Feasible Trajectories By Planning Through Pre-Computed Actions

Akshay Jaitly, Ethan Chandler, Hushmand Esmaeilli, Yaşar İdikut

***Abstract*—In this paper, we develop planners for constrained systems using two methods, one in discrete and another combining discrete and continuous. In the discrete method, a state-lattice is constructed by forward simulating shooting-based approach given a range of control inputs. This graph is then searched with and without a heuristic. In the second method, we propose a method which ensures action composability and satisfies dynamic constraints, resulting in a newly devised feasible trajectory model that enables planning over sets of trajectories. Our approach to trajectory planning leverages differential flatness and Bézier curves to transform arbitrary constraint spaces into a common design space. Our trajectory sets are built offline, with each set representing an action in a library. At runtime, the optimal trajectory is selected from the corresponding set based on the global task, making our method well-suited for real-time planning in applications with limited computational resources. We present a simple thought experiment using a swing-up pendulum, and demonstrate the effectiveness of our approach on a differential drive robot in an obstacle field, demonstrating its ability to generate high-quality trajectories in the face of highly non-convex constraints. Our approach offers a promising avenue for enabling robots to perform complex actions in dynamic and constrained environments.**

***Index Terms*—Trajectory optimization, feasible sets, motion planning, composability, differential flatness.**

I. Introduction

Motion planning is an important problem in robotics and automation, with applications ranging from manufacturing and logistics to autonomous driving and exploration. One key challenge in motion planning is the need to generate feasible trajectories that satisfy various constraints, such as collision avoidance, actuator limits, and task-specific requirements. While trajectory optimization techniques have been widely used to address this challenge, they typically require significant computational resources and cannot easily adapt to changing environments or task specifications. In this context, the idea of precomputing a library of feasible trajectories, which can be adapted online for specific tasks, has gained increasing attention in recent years.

Our work is motivated by the need to create a library of feasible trajectories that can be adapted online for use in complex motion planning tasks. We draw inspiration from the "Partners in Parkour" [1] video released by Boston Dynamics, in which Atlas robots perform a variety of behaviors as they traverse an obstacle course. These behaviors were derived from a set of template trajectories created ahead of time using trajectory optimization [2]. Utilizing offline optimization enabled them to incorporate important constraints during the design stage and allowed for their online adaptation. Our research addresses the question of how such a behavior library can be created in the first place, specifically by exploring and proposing methods for designing and characterizing sets of feasible trajectories for constrained, collision-free motion planning.

In this paper, we present our trajectory composability model, along with its validation on a simple swing-up pendulum and a differential drive robot in an obstacle field. Our work provides a framework for planning over sets of feasible trajectories, offering a promising solution for navigating through complex environments while satisfying system constraints.

II. Related Work

The problem of designing continuous collision-free trajectories for robots given constraints has been widely studied [3]. Choosing the right approach requires compromising between different aspects of the problem, including dimensionality, kinematic and dynamic constraints, computation limits, completeness and optimality requirements.

Sampling-based planners. Sampling-based planners are one approach that attempt to address the problem of planning in high-dimensional spaces. In general, this approach consists of randomly sampling the configuration or state space, looking within for collision-free connections. On such approach is Rapidly-exploring Random Tree (RRT) [4]. It allows for fast planning in semi-structured spaces, and can also consider non-holonomic constraints. RRT^* is an improvement to the RRT search algorithm and uses 'rewiring' to asymptotically converge to the optimal path. For a long time, this was the state of the art of planning in high-dimensional graphs, which more primitive planners like Dijkstra's algorithm and A^* cannot handle.

Anytime RRT^* [5] expands on RRT^* by adding a heuristic that removes nodes which do not produce better trajectories, improving the computation costs. Anytime RRT^* introduces two key features known as committed trajectories and branch-and-bound adaptation. The strategy of committed trajectory originates robot's movement to follow the first segment of initially planned path while improving remaining segments of the path using an iterative strategy. Branch-and-bound optimizes the tree for optimal cost. Unfortunately, this algorithm struggles at considering non-holonomic and other dynamic constraints, and produces jagged trajectories in the process.

