics above. Each nominal trajectory represents a constant steering angle. Since each nominal trajectory is generated in discrete version, we specify a time step $dt = 0.04$ and let x_{k+1} be new state vector on the next time step. Then we have $x_{k+1} = x_k + f(x,u) * dt$ where $f(x,u)$ represents the dynamic model of the simple car and $u = (u_s, u_{\phi_{des}})$. We set $u_s = 1.5m/s$ and $u_{\phi_{des}} = -45\pi/180, -20\pi/180, 0, 20\pi/180, 45\pi/180$ rad. We can also generate different nominal trajectory set on these steering directions for each fixed u_s as needed. The nominal trajectories on five directions are shown on Figure 4. Here we should notice that the starting position of each trajectory is the origin because nominal trajectories are always generated from origin of car's body frame which is located at the rear axis of the car.

We implement the higher-level path planning algorithm in F1tenth simulation environment [13] in ROS(Robotics operating System) [18]. Two pictures in Figure 5 show how high-level path planning algorithm generate optimal nominal trajectory indicated in green dot. The fixed steering angles for generating nominal trajectories are the same as those shown on Figure 4. The black lines in both sub-figures are parts of the track boundaries and the vehicle is travelling counter-clockwise. Blue line is the reference line which we calculate the best nominal trajectory from the integer program. It does not have to be the exact centerline shown on Figure 2. It can be any closed-loop line on the track that enables the vehicle to follow. The high-level path planning algorithm can be run in more than 10Hz which is real-time efficient.

We also plot the 1-sublevel of calculated $V(\mathbf{x})$ on Figure 6 based on SOS program. For each fixed steering direction listed on the legend, we need to compute its corresponding constraints offline and append these constraints to the optimization problem for each nominal direction. The current state of the vehicle in Figure 6 is $(x_0, y_0, \theta_0) = (-12.8, 0.3, 0.0)$ which means vehicle's heading is directly pointing toward the obstacles located at $(-10.820, 0.320)$ and has a radius $r = \sqrt{0.4}$. The circle obstacle can be converted to semi-algebraic set $X_{obs} = \{(x,y) | r - (x + 10.82)^2 - (y - 0.32)^2 \geq 0\}$ and green dot in the Figure 6 is the origin of car's body frame located at the rear axis of the vehicle and here we treat our car object as a point. In the real simulation, we should take length between car's front and rear wheels into consideration. For instance, after obtaining the global position of the vehicle, we can calculate the coordinates located at the middle of the

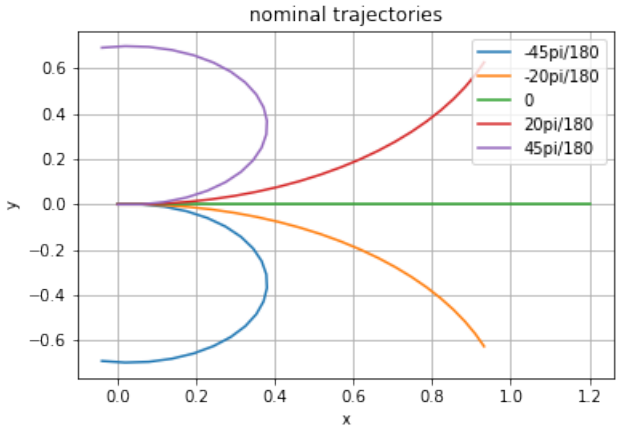
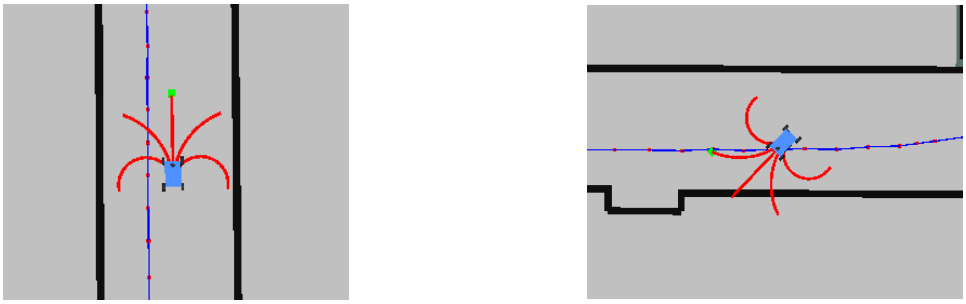


Figure 4: Nominal trajectories



(a) Going straight maximizes the track progress

(b) Slight right turn maximizes the track progress

Figure 5: Green points indicate best nominal trajectory in both vehicle state, red lines denote five nominal trajectories and blue line is "centerline"

two front wheels by generating rotation matrix [17].

