Generalized Motion Planning for Dynamical Systems

***Abstract*—We propose a method for finding probabilistically optimal control trajectories for dynamical systems with differentiable dynamics and differentiable cost function. The method consists of the RRT* sample-based motion planning algorithm with a heuristic motivated by linear control system theory.**

I. Introduction

Recently, RRT, a sampling-based, probablistically complete motion planning algorithm was extended to RRT* which provides probabilistic optimality. It is straightforward to apply RRT* to kinematic motion planning problems and the this is typically the state-of-the-art solution. For problems with differential constraints the application is not so straightforward – some of the primitives require domain-specific design. We present a method that works well across many domains with differential constraints. Contribution also includes code

II. Background

A. RRT*

The RRT* algorithm is a sample-based motion planner to finds probabalistically optimal solutions. It requires the following algorithmic primitives:

Steer: $(\text{State}\, s_{\text{from}}, \text{State}\, s_{\text{to}}) \mapsto$ Action a where a is an action that can be applied in s_{from} in order to get to s_{to}

Distance: $(\text{State}\, s_{\text{from}}, \text{State}\, s_{\text{to}}) \mapsto$ Cost c where c is the cost of getting from s_{from} to s_{to} without obstacles

CollisionFree: $(\text{State}\, s_{\text{from}}, \text{State}\, s_{\text{to}}) \mapsto$ StateTraj $\langle s_1, \cdots, s_T \rangle$

B. Modifications to Primitives

Modification

Steer: $(\text{State}\, s_{\text{from}}, \text{State}\, s_{\text{to}}) \mapsto$ Action a where a is an action that can be applied in s_{from} in order to get to s_{to}

Distance: $(\text{State}\, s_{\text{from}}, \text{State}\, s_{\text{to}}) \mapsto$ Cost c where c is the cost of getting from s_{from} to s_{to} without obstacles

CollisionFree: $(\text{State}\, s_{\text{from}}, \text{Action}\, a) \mapsto$ StateTraj $\langle s_1, \cdots, s_T \rangle$, ActionTraj $\langle a_0, \cdots, a_{T-1} \rangle$ where $\langle s_1, \cdots, s_T \rangle$ is a trajectory of states arising from taking a from s_{from} until hitting an obstacle or completing the action and $\langle a_0, \cdots, a_{T-1} \rangle$ is a decomposition of a that corresponds to the state trajectory.

Cost: $(\text{State}\, s_{\text{from}}, \text{Action}\, a) \mapsto$ Cost c

Algorithm 1: $\texttt{ALG}^*((V,E),N)$

for $i = 1, \ldots, N$ **do**
- $x_{\text{rand}} \leftarrow \texttt{Sample}$;
- $(x_{\text{nearest}}, c_{\text{nearest}}) \leftarrow \texttt{Nearest}(V, x_{\text{rand}})$;
- **if** $c_{\text{nearest}} \neq \infty$ **then**
  - $a_{\text{nearest}} \leftarrow \texttt{GetAction}(x_{\text{nearest}}, x_{\text{rand}})$;
  - $(\sigma, x_{\text{new}}) \leftarrow \texttt{CollisionFree}(x_{\text{nearest}}, a_{\text{nearest}})$;
  - **if** $\sigma \neq \emptyset$ **then**
    - $X_{\text{near}} \leftarrow \texttt{Near}(V, x_{\text{new}})$;
    - $x_{\text{min}}, u_{\text{min}} \leftarrow \texttt{ChooseParent}(X_{\text{near}}, x_{\text{new}})$;
    - $x_{\text{new}} \leftarrow \texttt{SimForward}(x_{\text{min}}, u_{\text{min}})$;
    - $X \leftarrow X \cup \{x_{\text{new}}\}$;
    - $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    - $(V,E) \leftarrow \texttt{Rewire}((V,E), X_{\text{near}}, x_{\text{new}})$;

return $G = (V,E)$;

Algorithm 2: $\texttt{ChooseParent}(X_{\text{near}}, x_{\text{new}})$

minCost $\leftarrow \infty$; $x_{\min} \leftarrow$ NULL; $\sigma_{\min} \leftarrow$ NULL;
for $x_{\text{near}} \in X_{\text{near}}$ **do**
    $a \leftarrow \texttt{GetAction}(x_{\text{near}}, x_{\text{new}})$;
    $(\sigma, x') \leftarrow \texttt{CollisionFree}(x_{\text{near}}, a)$;
    if $\texttt{Cost}(x_{\text{near}}) + \texttt{Cost}(\sigma) <$ minCost **then**
        minCost $\leftarrow \texttt{Cost}(x_{\text{near}}) + \texttt{Cost}(\sigma)$;
        $x_{\min} \leftarrow x_{\text{near}}$; $\sigma_{\min} \leftarrow \sigma$;
return $(x_{\min}, \sigma_{\min})$;