Informed RRT^* [6] extends RRT^*, and computes the optimal path from a starting point to a goal point in a given environment. It uses an ellipsoid heuristic that gives an estimate of the minimum cost required to reach the goal point from any given point in the environment. This

heuristic is used to bias the sampling process towards the most promising areas of the search space. This approach also suffers from producing jagged trajectories, which does not guarantee dynamic feasibility. It is also expensive in terms of memory computation cost since it requires many nodes and edges to converge to an optimal solution. Although Informed RRT^* is an example of an asymptotically optimal sampling-based planner, the trajectories designed can be considerably suboptimal in practice, since only a finite number of samples can be taken.

A sampling-based algorithm that considers dynamics constraints is LQR-RRT^* [7]. LQR-RRT^* linearizes the dynamics around the state and extends dynamically feasible nodes. This ensures that visiting the resulting path from this tree is dynamically feasible because there is feasibility between each parent and child. This allows the algorithm to search with non-holonomic constraints. Additionally, since the node expansion is dependent on the system dynamics, it can incorporate energy costs and derivative constraints, which are useful for complex or underactuated systems. A downside of this approach is that it is computationally expensive. Another approach in this class is DT-RRT [8]. DT stands for dual tree, workspace, and state. The workspace tree searches for a path without considering robot constraints, while the state tree is expanded based on the robot dynamics/kinematics. This dual tree approach allows for higher success rates for node extensions, saving on computation. However, it is only applicable in known environments.

Direct trajectory optimization. Methods based on direct trajectory optimization can design trajectories in high-dimensional spaces while taking into account the robot kinematics and dynamics. Transcription is the process of turning a trajectory optimization problem into a nonlinear program. Decision variables in the trajectory optimization problem are vector functions which lie in infinite dimensional continuous spaces, which is substantially simpler than the global close-loop problem. In nonlinear programs, decision variables are finite-dimensional real numbers governed by algebraic equations. A downside of these methods is that they cannot guarantee a solution for more complex systems. Collocation methods on the other hand tend to yield a larger non-linear program size than shooting methods, but with better sparsity. These approaches tend to be fast and can be used in higher-dimensional systems, however, they are a local method and solution.

The authors in [9] explore different transcription methods. Basic transcription uses the Euler method to discretize the dynamics, but has a lot of 'drift' or error buildup in the dynamics. Collocation with Buamgarte Stabilization alters the actual system dynamics and adds 'artificial forces' to dampen the trajectory drift. The downside is that it cannot guarantee the dynamic constraints at the actual collocation points. The PKT method modifies the Hermite-Simpson collocation method to deal with constrained systems. The disadvantage to this approach is that at each midpoint, the robot is essentially treated as an unconstrained system subject to forces defined by the Lagrangian multipliers. The PKT method cannot deal with non-holonomic constraints either. Newer transcription methods include projecting the end state of each polynomial to eliminate drift, and satisfies the dynamics at all the collocation points. The downside of all of these methods is that they are slow and cannot guarantee a solution for more complex systems.

Mixed-Integer Convex Programming (MICP) planners. Mixed-Integer Convex Programming (MICP) is a mathematical optimization technique used to solve problems that involve both continuous and discrete variables. In MICP, the objective function and the constraints are expressed as convex functions, which ensures that the optimization problem is solvable efficiently and reliably. MICP planners offer the advantages of both sampling-based algorithms and trajectory optimization methods, while also providing global optimality within a single optimization framework. This means that they can handle robot kinematics and dynamics with ease and guarantee the best possible solution. Therefore, MICP planners are considered to be a promising approach that combines the benefits of both worlds. Kinodynamic RRT^* [10] finds sequentially optimal movements and connects nodes to the path where the resulting path has the lowest cost, like RRT. However, it optimizes a continuous path to connect two nodes. This requires trajectory optimization across the tree, which gets expensive when you have to also evaluate the kinematic and dynamic constraints. Optimality and dynamic feasibility is guaranteed using LQR by solving the Algebraic Riccati Equation.

Mixed integer planning in [11] solves the trajectory optimization problem by finding a set of piece-wise trajectories, similar to RRT^*. However, in this case, a constraint is added that each trajectory must fall within a convex region, and only one trajectory may exist in that convex region. By adding these convex regions to decompose the configuration space, collision-free movement can be guaranteed, and a highly non-convex optimization problem can be broken into multiple convex problems. The problem then involves optimizing trajectories for a system in different regions. The optimization problem has binary variables that determine whether a region will have a trajectory or not. If a region has a trajectory, the trajectory optimization in that region is a convex problem. The optimization problem requires making decisions in both discrete space (determining which regions to plan through) and continuous space (finding the optimal trajectory parameters in each region). However, MICP techniques have limited usage due to their long runtimes, which can take several minutes to design a trajectory even for small-scale problems.