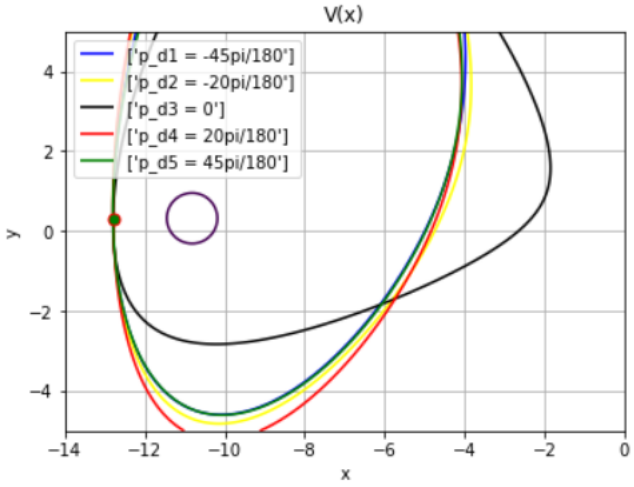
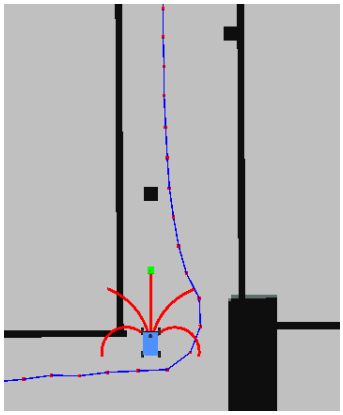
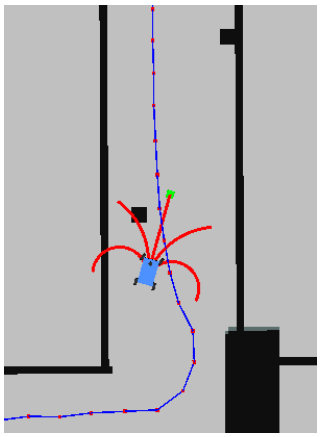


Figure 6: 1-sublevel set of V(x) for five nominal steering directions

You may notice that it seems the current state of the vehicle (x_0, y_0) denoted as green dot in the Figure



(a) Vehicle's optimal direction pointing toward the obstacle



(b) Vehicle steers to the right to avoid obstacles

Figure 7: Pictures indicate when solving time of optimal direction indicated in green dot exceed threshold, vehicle needs to find a candidate direction and steers to that direction

6 lies directly on the 1-sublevel of the generated $V(\mathbf{x})$. But actually current state of the vehicle lies on the 0-sublevel of $V(\mathbf{x})$ since we enforce the equality constraint $V(\mathbf{x}_0) = 0$ in the optimization problem. Therefore, 1-sublevel of $V(\mathbf{x})$ actually separates current state of the vehicle and circle obstacle. We can also add more circle obstacles to the optimization problem, it just needs to satisfies the condition (33) and we need to define new multiplier $J_i = b_2^T N_i b_2$ and let $N_i \succeq 0$. Then we can modify condition (33) into the following,

$$(V(\mathbf{x}) - (1+\epsilon)) - J_i g_{obs_i} \in SOS \quad \forall (x,y) \in X_{obs_i} \tag{42}$$

where g_{obs_i} indicates semi-algebraic inequality and $X_{obs_i} = \{(x,y)|g_{obs_i} \geq 0\}$. Based on the condition (42), we can make sure that 1-level separates initial state of the vehicle and any obstacle taken into consideration. Through experimental results, it turns out that if the nominal steering direction drives the vehicle toward the obstacle, it takes more time to compute a $V(\mathbf{x})$ satisfies the constraints above compared with the situation when vehicle is far away from the obstacle. It means when vehicle is travelling toward the obstacle, it takes more iterations to force zero duality gap during the optimization procedure. The maximum iterations of SCS solver [12] is set to 10000. Experimental results show that when vehicle is in the obstacle, solving SOS program on any direction will reach maximum iterations and still does not satisfy the required tolerance. Optimal status is reached when primal and dual variables satisfy primal and dual feasibility respectively and duality gap is less than certain defined tolerance. For our simulation, we set solving time threshold to be less than 100ms to indicate it's a valid steering direction for current state of the vehicle.