Algorithm 3: $\texttt{Rewire}(\,(V,E), X_{\text{near}}, x_{\text{new}}\,)$

for $x_{\text{near}} \in X_{\text{near}}$ **do**
    $\sigma \leftarrow \texttt{LQRSteer}(x_{\text{new}}, x_{\text{near}})$;
    if $\texttt{Cost}(x_{\text{new}}) + \texttt{Cost}(\sigma) < \texttt{Cost}(x_{\text{near}})$ **then**
        if $\texttt{CollisionFree}(\sigma)$ **then**
            $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
            $E \leftarrow E \setminus \{x_{\text{parent}}, x_{\text{near}}\}$;
            $E \leftarrow E \cup \{x_{\text{new}}, x_{\text{near}}\}$;
return (V,E);

The changes to the primitives do not logically change the RRT* algorithm, but instead make it easier to plug-in the contributed components.

C. *Math Tools*

Linear Quadratic Regulator (LQR) is an optimal control technique which determines the policy (mapping from state to action) for a linear dynamical system which minimizes a quadratic functional on the state and action.

LQR is used to solve the following problem: Find $\langle u_0, \cdots u_{T-1} \rangle$ such that

$$x_{k+1} = Ax_k + Bu_k$$

minimizing

$$J(x,u) = \sum_{k=0}^{T-1} \left(x_k^T Q x_k + u_k^T R u_k \right) + x_T^T Q_T x_T$$

given A, B, Q, R, x_0, x_T, and T.

We can enforce that LQR policies obey the final-value constraint on x_T by placing very high cost (practically infinite) on the Q_T fed to the finite-horizon LQR solver. The textbook LQR problem statement requires that the quadratic bowl $x^T Q x$ be centered at $x = 0$. A shift of coordinates can change the center of all bowls, which is not what we want.

We can enrich the class of LQR problems to have arbitrary second-order penalty (along with first-order dynamics, which will be essential later) by considering the follwing transformation:

$$\underbrace{\begin{bmatrix} x_{k+1} \\ 1 \end{bmatrix}}_{\hat{x}_{k+1}} = \underbrace{\begin{pmatrix} A & c - R^{-1} r \\ 0 & 1 \end{pmatrix}}_{\hat{A}} \underbrace{\begin{bmatrix} x_k \\ 1 \end{bmatrix}}_{\hat{x}_k} + \underbrace{\begin{pmatrix} B \\ 0 \end{pmatrix}}_{\hat{B}} \hat{u}_k$$

$$J(x,u) = \sum_{k=0}^{T-1} \hat{x}_k^T \hat{Q} \hat{x}_k + \hat{u}_k^T R \hat{u}_k + \hat{x}_T^T \hat{Q}_T \hat{x}_T$$

with $\hat{u}_k = u_k + R^{-1} r$ and $\hat{Q} = \begin{pmatrix} Q & q \\ q^T & d \end{pmatrix}$.

Here we augment the state vector with an additional dimension whose value is constrained by the dynamics to remain constant. As long as this additional dimension is unity in x_0 , then it will be unity along all trajectories.

In addition to allowing first-order dynamics (increasing the class of systems from linear to affine), this transformation also allows an arbitrary second-order function on x and u. For example, say we want the following bowl:

$$(x - x_0)^T Q (x - x_0)$$

Then choose $\hat{Q} = \begin{pmatrix} Q & Q x_0 \\ x_0^T Q & x_0^T Q x_0 \end{pmatrix}$.

III. Application to a Constrainted Linear System with Quadratic Cost

Necessary since in general, not possible or practical for steer function to be exact.

Logically equivalent to common formulation of RRT*, but easier to plug in the primitives.

Algorithm detailing the different structure here

For this class of systems, the LQR method is the standard way to find optimal solution. LQR, though, cannot handle constaints on the state nor on actuation.

We can use LQR in the distance metric calculation.

A. Augmenting State with Time

Augmenting time in the RRT* state space allows us to set explicit time goals and allows us to apply the LQR heuristic. Without this, it's not clear what the cost between two states should be. Penalizing time is not correct. Lowest cost over all possible time horizons not correct. These are crucial points.

Many scenarios include time in the state anyway. For example: time-varying dynamics or obstacles and constraints.

Introduce notation especially with time dimension. For LQR math to look good, need easy notational access to state-space component and time component of the RRT state vector.

Let $\langle x,k \rangle \in \mathbb{R}^n \times \mathbb{Z}_{0+}$ be the space of the RRT state.

1) Primitives: Algorithm for all primitives

$Steer(s_1,s_2)$ $T = k(s_2) - k(s_1)$ LQR around $x(s_1)$ with goal $x(s_2)$

Explain what happens if $T \leq 0$.