The proposed planning problem formulation in [12] combines Bézier curves and a framework for shortest paths in Graphs of Convex Sets to create a compact mixed-integer optimization. This approach is highly efficient and provides a globally-optimal solution with optimality bounds for planned trajectories. Unlike existing mixed-integer planners, the convex relaxation of the proposed program is very tight, making it possible to reduce the mixed-integer program to a simple convex optimization with a cheap rounding of the solution. One of the methods we propose in this paper is inspired in the hybrid approach of MICP and GCS planners, where a trajectory-optimization planner is driven by a higher-level

graph search.

In this paper, we propose two distinct but related methods for generating feasible trajectories with constraints. Our first method relies on a shooting-based method that takes into consideration the dynamics of the vehicle and environmental constraints. It uses shooting to build a graph of states and associates a cost between nodes in terms of estimated energy spent, which should make this approach near-optimal. Finally, an admissible heuristic is proposed to make the graph-search more efficient. In the second method, we create feasible trajectories training actions offline and use a newly devised trajectory composability model to guarantee feasibility. From these sets, we find pre-conditions and post-conditions of the states. From these, we determine the composability of sets to create a directed graph. Lastly, the constructed graph of sets of feasible trajectories can be used to plan from any node in the graph to another.

III. Proposed Method

A. Method 1: Baseline Solutions for Feasible Trajectories

To serve as a baseline solution to the problem of motion planning given dynamic constraints, we first present a shooting-based state-lattice planner in the discrete state-space.

The proposed shooting-based planner is as follows. Consider a robot with given forward/inverse kinematics and ranges of control inputs for each joint. Given this information, one can extend dynamically feasible nodes by sampling the control inputs and then simulating them forward in time by a constant time step, Δt. The easiest way to approach this would be to uniformly sample the range of each control joint. In the platform used for testing/developing this approach, we are using the bicycle dynamics to represent a car on the road. [13] includes the bicycle model. A modified kinematics with friction can be written as follows:

$$\dot{X} = Vcos(\psi + \beta(u_2))$$
$$\dot{Y} = Vsin(\psi + \beta(u_2))$$
$$\dot{V} = u_1 - (ResistiveTerm)V$$
$$\dot{\psi} = \frac{V}{l_r} sin(\beta(u_2))$$

where *ResistiveTerm* is the resistive term, and $\beta(u_2)$ is the slip angle at the center of gravity:

$$\beta(u_2) = arctan(tan(u_2)\frac{l_r}{l_f + l_r})$$

Consider control input ranges of $[-5\frac{m}{s^2}, 5\frac{m}{s^2}]$ and $[-\frac{\pi}{4}rad, \frac{\pi}{4}rad]$ for u_1 (acceleration) and u_2 (steering angle), respectively. With a resolution of $1.111\frac{m}{s^2}$ and $0.0436rad$, we can have ten uniform samples in both control inputs. Combined, this gives 100 possible states extended from a parent state. By using input u_1 as an estimate for cost, we can turn the dynamically constrained motion planning problem into a discrete graph-search problem. There is the curse of dimensionality as can be seen, however, there are two approaches to make this less of a problem: (1) use lower resolution to reduce dimensionality, and (2) use an informed graph-search algorithm to make the computation faster.

The formulation of the proposed admissible heuristic is as follows. First, we need to make sure that the heuristic never over-estimates the actual cost to goal. For this, we first need to define cost. Following is the actual energy a system uses to travel a distance at constant acceleration, a, for a fixed time step, Δt:

$$EnergyCost(m, a, \Delta t) = F\Delta d = ma\Delta d = ma(\frac{1}{2}a(\Delta t)^2)$$

however, the mass of the car is assumed to not change throughout the graph, so we can ignore the mass term in the graph search. Additionally, if we don't add a time term, the heuristic is always 0 because in our model, the resistant force is never able to have the car come to a stop. So, we need to introduce a path duration term. The cost used to represent an edge is as follows:

$$Cost(V, u_1, \Delta t) = \frac{1}{2}u_1^2(\Delta t)^2 + TimeMultiplier(\Delta t)$$