We simulate two point obstacles indicates as black pixels on figure 7. We treat them as circle obstacles with radius $r = \sqrt{0.4}$ located at $(-10.82, 0.32)$ and $(-8.54, 0.84)$ respectively. When vehicle is getting close to any of the obstacle listed above, we append its corresponding constraints to the optimization problem and calculate its solving time. We do not implement parallel computing during the simulation. It means solving SOS program in the lower-level part of the algorithm suspends the update of optimal nominal direction on the higher-level part. It can be handled by parallel computing [8] which enables the system to solve integer program and SOS problem independently. Furthermore, we highly recommend using parallel computing when best nominal steering direction is not safe, it enables the system to find the first available steering direction ϕ around candidate trajectories $\psi_{candidates}$ shown in algorithm 1. For the higher-level part, we can also simulated more nominal directions which make vehicle follow the reference line smoothly.

5 Discussion and future work

As we implemented the algorithm 1 described on method above, the most challenge we encountered is computation efficiency. Solving process of SCS solver [12] in CVXPY package [5] includes both setting up time and solving time. In the coefficient matching of SDP, there are almost hundreds of terms to match if we parameterize $V(\mathbf{x}) = b_1^T Q b_1$ with degree 4 and multiplier $J = b_2^T N b_2$ with degree 2. Experimental results show that setting up time is at least 0.2 seconds and actual solving time may vary depending upon current vehicle situation. We also try to detect obstacles with 2-D LiDAR in the simulation [13]. We use LiDAR detection to form convex hull of obstacles and transform coordinates of LiDAR points from laser frame to global coordinates with convexhull package in Scipy [21]. Recall a polyhedron is the intersection of finite number of closed half spaces,

$$P_{polyhedron} = \{\mathbf{x} \in \mathbb{R}^2 \mid \alpha_i^T \mathbf{x} \leq b_i,\ i = 1, ..., n\} \tag{43}$$

where n represents total number of half space constraints to form the polyhedron. Then we can form the matrix inequaility $A\mathbf{x} \leq \mathbf{b}$. Each row in $A \in \mathbb{R}^{n \times 2}$ is α_i^T and $\mathbf{b} \in \mathbb{R}^n$ where each entry in $\mathbf{b}$ is b_i. Let vector $\mathbf{G}(\mathbf{x}) = \mathbf{b} - A\mathbf{x}$ that each entry in vector $\mathbf{G}(\mathbf{x})$ is non-negative. Then we can convert it to semi-algebraic set $g_{polyhedron} = \{\mathbf{x} \in X \mid \mathbf{G}(\mathbf{x}) \geq \mathbf{0}\}$. Conditions for searching for a valid $V(\mathbf{x})$ can be modified as the following to handle polyhedron obstacle,

$$V(\mathbf{x}_0) = b_0^T Q b_0 = 0, \tag{44}$$

$$(V(\mathbf{x}) - (1 + \epsilon)) - \mathbf{J}^T \mathbf{G}(\mathbf{x}) \in SOS, \tag{45}$$

$$-\dot{V}(\mathbf{x}, u_i) \in SOS, \tag{46}$$

$$J_i \in SOS,\ \ \forall i = 1, ..., n \tag{47}$$

$\mathbf{J}$ is vector of multipliers containing J_i. We parameterize each $J_i = b^T N_i b$. N_i is the coefficient matrix for each multiplier and we enforce each multiplier to be sum-of-squares. However, SDP problem generated from conditions (44)-(47) needs to be changed whenever LiDAR detects and form new polyhedron equation and CVXPY package cannot change the obstacle constraints completely during run time. If this issue can be solved, we can apply LiDAR detection and generate convex obstacles instead of circle obstacles. Furthermore, we did not take noise into consideration this time. To deal with noise, we can use the same approach mentioned in the paper [1]. Assume there is an uncertain "cross-wind" term $\mathbf{w}$ bounded between $[-0.05, 0.05]$ in the dynamics, then we can modify condition (31) in section 3.4 to the following,

$$\dot{V}(\mathbf{x}, \mathbf{w}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}, u_i, \mathbf{w}) < 0,\ \ \forall \mathbf{x} \in X,\ \forall \mathbf{w} \in [-0.05, 0.05] \tag{48}$$

then solution of sum-of-squares programming generate valid $V(\mathbf{x})$ with noise term w presenting on the dynamics.

6 Conclusion

In this report, we demonstrate one application of two optimization problems on F1tenth simulation environment [13]. The higher-level part of algorithm enables vehicle to find a nominal trajectory that maximizes the progress along the track. The lower-level part is solving sum-of-squares programming by defining the set of non-negative polynomials. We also demonstrated how SOS programming can be converted into a SDP program and use solver package to handle the problem. Finally, we indicated that computation efficiency and robustness of the algorithm 1 can be improved by introducing parallel computing and LiDAR detection for obstacles.

