Motion Planning Under Temporal Constraints

Jeel Chatrola[1] and Dev Soni[1]

***Abstract*— In this project we try to address motion planning problem with spatial and temporal constraints. While, traditionally motion planning is researched thoroughly most algorithms are developed for simple tasks. In this work we explore the domain of planning under timed constraints using Signal Temporal Logic (STL).**

I. INTRODUCTION

Traditionally, the kinodynamic motion planning problem is defined as: Given a dynamical system, an environment, an initial state and a goal state, find a sequence of control inputs so as to drive the system from the initial state to the goal state while fulfilling some constraint dictated by the environment, e.g., avoiding obstacles. But as robotic systems become more complex, it opens up new advanced applications. We need algorithms which can handle more constraints automatically rather than breaking down the problem to simpler task. Timed constraints is similar constraints, this can be helpful in situations where you have applications which are time critical such as surgical robots, cobots and manufacturing applications. If we are able to achieve this we can solve problem along the lines of "Pick up object bottle in 10 seconds, pour the liquid in a glass for 5 seconds and keep the bottle down, do this entire task in 30 seconds".

In this project we try to extend the constraints that a motion planning algorithm can handle. We will use Signal Temporal logic (STL) specifications as it consists of both spatial and temporal constraints on continuous signals. A key benefit of STL is its quantitative semantics – robustness - which provides a score indicating to what extent a signal satisfies or violates a given specification.

II. RELATED WORK

Most of current approaches solve the problem using RRT*, they modifying the cost function using STL formulation to maximize the STL Robustness [1], [2] and [3]. Closest of what we want to achieve is showcased in [3] where they solve the problem using a modified biased sampling and a guided steering for the RRT* algorithm.

Sampling-based methods for planning have already proven useful in a variety of contexts, including temporal logic controller synthesis. Extensions of Rapidly Exploring Random Trees (RRT) [19] and their optimal version, RRT*[20], have been proposed to incorporate specifications in μ- calculus

[21], [22] and LTL [23], [24]. Probabilistic Roadmaps (PRMs) [25] have also been exploited for temporal logic synthesis [26], [27].

[1]Jeel and Dev are with Robotics Engineering, Worcester Polytechnic Institute, MA, USA. {jchatrola,djsoni}@wpi.edu

The Major Flaw in the current approaches, according to our understanding is that all of them revolve around RRT* and RRT - which can become limiting in higher dimensional spaces and the algorithm itself is very complicated due to inclusion of STL semantics.[3][2][1]

III. PRELIMINARIES

A. Signal Temporal Logic Specifications

Signal temporal logic (STL) is a temporal logic formalism for specifying properties of continuous signals. We will define STL in this section. This section is refered from [3]

Given an interval $I = [a, b]$, the interval $[t + a, t + b]$ is denoted by $t + I$. Let (M, d) be a compact metric space with $M \subset R^n, n \geq 1$ and $S = \{s : R_{\geq 0} \to M\}$ the set of all infinite-time signals in M. Components of a signal $s \in S$ are denoted by $s_i, i \in \{1, ..., n\}$ The set of all linear functions over R^n is denoted by $F = \{\pi : R^n \to R\}$. Let $\delta_\tau : S \to S$ be the time-shift operator acting on signals in S with $\tau \geq 0$, i.e., $\delta_\tau s(t) = s(t + \tau)$ for all $t \geq 0$.
The Syntax of STL is defined as:

$$\phi ::= \top \left| p_{\pi(x)\sim\mu} \right| \neg\phi \left| \phi_1 \wedge \phi_2 \right| \phi_1 \mathcal{U}_{\langle a,b\rangle} \phi_2$$

where $\top$ is the Boolean *true* constant; $p_{\pi(x)\sim\mu}$ is a predicate over $\mathbb{R}^n$ parameterized by $\pi \in \mathcal{F}, \mu \in \mathbb{R}$ and an order relation $\sim \in \{\geq, >, \leq, <\}$ of the form $p_{\pi(x)\sim\mu} = \pi(x) \sim \mu$; $\neg$ and $\wedge$ are the Boolean operators for negation and conjunction, respectively; and $\mathcal{U}_{\langle a,b\rangle}$ is the bounded temporal operator until, where $\langle \in \{[, (\}$ and $\rangle \in \{]), \}$.