The proposed heuristic relies upon the shortest-path search of the state in the lower dimension, namely, X and Y. The approach is to use the road network by getting the road segments' start and end points, connecting them based on how the road segments are connected, and finally connect the goal position to the graph by iterating the shortest nodes from goal. Before the full state-space search is conducted, this graph is exhaustively searched starting from the goal node. This way, we have O(1) lookup for the minimum distance from every road network segment/node to the goal position. In the full state-space search, we then get the nearest road network nodes for a given full state-space node candidate, and use the minimum combined distance (from node candidate to the road network node, and from road network node to the goal position) and the following equations to get a lower bound on the cost for the whole motion.

$$minCostD = \min_{u_1} \frac{ResistiveTerm}{u_1}(TimeMultiplier + \frac{1}{2}u_1^2(\Delta t))$$
$$\text{s.t.} \qquad 0 < u_1 < u_{max}$$

$$Heuristic(state) = LookupMinD[state] * minCostD$$

where $minCostD$ is the solution to the optimization problem that, when multiplied with a distance, gives the lower bound on the cost; $LookupMinD[state]$ is the minimum distance from a given state candidate to the goal position. Since the solution to the optimization problem doesn't change throughout the graph search, this can be calculated once before the graph search. In our implementation, we use SciPy's optimization framework to solve this optimization problem.

Finally, since this heuristic always results in lower cost estimation than the actual cost, it is admissible, which makes a graph search using A* directly optimal (with regards to the cost proposed) and resolution-complete, unlike probabilistic approaches like ones from the RRT* family.

B. Method 2: Building Sets of Feasible Trajectories

A trajectory can be represented as a function of design variables, $f(\omega, t)$, where ω is the vector representing the design variables. Using $f(w, t)$, each value of ω can be mapped to a single trajectory in the state space. Thus, ω will be used to refer to any single trajectory for the duration of this paper.

A system imposes constraints on trajectories. These could be constraints on the dynamic feasibility of trajectories, holonomic constraints, or obstacle constraints in the configuration space. Given a vector of inequality constraints, $\vec{g}(\omega)$, the set of possible solutions (Ω) is given by $\Omega = \{\omega \in \mathbb{R}^n | \vec{g}(\omega) \preceq 0\}$ where n is number of design variables.

If $n = 3$, a trajectory ω can be represented as a point in 3 dimensional space. A set Ω can be represented as a volume, where $g_m(\omega)$ (element m of vector $\vec{g}$) represents a separating manifold in the space, characterizing a surface of the volume. There is no guarantee that $g_m(\omega)$ is convex, so Ω is very likely to be a non-convex set.

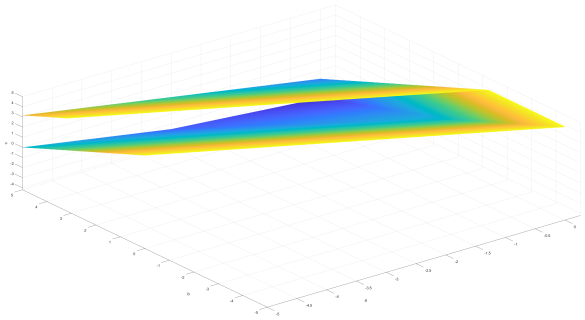

Fig. 1. A set of feasible trajectories, Ω, where the design variables are a,b, and c and $f(\omega, t) = a + bt + ct^2$

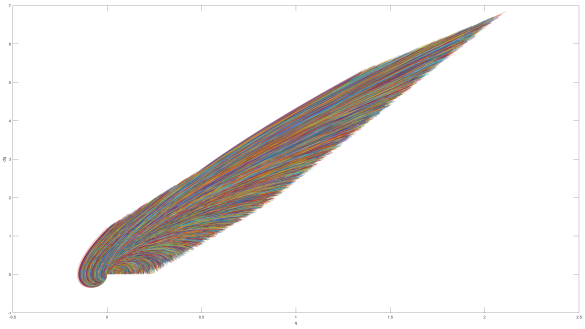

Fig. 2. Actual trajectories from the set

In order to segment this space, more constraints are introduced. $\vec{\phi}_i(\omega)$ is a vector of 'characteristic' constraints. Furthermore, the addition of $\vec{\phi}_i(\omega)$ constraints create subsets, Ω_i.

$$(\Omega_i \subseteq \Omega) = \{\omega \in \mathbb{R}^n | \vec{g}(\omega) \preceq 0, \vec{\phi}_i(\omega) \preceq 0\}$$

Let it be noted that these characteristic constraints are entirely specific to the type of action the user wants to encapsulate. In this paper, we provide several examples of how one might design these characteristic constraints to model desired behavior.