References

[1] Amir Ali Ahmadi and Anirudha Majumdar. *Some Applications of Polynomial Optimization in Operations Research and Real-Time Decision Making*. 2015. DOI: 10.48550/ARXIV.1504.06002. URL: https://arxiv.org/abs/1504.06002.

[2] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957. ISBN: 9780486428093.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] *Decomposition of a positive semidefinite matrix*. https://math.stackexchange.com/questions/1801403/decomposition-of-a-positive-semidefinite-matrix. Accessed: 2022-05-04.

[5] Steven Diamond and Stephen Boyd. "CVXPY: A Python-embedded modeling language for convex optimization". In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[6] Andreas Huber and Matthias Gerdts. "A dynamic programming MPC approach for automatic driving along tracks and its realization with online steering controllers **This material is partly based upon work supported by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under Award No, FA9550-14-11-0298." In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 8686–8691. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2017.08.1529. URL: https://www.sciencedirect.com/science/article/pii/S2405896317321158.

[7] Hassan K Khalil. *Nonlinear systems; 3rd ed.* The book can be consulted by contacting: PH-AID: Wallet, Lionel. Upper Saddle River, NJ: Prentice-Hall, 2002. URL: https://cds.cern.ch/record/1173048.

[8] Vipin Kumar. *Introduction to Parallel Computing*. 2nd. USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN: 0201648652.

[9] S. M. LaValle. *Planning Algorithms*. Available at http://planning.cs.uiuc.edu/. Cambridge, U.K.: Cambridge University Press, 2006.

[10] Alexander Liniger, Alexander Domahidi, and Manfred Morari. "Optimization-based autonomous racing of 1:43 scale RC cars". In: *Optimal Control Applications and Methods* 36.5 (July 2014), pp. 628–647. DOI: 10.1002/oca.2123. URL: https://doi.org/10.1002%2Foca.2123.

[11] Brendan O'Donoghue. *Operator splitting for a homogeneous embedding of the linear complementarity problem*. 2020. DOI: 10.48550/ARXIV.2004.02177. URL: https://arxiv.org/abs/2004.02177.

[12] Brendan O'Donoghue et al. "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding". In: *Journal of Optimization Theory and Applications* 169.3 (June 2016), pp. 1042–1068. URL: http://stanford.edu/~boyd/papers/scs.html.

[13] Matthew O'Kelly et al. *F1/10: An Open-Source Autonomous Cyber-Physical Platform*. 2019. DOI: 10.48550/ARXIV.1901.08567. URL: https://arxiv.org/abs/1901.08567.

[14] A. Papachristodoulou and S. Prajna. "A tutorial on sum of squares techniques for systems analysis". In: *Proceedings of the 2005, American Control Conference, 2005.* 2005, 2686–2700 vol. 4. DOI: 10.1109/ACC.2005.1470374.

[15] A. Papachristodoulou and S. Prajna. "On the construction of Lyapunov functions using the sum of squares decomposition". In: *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.* Vol. 3. 2002, 3482–3487 vol.3. DOI: 10.1109/CDC.2002.1184414.

[16] Stephen Prajna, Ali Jadbabaie, and George J. Pappas. "A Framework for Worst-Case and Stochastic Safety Verification Using Barrier Certificates". In: *IEEE Transactions on Automatic Control* 52.8 (2007), pp. 1415–1428. DOI: 10.1109/TAC.2007.902736.

[17] *Rotation matrix*. https://en.wikipedia.org/wiki/Rotation_matrix. Accessed: 2022-05-07.

[18] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Version ROS Melodic Morenia. May 23, 2018. URL: https://www.ros.org.

[19] *Sum of Squares Programming, Global Non-Convex Optimization, Lyapunov Function and Semidefinite Programming*. https://wang-yimu.com/sum-of-squares-programming/. Accessed: 2022-05-03.

[20] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[21] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[22] Li Wang, Dongkun Han, and Magnus Egerstedt. *Permissive Barrier Certificates for Safe Stabilization Using Sum-of-squares*. 2018. DOI: 10.48550/ARXIV.1802.08917. URL: https://arxiv.org/abs/1802.08917.

[23] Bai Xue, Naijun Zhan, and Yangjia Li. *Robust Regions of Attraction Generation for State-Constrained Perturbed Discrete-Time Polynomial Systems*. 2020. arXiv: 1810.11767 [math.DS].