$$\begin{aligned}
&s \models \top \quad \Leftrightarrow \top \\
&s \models p_{\pi(x)\sim\mu} \quad \Leftrightarrow \pi(s(0)) \sim \mu \\
&s \models \neg\phi \quad \Leftrightarrow \neg(s \models \phi) \\
&s \models (\phi_1 \wedge \phi_2) \quad \Leftrightarrow (s \models \phi_1) \wedge (s \models \phi_2) \\
&s \models \left(\phi_1 \mathcal{U}_{\langle a,b\rangle} \phi_2\right) \quad \Leftrightarrow \exists t_u \in [a, b) \text{ s.t. } (\delta_{t_u} s \models \phi_2) \\
&\wedge (\forall t' \in [0, t_u)\, \delta_{t'} s \models \phi_1)
\end{aligned}$$

where $a, b \in \mathbb{R}, a < b$. A signal $s \in \mathcal{S}$ is said to satisfy an STL formula ϕ if and only if $s \models \phi$. The Boolean value false $\bot \equiv \neg\top$ and additional operations (i.e., disjunction, implication, and equivalence) are defined in the usual way. Also, the temporal operators eventually and globally are defined as $\Diamond_{\langle a,b\rangle}\phi \equiv \top\mathcal{U}_{\langle a,b\rangle}\phi$ and $\Box_{\langle a,b\rangle}\phi \equiv \neg\Diamond_{\langle a,b\rangle}\neg\phi$, respectively. the language associated with Signal Temporal Logic Formula ϕ is the set of all signal in S that satisfy ϕ. In addition to Boolean semantics, STL admits quantitative semantics which is formalized by the notion of robustness degree.

Without adding more complex STL specification math, we will simply define our motion planning problem as a Reach-Avoid Specification along with time bounds, using simple STL semantics.

For example: Reach target region 1 in between $[0, 10]$secs, and reach target region 2 in $[10, 25]$ secs while avoiding obstacles. This can be defined as:

$$\phi = (\Box_{\langle 0,25 \rangle} \neg Obs) \wedge (\Diamond_{\langle 0,10 \rangle} T_1) \wedge (\Diamond_{\langle 10,25 \rangle} T_2)$$

IV. Problem Formulation

We formalize the problem of designing the control policy/trajectory for a given dynamical system to satisfy given STL specification.

Let $\mathcal{R} = (f, X, U, x_{\text{init}})$ be a dynamical system, where $X \subseteq \mathbb{R}^n$ and $U \subseteq \mathbb{R}^m$ are the state and control spaces, $f : X \times U \to X$ is a Lipschitz continuous function, and x_{init} is the initial state of the system. The system behavior is given by:

$$\mathcal{R} : \dot{x} = f(x, u), x(0) = x_{\text{init}}$$

We denote by $\mathbf{x}[x_{init}, u]$ the state trajectory originating at x_{init} obtained by implementing control policy u. Let $\mathcal{U} = \{u : \mathbb{R}_{\geq 0} \to U\}$ be the set of all control policies.

The system $\mathcal{R}$ is said to satisfy an STL specification ϕ under a control policy $u \in \mathcal{U}$ if the state trajectory starting at x_0 satisfies ϕ, i.e., $\mathbf{x}[x_{\text{init}}, u] \models \phi$.

Problem: Given a dynamical system $\mathcal{R}$ and an STL specification ϕ, find a control policy u such that the system satisfies ϕ under policy u.

V. Proposed Approach

Instead of solving for entire class of STL specification which is a difficult problem. We use a sub-class of specification which are capable enough to define the motion planning task at hand. One of properties of obstacle avoidance is inherently embedded as collision check routines in the algorithm.

Now to solve multi-goal task we iteratively pass the sub-tasks to a single-query planner such as Kinodynamic RRT. Next we try to include time constraints for a given dynamical system using Time Robustness. For this we propose a rewire algorithm explained below.

