KinoFMT*

An Algorithm for Real-time Motion Planning

Based on the works of M. Pavone and R. Allen

Luc Larocque

Department of Applied Mathematics
University of Waterloo

January 9, 2018

Outline

Optimal Kinodynamic Planning Problem:

$$
\begin{aligned}
\text{Find control: } & u(t) & \\
\text{that minimizes: } & \mathcal{J}[x(t), u(t), t_{final}] & \\
\text{subject to: } & u(t) \in \mathcal{U} & , \forall t \in [t_{init}, t_{final}] \\
& x(t) \in \mathcal{X}_{free} & , \forall t \in [t_{init}, t_{final}] \\
& f_l \leq f[\dot{x}(t), x(t), u(t), t] \leq f_u & , \forall t \in [t_{init}, t_{final}] \\
& x(t_{final}) \in \mathcal{X}_{goal} &
\end{aligned}
$$

Note:

- t_{final} is **free**
- We focus on problems where $\mathcal{X}_{free}$ is not explicitly represented.

Outline

Approximate Dynamics

There are no known analytical solutions to the minimum-time optimal control problem under the quadrotor's nonlinear dynamics.

Approximate Dynamics

There are no known analytical solutions to the minimum-time optimal control problem under the quadrotor's nonlinear dynamics.

Use approximator-corrector structure to simplify computations. Approximate using double-integrator which can be solved analytically for the **unobstructed** minimal-time control problem:

$$\dot{x}(t) = Ax + Bu + c$$

$$A = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ I \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ g \end{bmatrix}, \quad x = \begin{bmatrix} \xi_B \\ \dot{\xi}_B \end{bmatrix} \in \mathbb{R}^6, \quad u = \ddot{\xi}_B \in \mathbb{R}^3$$

where ξ_B is the position of the body frame.

Sampling (Offline)

"The key idea behind sampling-based algorithms is to **avoid the explicit construction of the configuration space** and instead conduct a search that probes the configuration space with a sampling scheme."

Offline phase:

- Randomly draw N_s samples from state space.

Sampling (Offline)

"The key idea behind sampling-based algorithms is to **avoid the explicit construction of the configuration space** and instead conduct a search that probes the configuration space with a sampling scheme."

Offline phase:

- Randomly draw N_s samples from state space.
- From N_s samples, randomly draw N_{pair} pairs of states, with replacement, $N_{pair} \leq N_s(N_s - 1)$.
  Store one state from each of these pairs in A, the other in B.

Sampling (Offline)

"The key idea behind sampling-based algorithms is to **avoid the explicit construction of the configuration space** and instead conduct a search that probes the configuration space with a sampling scheme."

Offline phase:

- Randomly draw N_s samples from state space.
- From N_s samples, randomly draw N_{pair} pairs of states, with replacement, $N_{pair} \leq N_s(N_s - 1)$.
  Store one state from each of these pairs in A, the other in B.
- Solve the OBVP determined by each pair in A, B;
  Store solutions in table titled COST.

Sampling (Offline)

"The key idea behind sampling-based algorithms is to **avoid the explicit construction of the configuration space** and instead conduct a search that probes the configuration space with a sampling scheme."

Offline phase:

- Randomly draw N_s samples from state space.
- From N_s samples, randomly draw N_{pair} pairs of states, with replacement, $N_{pair} \leq N_s(N_s - 1)$.
  Store one state from each of these pairs in A, the other in B.
- Solve the OBVP determined by each pair in A, B;
  Store solutions in table titled COST.
- A SVM classifier is trained using the look-up table COST.
  The SVM is used to approximate cost-limited reachable sets.

Planning (Online)

Obstacle data is obtained from the collision detection module.
Start state x_{init} and goal region $\mathcal{X}_{goal}$ are now specified.

Online phase:

- Randomly draw N_{goal} states from $\mathcal{X}_{goal}$.

Planning (Online)

Obstacle data is obtained from the collision detection module. Start state x_{init} and goal region $\mathcal{X}_{goal}$ are now specified.

Online phase:

- Randomly draw N_{goal} states from $\mathcal{X}_{goal}$.
- Use the SVM to approximate outgoing nbhd of x_{init} and incoming nbhd of $\mathcal{X}_{goal}$ from pre-sampled states.

Planning (Online)

Obstacle data is obtained from the collision detection module. Start state x_{init} and goal region $\mathcal{X}_{goal}$ are now specified.