1) Pre-Conditions and Post-Conditions: Each ω in Ω_i corresponds to a starting and ending state, x_0 and x_f, where $x_0 = f(w, 0)$, $x_f = f(w, t_f)$ and t_f is the duration of the trajectory. The set of all starting states for Ω_i are called the Pre-Conditions for Ω_i – denoted by X_{0i}. All ending states are the Post-Conditions, denoted by X_{fi}.

$$X_{0i} = \{f(\omega, 0) | \omega \in \Omega_i\}$$

$$X_{fi} = \{f(\omega, t_f) | \omega \in \Omega_i\}$$

If a desired state (x_{fd}) is not in X_{fi}, then, by definition, no value of ω in Ω_i can be used to plan to x_{fd}. It is important to note that for any x in either X_{0i} or X_{fi}, there may be multiple corresponding trajectories.

C. Method 2: Building Graphs

Two sets, Ω_i and Ω_j, can be composed if a state, x, exists such that $x \in X_{fi}$ and $x \in X_{0j}$. Since a trajectory (ω_1) exists in Ω_i that has x as a post-condition, and a trajectory (ω_2) exists in Ω_j that has x as a pre-condition, we can say that at least one trajectory exists (ω_2 composed onto ω_1) that plans through Ω_i and Ω_j.

This is equivalent to stating that they are composable if $X_{fi} \cap X_{0j}$ is non-empty. It is important to note that, if Ω_i is composable to Ω_j, this does not mean that any pre-condition of Ω_i can be planned into any post-condition of Ω_j. It means rather, that if Ω_i is not composable with Ω_j there is no feasible pair of trajectories through each set that can plan from pre-condition to post-condition.

Using this concept of composability, a library of sets can be represented as a graph, where the nodes correspond to different subsets. The edges of this graph are directed, and represent composability of the nodes it connects to. If an edge is sourced from Ω_1 and directed towards Ω_2, this would represent that Ω_1 has post conditions that intersect with the pre-conditions for Ω_2.

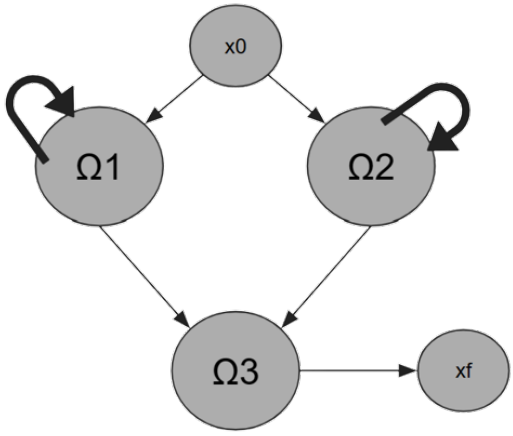

Fig. 3. Composed Sets for a simple pendulum

D. Method 2: Feasible Trajectories

In the example of the simple pendulum, we define three sets of feasible trajectories:

Ω_1 is the set of trajectories in which the angle is monotonically increasing Ω_2 is the set of trajectories in which the pendulum 'swings' back and forth to gain energy Ω_3 is the set of trajectories in which the pendulum is stabilizable about the inverted point.

If a trajectory was desired from x_0, the downwards equilibrium, to x_f, the inverted equilibrium, the composability analysis for graph creation can be performed.