A. Kinodynamic Planning

Part of the problem statement of designing the control policy for a given dynamical system can be framed as Traditional Kinodynamic Planning problem which helps take into account the system dynamics. Instead of planning in the Geometric Space we plan in the State-space of system defined by a differential equation of the form

$$\dot{x} = f(x, u).$$

where, $x \in X$ is state and X is the state space. $u \in U$ is control input and U is the control space. Now, contrary to geometric planning, we perform sampling in the control space and use numerical methods such as Range Kutta to compute approximate trajectory based on system kinematics/dynamics.

Algorithm 1 Kinodyamic-RRT

function RRT(x_s, x_g, Obs, system_model)
  $T \leftarrow$ rooted at x_{start}
  while solution not found **do**
    $x_{rand} \leftarrow$ StateSample()
    $x_{near} \leftarrow$ nearest state in T to x_{rand} as per ρ
    $\lambda \leftarrow$ GenerateLocalTrajectory(x_{near}, x_{rand})
    if IsSubTrajectoryValid($\lambda, 0, step$) **then**
      $x_{new} \leftarrow \lambda(step)$
      add x_{new} and edge (x_{near}, x_{rand}) to T
      if $\rho(x_{new}, x_{goal}) \sim 0$ **then**
        return T from root to x_new
function GenerateLocalTrajectory(x_{near}, x_{rand})
  for i = (1,...,m) **do**
    $u \leftarrow$ ControlSample(U)
    $\lambda \leftarrow x(t) = x_{near} + \int_0^{\Delta t} f(x(\tau), u) d\tau$
    $d_i \leftarrow \rho(x_{rand}, \lambda_i(\Delta t))$

The Only modification to geometric RRT planning is the **Generate Local Trajectory** step for the given system. There are two variations on how we can tackle this problem 1) Two-point boundary value problem. 2) Extend According to random control or Select best from n samples of control input. The problem with first approach is computational expensive and it is possible that we dont find a solution. But in case of approach 2 we can select best possible control u which takes us closer to x_{near}. To generate a trajectory for a selected control input involves solving the differential equation leading to integration. Hence, for this we use Range-Kutta Approximation to find the new state of the system given a control input and time-sample. For check the validity of the trajectory we perform collision check for that specific edge between the nodes before adding the node to our tree.

Drawbacks of this approach is that, the state-space size is very large and due to differential constraints this algorithms doesn't scale when we have challenging scenarios such as geometrically narrow paths and/or less number of solution paths for a specific goal due to control input constraints. Computation of the sub-methods of this algorithm are also not complete in the sense that they are approximate or might not provide the exact solution we were looking for.

But for most of experimental cases where mobility of the system is not restrictive this works well enough, so in this work we use Kinodynamic RRT. More analysis can be found in the Section 6.

B. Time-Rewire Algorithm

The core idea behind this algorithm is to get a approximate trajectory from x_{start} to x_{goal} configuration using Kinodynamic RRT. Using That Trajectory nodes and time-samples we modify the time-samples for their state-control pair. Key idea is to not to sample time in a big interval because that might lead to deviation of newly generated trajectory which might not be able to reach the goal.

Algorithm 2 Rewire Algorithm for Kinodyamic RRT

function REWIRE_RRT_TIME(T, Obs, $Traj\_Time$)
  $T_{modified}$ = [ ]
  for nodes in T **do**
    $t_{b-sample}$ = Inf
    $t_{b-robustness}$ = Inf
    for i in (1, .., n) **do**
      $t_{new} \leftarrow$ random(0,1)
      $dt_m = |(t_{goal} - t_{modified})|$
      $dt_o = |(t_{goal} - t_{orig})|$
      $t_{robustness} = |dt_o - dt_m|$
      if $t_{robustness} < t_{b-robustness}$ **then**
        $t_{b-sample} = t_{new}$
        $t_{b-robustness} = t_{robustness}$
    $\lambda_{new} \leftarrow$ RK4_step($t_{b-sample}$)
    if IsSubTrajectoryValid($\lambda, 0, step$) **then**
      $x_{new} \leftarrow \lambda(step)$
      add x_{new} to $T_{modified}$
  return $T_{modified}$

We iterate over each node in the trajectory, calculate how robust that partial trajectory is with respect to our specification. We keep track of best-time sample with respect to robustness/closeness to our desired time-specification.