2) Optimality Proof Sketch:

IV. Application to General Dynamical Systems

Consider a discrete-time dynamical system in the form

$$x_{k+1} = f(x_k, u_k) \tag{1}$$

with additive cost function

$$J(\mathbf{u},\mathbf{x}) = \sum_{k=0}^{T} g(x_k, u_k) \tag{2}$$

and starting point x_0. The state vector, x, is n-dimensional and the control vector, u, is m-dimensional. We aim to find a sequence $\mathbf{u} = \{u_0, \cdots, u_T\}$ which induces a trajectory $\mathbf{x} = \{x_1, \cdots, x_T\}$ satisfying the dynamics (1) such that C is minimized according to (2).

The real cost of moving from point x' to x'' is

$$C(x,x') = \min_{\mathbf{u}} J(\mathbf{u},\mathbf{x}) \quad \text{subject to (1), } x_0 = x,\ x_T = x'$$

Note that the minimization happens over control sequences $\mathbf{u}$ of a fixed time lengths, according to

We approximate $C(x',x'')$ by taking a first-order approximation of the dynamics and a second-order approximation of the cost and applying LQR control. In general, the approximated dynamics and cost are of the following form

$$x_{k+1} \approx Ax_k + Bu_k + c \tag{3}$$

$$J(\mathbf{u},\mathbf{x}) \approx \sum_{k=0}^{T} x_k^T Q x_k + u_k^T R u_k + 2q^T x_k + 2r^T u_k + d \tag{4}$$

A and Q are $n \times n$, B is $m \times n$, R is $n \times n$. c and q are $n \times 1$, r is $m \times 1$ and d is a scalar.

$$A = \left.\frac{\partial f}{\partial x}\right|_{x^*,u^*}$$
$$B = \left.\frac{\partial f}{\partial u}\right|_{x^*,u^*}$$
$$c = -Ax^* - Bu^* + f(x^*,u^*)$$

x^*, u^* is the point about which the linearization is performed. Typically u^* is taken to be $\mathbf{0}$ and $x^* = x'$

Equations 3 and 4 are the truncated Taylor expansions of f and g. The dynamics f must be once-differentiable and addition cost g must be twice-differentiable.

A. Reduction to the Previous Problem

It is possible to transform the problem specified with 3 and 4 into LQR form (where there is only an A, B, Q, R matrix) using the following:

$$\underbrace{\begin{bmatrix} x_{k+1} \\ 1 \end{bmatrix}}_{\hat{x}_{k+1}} = \underbrace{\begin{pmatrix} A & c - R^{-1} r \\ 0 & 1 \end{pmatrix}}_{\hat{A}} \underbrace{\begin{bmatrix} x_k \\ 1 \end{bmatrix}}_{\hat{x}_k} + \underbrace{\begin{pmatrix} B \\ 0 \end{pmatrix}}_{\hat{B}} \hat{u}_k$$

$$C(\mathbf{u}, \mathbf{x}) = \sum_{k=0}^{T} \hat{x}_k^T \hat{Q} \hat{x}_k + \hat{u}_k^T R \hat{u}_k$$

with $\hat{u}_k = u_k + R^{-1} r$ and $\hat{Q} = \begin{pmatrix} Q & q \\ q^T & d \end{pmatrix}$.

The $\hat{A}$, $\hat{B}$, $\hat{Q}$, and R matrices specify a linear dynamical system with quadratic costs to which an optimal solution can be found with LQR.

B. Nuances and Subtleties

1) Non-exact steering: rewiring and propagating dynamics

2) Uncontrollable Dynamics: The linearized system may be uncontrollable – the A and B matrices are such that it's not possible to control all the modes of the system. This is the case, for example, for a cart with two inverted pendulums of the same length linearized about the upward-pointing fixed point. The control input to the system affects both linearized pendulums in the same way, so it's not possible to independently stabilize them. For the infinite-horizon LQR control problem, there is no solution. For the finite-horizon problem, there is a solution, though it might not be possible to go to any arbitrary location. If the system linearized at x' cannot reach x'', then $C(x', x'')$ needs to be defined in another way.

is Therefore using the LQR cost metric cannot approximate the cost

3) Indefinite Cost:

4) Actuation Constraints: The LQR framework does not permit actuation constraints.

5) Asymmetric Cost:

V. Related Work

vdB

VI. Results

A. Linear Domain

spaceship no orientation

B. Non-linear Domain

spaceship orientation

C. Results

Quick mention of performance (or not). Picture of tree, cost over iteration

VII. Discussion and Conclusion and Future Work

Code available. Spatial Data structure future

VIII. Acknowledgments

IX. References