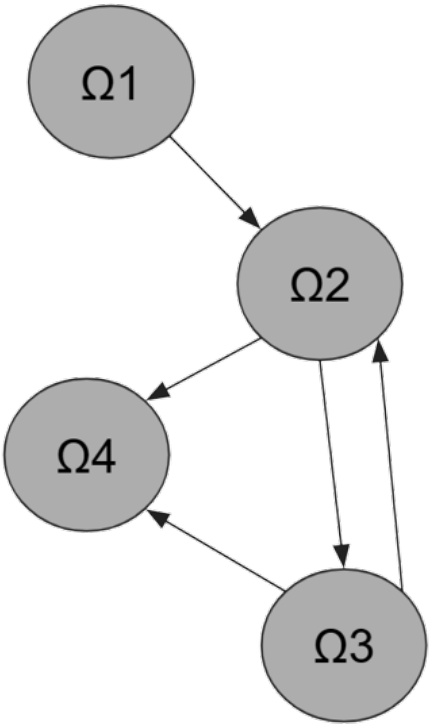

Fig. 4. an example of Composed Sets

The shortest route from x_0 to x_f would appear to be through either Ω_1 or Ω_2, then Ω_3. However, these may not necessarily be feasible trajectories. It could be that there is no trajectory between the state x_0 and the post-conditions that intersect with Ω_3 through either set. Rather, it could be that there could be some $\omega_1 \in \Omega_1$ that connects x_0 with a post-condition that is still in the pre-condition for Ω_1, where a trajectory now exists to the pre-conditions of Ω_3. This is the behavior of energy-shaping, where a 'swing' is followed by another 'swing'. Since the post-conditions of Ω_1 intersect with the pre-conditions, it can be stated that Ω_1 is composable with itself.

IV. Experiment

A. Platform for Method 1

All the testing for the feasible trajectory calculations in the discrete set is conducted in a custom Gymnasium environment using Highway Env [14]. Highway Env project is usually used by ML researchers to train agents using reinforcement learning, however, in this project, we only use the framework and the tooling to simulate and validate our approach for path planning. Following are the modifications made:

1) Custom $SimpleRBE550Vehicle$ class that allow us to specify the dynamics given in Section III.A.
2) Custom $SimpleRBE550Env$ environment to represent the environment
3) An alternative way to construct the road network by straight and circular segmentation that also allows search through the segments' start and end positions.
4) Modified visualizer to show current search progress and the path found.

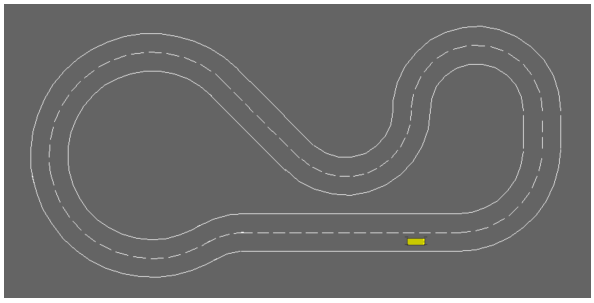

Fig. 5. Example customized racetrack Gymnasium environment created.

Since above modifications required much reverse engineering due to lack of documentation and buggy APIs, the extent to which we were able to compare the path planning problem was as with and without heuristic. So, the state-lattice is expanded using the same cost, however, the priority of expansion is compared between a uniform and informed approach (Dijkstra vs. A*).

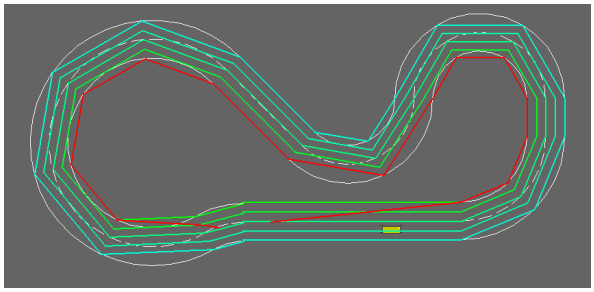

Fig. 6. Visualization of the graph search in the lower dimension (X and Y) in the modified racetrack environment. The ends of the road segments are connected based on the direction of the road, so that the planner never calculates a path in the opposite direction of the traffic flow. The edges of the road network is visualized by the blue-green colors and the red colored path is a solution to an example minimum distance query.

Fig. 7. Visualization of the (uniform) graph search in the full state-space (X, Y, Velocity, Heading, Time) in the modified racetrack environment. Due to computational inefficincies resulting from the curse of dimensionality, this figure shows how far an uninformed search can explore the graph in 10 minutes.