Once we find the best time sample among n sampling iterations, we can now use the best time sample to find the new state using Range-kutta approximation. Check if the trajectory is valid, add it to the new Tree.

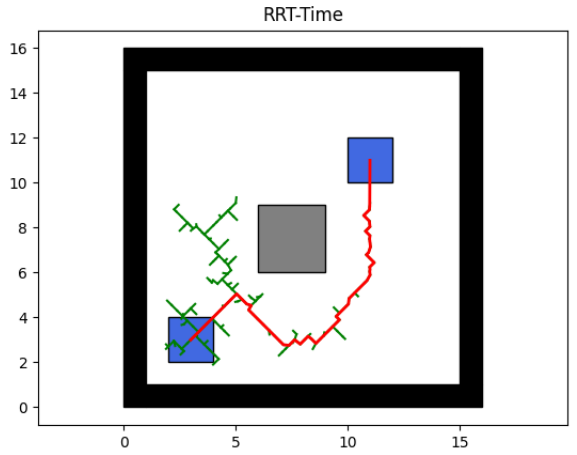
One of the intricacies in this is that the time sample that we collect should not modify the trajectory/node drastically. For example if our control input is very high, so even for a small time sample change we will deviate from trajectory too much. While our if our control input is less then we have much more room in selecting time sample as it will not modify the trajectory drastically.

VI. Analysis and Case Study

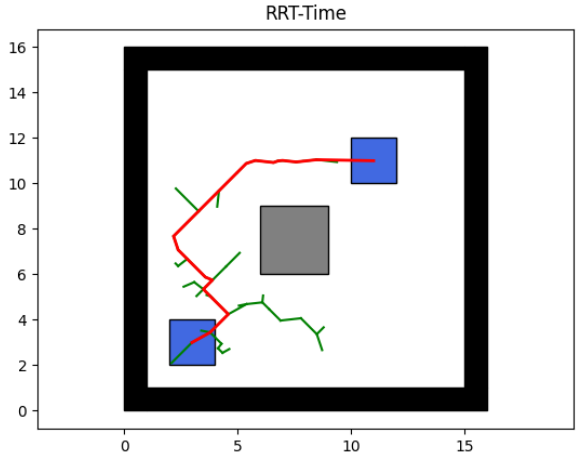
In this section, we analyse the practical effectiveness of the proposed algorithm. Consider a system R whose dynamics are represented by a single integrator.

$$\dot{x} = u$$

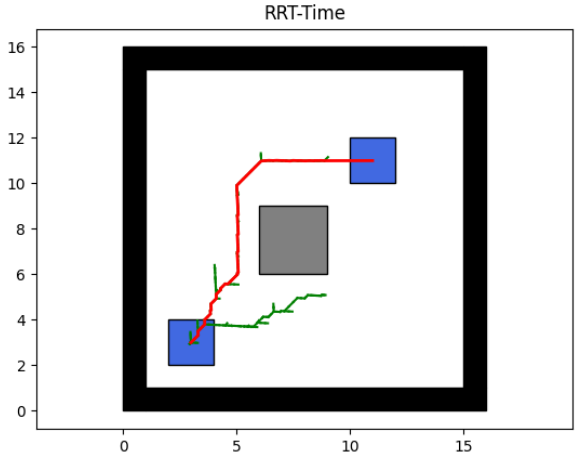
where, input u is bounded as $\|u\| \leq u_{max}$. $x = [x1, x2]$ and $u = [u1, u2]$. Now we perform experiments varying different parameters, such as control bounds and time step.



(a)



(b)



(c)

Figure. Trajectory for given system R_1.

No	U	T	Time to Goal	No of Nodes
a	[-0.5, 0.5]	[0, 0.5] dt = 0.1	18.61	68
b	[-1, 1]	[0, 1] dt = 0.1	10.43	23
c	[-0.1, 2]	[0, 0.1] dt = 0.01	4.99	94

Table shows parameters that are changed in experiment.

We can see that the total time of trajectory in figure a, is highest among the three variations as we have negative control bounds control space. Also time interval for which the control input is applied is also higher. Now in figure b, we reduce the time interval which enable algorithm a finer control over trajectory. So relatively, we can see that trajectory is shorter in time compared to previous case.