Online phase:

- Randomly draw N_{goal} states from $\mathcal{X}_{goal}$.
- Use the SVM to approximate outgoing nbhd of x_{init} and incoming nbhd of $\mathcal{X}_{goal}$ from pre-sampled states.
- Solve OBVPs from x_{init} and $\mathcal{X}_{goal}$ to their nearest neighbors (determined by SVM) and also store these solutions in $\textsc{Cost}$.

Planning (Online)

Obstacle data is obtained from the collision detection module. Start state x_{init} and goal region $\mathcal{X}_{goal}$ are now specified.

Online phase:

- Randomly draw N_{goal} states from $\mathcal{X}_{goal}$.
- Use the SVM to approximate outgoing nbhd of x_{init} and incoming nbhd of $\mathcal{X}_{goal}$ from pre-sampled states.
- Solve OBVPs from x_{init} and $\mathcal{X}_{goal}$ to their nearest neighbors (determined by SVM) and also store these solutions in $\textsc{Cost}$.
- Use kinoFMT* algorithm to determine the optimal trajectory through the set of sampled states.

Planning (Online)

Obstacle data is obtained from the collision detection module. Start state x_{init} and goal region $\mathcal{X}_{goal}$ are now specified.

Online phase:

- Randomly draw N_{goal} states from $\mathcal{X}_{goal}$.
- Use the SVM to approximate outgoing nbhd of x_{init} and incoming nbhd of $\mathcal{X}_{goal}$ from pre-sampled states.
- Solve OBVPs from x_{init} and $\mathcal{X}_{goal}$ to their nearest neighbors (determined by SVM) and also store these solutions in $\textsc{Cost}$.
- Use kinoFMT* algorithm to determine the optimal trajectory through the set of sampled states.
- The selected optimal sample states are traced with a smooth path, which the online flight controller tracks.

SVM

A *support vector machine* (SVM) is used for classification.
E.g., "Does point A satisfy property P?"

SVM

A *support vector machine* (SVM) is used for classification. E.g., "Does point A satisfy property P?"

Here, the SVM is used to determine whether or not an edge/transition between states has optimal cost below some threshold, J_{th}.

SVM

A *support vector machine* (SVM) is used for classification. E.g., "Does point A satisfy property P?"

Here, the SVM is used to determine whether or not an edge/transition between states has optimal cost below some threshold, J_{th}.

The SVM is trained using information from the COST table. Example entry:

$$((a, b), (c, d)) : (\underbrace{223.4}_{J_{opt}}, \underbrace{0.8791}_{T_{opt}})$$

SVM : Cost-limited Reachable Set Figure

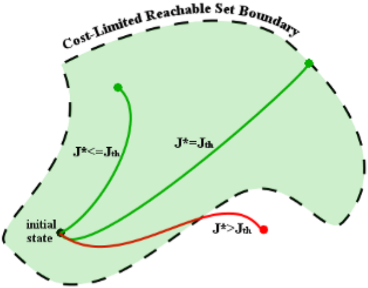

Figure 4. Conceptual representation of a cost-limited reachable set for a notional 2D dynamical system. Formally, a (forward) cost-limited reachable set is the set of states that can be reached from a given state with a cost bounded above by a given threshold (denoted as J_{th}).

Differential Flatness

The nonlinear quadrotor dynamic equations represent a *differentially flat* system.

That is, the state and control variables can be expressed in terms of position (ξ_N) and yaw (ψ_N) and their derivatives.

These are called *flat output variables*.

Differential Flatness

The nonlinear quadrotor dynamic equations represent a *differentially flat* system.

That is, the state and control variables can be expressed in terms of position (ξ_N) and yaw (ψ_N) and their derivatives.

These are called *flat output variables*.

Differential flatness is an important property that allows us the following freedom:
Any smooth trajectory (with reasonably bounded derivatives) in the space of flat outputs can be followed by the underactuated quadrotor.

Trajectory Smoothing

Goal: Find minimum-snap polynomial trajectories connecting waypoints chosen by kinoFMT*. We need one polynomial for each of the four flat output variables, and for each segment of the trajectory.