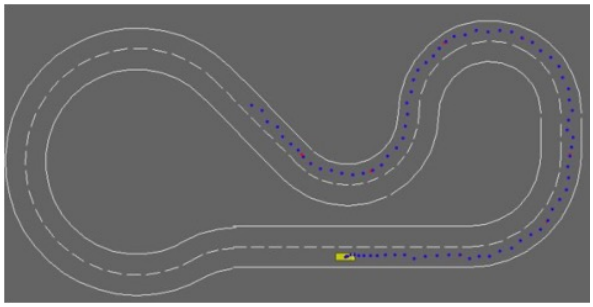

Fig. 8. Visualization of the (informed) graph search in the full state-space (X, Y, Velocity, Heading, Time) in the modified racetrack environment. Given the admissible heuristic, the graph can be explored using less steps. This figure shows a path from initial vehicle position to a randomly picked goal position with constraints mentioned in this section. Calculated in 12 seconds.

A benefit of the discretized shooting-based state-lattice search is the ease of evaluating constraints. In this problem, following constraints are applied:

1) $Velocity > 0$
2) $Velocity < SpeedLimitOfRoadSegment$
3) $|SteeringAngle| > Velocity \ \frac{0.75|MaxSteeringAngle|}{MaxSpeed}$
4) $isOnRoad(State)$

The method for evaluating above constraints are fairly trivial. Basically, for each node expansion given inputs u_1, u_2, and Δt, check the input constraints (item 3), simulate the system foward in time using the forward kinematics model, then check the state constraints (items 1, 2, 4).

Finally, in the time allotted, we were able to implement a baseline solution for the problem of constrained path planning for a car simulated by a modified bicycle dynamics. The stretch plan was to also implement LQR-RRT*, and compare path execution cost and planning time. Given the time constraints, this wasn't implemented. Currently, Dijkstra isn't able to find a path because of dimensionality, and A* with the proposed heuristic is able to calculate a resolution-complete optimal path in 12 seconds as can be seen in Fig. 8.

B. Platform for Method 2

In order to experiment with this technique, it was implemented on a simulated non-holonomic platform — the differential drive vehicle through a Figure-8 course. The robot is given a task to drive from the top of the Figure-8 to a point on the bottom, while staying inside the Figure-8. The states of the robot are the x position, the y position and the heading, θ.

The trajectory is defined as a Bézier curve with 4 control points (determined by 8 design variables), and the duration of the trajectory segment is fixed at 1 second.

The non-holonomic constraint is described by $\vec{g}_1(\omega)$.

$$\vec{g}(x) = \frac{d}{dx}(arctan(\frac{\dot{y}(\omega,t)}{\dot{x}(\omega,t)})) - \epsilon < 0$$

for $t = [0, 1]$, with epsilon being a significantly small value.

Additionally, a constraint was added to enforce the convexity of the trajectory segments (put image of convex Bézier curve curves from the slides here). This constraint states that the two middle control points of each Bézier curve must lie on the same side of the separating hyperplane formed by the two outer control points.

For this implementation, the set of all feasible trajectories is segmented spatially by starting state. Because of this, the characteristic constraints ($\vec{\phi}$) are functions of the trajectory initial conditions.

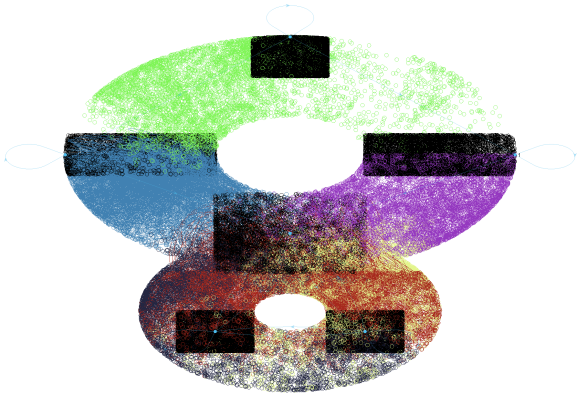

Fig. 9. The figure 8 with the segmented sets where the black regions are the sampled pre-conditions of each trajectory and the colored spaces are the sampled post-conditions. The directed edges are shown as well.

C. Results

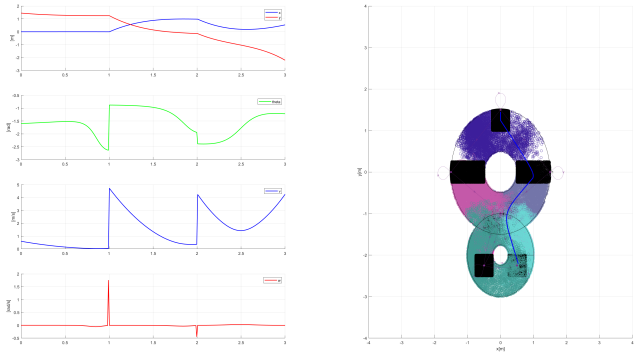

Fig. 10. Feasible trajectory found by IPOPT across the composed regions.