For Figure c, we reduce the time interval and also constraint the backward motion by reducing the control bounds in negative/backwards side. This modification allows us to search in finer control space leading to the shortest trajectory. But it increases the overhead as we can see the no of nodes are very high compared to other cases.

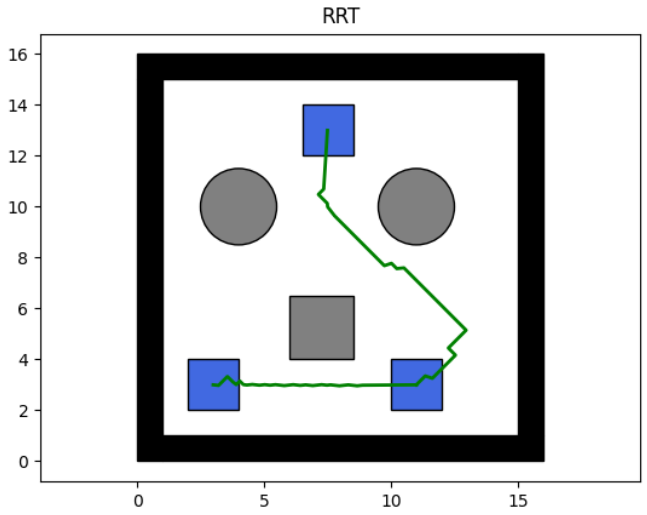


Fig. 1. Multi-query Kinodynamic RRT

This figure shows a sample case where we have multi-goal target. Starts from bottom-left → bottom-right → top. constraints were u = [-0.7, 0.7], T = [0,0.5]. Time to target 1 - 7.373 with 24 nodes and time to target 2 from target 1 - 9.628 with 35 nodes.

From this we can conclude, that this algorithm is highly sensitive to parameter selection and discretization of time. This also poses a scalability issue.

VII. Platform & Evaluation

To design and evaluate this project we majorly use Python along side Matplotlib to visualize the results. Code for this project was adopted from class assignments and `https://github.com/zhm-real/PathPlanning/tree/master`.

VIII. Contributions of Team Members

- Jeel Chatrola - Literature Review and Design
- Dev Soni - Implementation
- Combined Effort - Evaluation and Documentation

IX. Conclusions and Future Work

We proposed a extension to sampling-based Kinodynamic RRT, which can tries to converge the Path Extracted from Kinodynamic-RRT. There are limitations to the convergence we can bring from modify a existing path due to rewire algorithm's inability to modify the control input because it can lead to not reaching the path. In this project we learned that using a inefficient algorithm such as Kinodynamic RRT is limiting as even with higher number of iterations the path doesn't improve like a Asymptotically optimal planner.

Next, we will investigate the more complex Kinodynamic Planning Algorithm such as KPIECE[4] and Sysclop[5] which can bring scalability in cases of higher-dimension systems. We want to include Qualitative and Quantitative information from the STL predicates to perform a informed search using a State-of-the-art algorithm.

Acknowledgment

We would like to thank **Prof. Constantinos Chamzas** and **TA Zhuoyun Zhong** for his valuable guidance throughout the coursework and this project.

References

[1] J. Karlsson, F. S. Barbosa, and J. Tumova, "Sampling-based motion planning with temporal logic missions and spatial preferences∗∗this work was supported by the eu h2020 research and innovation programme under ga no. 731869 (co4robots), and by the swedish research council (vr).," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15537–15543, 2020. 21st IFAC World Congress.

[2] A. Linard, I. Torre, E. Bartoli, A. Sleat, I. Leite, and J. Tumova, "Real-time rrt* with signal temporal logic preferences," in *2023 IEEE/RSJ international conference on intelligent robots and systems (IROS) :*, 2023. QC 20231122.

[3] C.-I. Vasile, V. Raman, and S. Karaman, "Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3840–3847, 2017.

[4] I. A. Şucan and L. E. Kavraki, *Kinodynamic Motion Planning by Interior-Exterior Cell Exploration*, pp. 449–464. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[5] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 469–482, 2010.