Let $P(t) = \sum_{i=0}^{N} p_i t^i$ be an Nth order polynomial. We must find the coefficients p_i that minimize

$$J_{snap} = \int_0^T P^{(4)}(t)^2 dt = \boldsymbol{p}^\top Q(T)\boldsymbol{p}$$

under the constraints

$$A\boldsymbol{p} = \boldsymbol{d}$$

where $A = \begin{bmatrix} A_0 \\ A_T \end{bmatrix}$, $\boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_T \end{bmatrix}$

Trajectory Smoothing

The constrained QP problem is numerically unstable.
Instead, we can substitute $\boldsymbol{p} = A^{-1}\boldsymbol{d}$ into J_{snap} to obtain an unconstrained QP problem:

$$\begin{aligned} J_{snap} &= \boldsymbol{p}^\top Q(T)\boldsymbol{p} \\ &= \boldsymbol{d}^\top A^{-\top} Q(T) A^{-1}\boldsymbol{d} \end{aligned}$$

rewriting A, Q, and $\boldsymbol{d}$ as block diagonal matrices, one for each segment of the trajectory.

Trajectory Smoothing

$$J_{snap} = \boldsymbol{d}^{\mathsf{T}} A^{-\mathsf{T}} Q(T) A^{-1} \boldsymbol{d}$$

For convenience, we reorder $\boldsymbol{d}$ so that fixed derivatives ($\boldsymbol{d}_{fix}$) and free derivatives ($\boldsymbol{d}_{free}$) are grouped together, which is accomplished by multiplying $\boldsymbol{d}$ by the appropriate permutation matrix, C.

$$\begin{aligned} J_{snap} &= \begin{bmatrix} \boldsymbol{d}_{fix} \\ \boldsymbol{d}_{free} \end{bmatrix}^{\mathsf{T}} C^{\mathsf{T}} A^{-\mathsf{T}} Q(T) A^{-1} C \begin{bmatrix} \boldsymbol{d}_{fix} \\ \boldsymbol{d}_{free} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{d}_{fix} \\ \boldsymbol{d}_{free} \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \boldsymbol{d}_{fix} \\ \boldsymbol{d}_{free} \end{bmatrix} \end{aligned}$$

Differentiating and setting to zero yields:

$$\boldsymbol{d}^*_{free} = -H_{22}^{-1} H_{12}^{\mathsf{T}} \boldsymbol{d}_{fix}$$

Outline

FMT* Basics

- Uses dynamic programming over a fixed set of sampled points

FMT* Basics

- Uses dynamic programming over a fixed set of sampled points
- Sample state b is considered a *neighbour* of a if the optimal cost from a to b is less than some threshold, J_{th}

FMT* Basics

- Uses dynamic programming over a fixed set of sampled points
- Sample state b is considered a *neighbour* of a if the optimal cost from a to b is less than some threshold, J_{th}
- Whenever a locally-optimal connection intersects an obstacle, that sample is lazily skipped over in the current iteration

FMT* Algorithm

Algorithm 3 Kinodynamic Fast Marching Tree Algorithm (*kino*-FMT)

$V \leftarrow V \cup \{x_{\text{init}}\} \cup \{X_{\text{goal}}\}$
$E \leftarrow \emptyset$
$W \leftarrow V \setminus \{x_{\text{init}}\}$; $H \leftarrow \{x_{\text{init}}\}$
$z \leftarrow x_{\text{init}}$
while $z \notin \mathcal{X}_{\text{goal}}$ **do**
  $N_z^{out} \leftarrow \texttt{Near}(z, V\setminus\{z\}, J_{\text{th}})$
  $X_{\text{near}} = \texttt{Intersect}(N_z^{out}, W)$
  for $x \in X_{\text{near}}$ **do**
    $N_x^{in} \leftarrow \texttt{Near}(V\setminus\{x\}, x, J_{\text{th}})$
    $Y_{\text{near}} \leftarrow \texttt{Intersect}(N_x^{in}, H)$
    $y_{\min} \leftarrow \arg\min_{y \in Y_{\text{near}}} \{\texttt{Cost}(y, T = (V, E)) + \texttt{Cost}(\overline{yx})\}$
    if $\texttt{CollisionFree}(y_{\min}, x)$ **then**
      $E \leftarrow E \cup \{(y_{\min}, x)\}$
      $H \leftarrow H \cup \{x\}$
      $W \leftarrow W \setminus \{x\}$
  $H \leftarrow H \setminus \{z\}$
  if $H = \emptyset$ **then**
    return Failure
  $z \leftarrow \arg\min_{y \in H} \{\texttt{Cost}(y, T = (V, E))\}$
return $\texttt{Path}(z, T = (V, E))$

DEMO