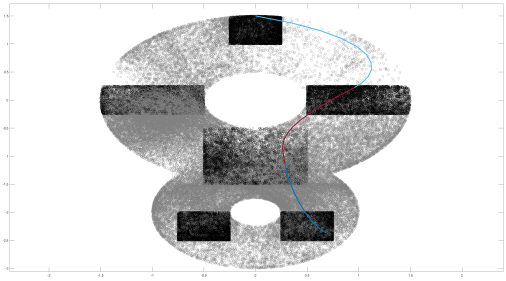

Fig. 11. Feasible trajectory found using fminunc.

We created a Matlab simulation to validate our approach, and used the interior point method to set up the optimization problem over the composable regions.

Our segmented region approach (Fig. 10, 11) is compared with an optimization over the full, unsegmented region. By segmenting the regions using the proposed characteristic constraints, the solver was able to successfully find feasible trajectories in 5-15 seconds, while the unsegmented approach failed to find a feasible trajectory after waiting 5 minutes. In order to validate the correctness of the problem formulation, the tests were conducted on two independently formulated interior point frameworks—one using Casadi and IPOPT (Fig. 10), and the other using Matlab's fminunc (Fig. 11).

V. Conclusion

A. Method 1

The proposed discrete planner for a non-holonomic car modeled by a modified bicycle dynamics is resolution complete and this wouldn't be possible without the proposed heuristic in a timely manner. However, there is still much room for improvement specific to this problem. For an example, the method for checking some constraints, such as $onRoad(State)$, are not O(1) and depends on the size of the road network graph. Additionally, generating a better heuristic will improve the planning time. There exists many tuning variables that allow the practitioner to generate faster or lower energy consuming paths. The constraint specification is also very simple. This can later be compared to LQR-RRT*. Due to the fact that we started working on this 2-3 weeks ago, we couldn't include the comparison for LQR-RRT*.

B. Method 2

The current implementation segments the regions spatially, which limits the reusability of the characteristic sets—in future work we would like to find ways to better characterize these sets. We'd also like to solidify the foundation of our approach by finishing our research with the swing-up pendulum model, and test the planning capabilities on a system with more interesting dynamics (e.g., a quadcopter). Our current work on the swing-up pendulum model includes a Sum of Squares formulation for generating Lyapunov functions and estimating the basin of attraction for arbitrary PD controller gains.

VI. Contributions

	Akshay	Ethan	Hushmand	Yasar
Theory/Foundation	25%	25%	25%	25%
Lit. Review	25%	25%	25%	25%
Approach 1	0%	0%	0%	100%
Approach 2	33%	33%	33%	0%
Presentation/Paper	25%	25%	25%	25%

References

[1] *Atlas — Partners in Parkour*. YouTube, Aug 2021.

[2] "Flipping the script with atlas," Aug 2021.

[3] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, pp. 463 – 497, 2014.

[4] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, pp. 473–479 vol.1, 1999.

[5] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *2011 IEEE International Conference on Robotics and Automation*, pp. 1478–1483, 2011.

[6] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic," *CoRR*, vol. abs/1404.2334, 2014.

[7] A. Perez, R. Jr, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Lqr-rrt* : Optimal sampling-based motion planning with automatically derived extension heuristics," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2537–2542, 05 2012.

[8] C.-b. Moon and W. Chung, "Kinodynamic planner dual-tree rrt (dt-rrt) for two-wheeled mobile robots using the rapidly exploring random tree," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 2, pp. 1080–1090, 2015.

[9] R. Bordalba, T. Schoels, L. Ros, J. M. Porta, and M. Diehl, "Direct collocation methods for trajectory optimization in constrained robotic systems," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 183–202, 2023.

[10] D. Webb and J. van den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," pp. 5054–5061, 05 2013.

[11] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 42–49, 2015.

[12] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," 2022.

[13] P. Polack, F. Altché, B. Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?," pp. 812–818, 06 2017.

[14] E. Leurent, "An environment for autonomous driving decision-making." https://github.com/eleurent/highway-env, 2018.