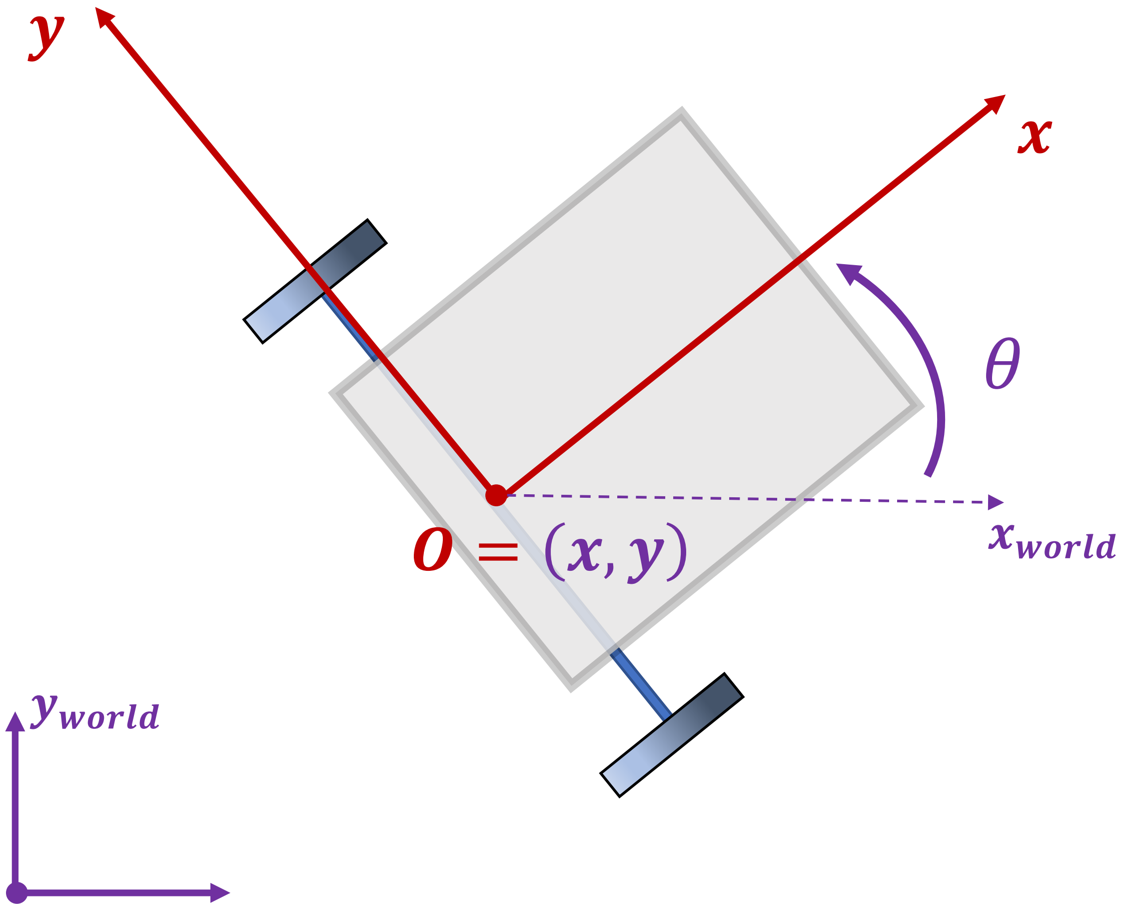
CS 3630!

Lecture 27:
RRTs and Trajectory Optimization

RRT Recap

Configuration Space

- A ***configuration*** is a complete specification of the position of every point in a robot system.
- The ***configuration space*** is the set of all configurations.
- We use q to denote a point in a configuration space Q.



Because our DDR can rotate in the plane, it is necessary to know both the position and the orientation of the body-attached frame to specify a configuration:

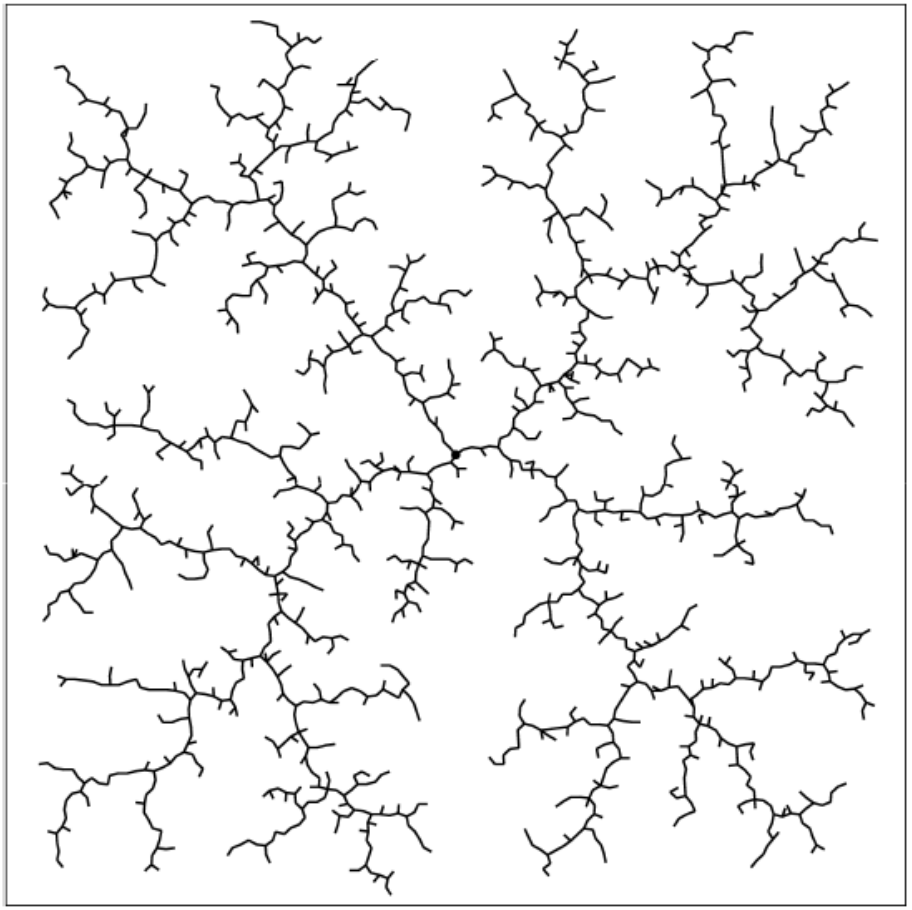
$$Q = \mathbb{R}^2 \times [0, 2\pi)$$

$$q = (x, y, \theta) \in Q$$

If we know the configuration, $q = (x, y, \theta)$, we can compute the location of any point on the robot.

Rapidly-Exploring Random Tree (RRT)

- Searches for a path from the initial configuration to the goal configuration by expanding a search tree
- For each step,
  - The algorithm samples a target configuration and expands the tree towards it.
  - The sample can either be a random configuration or the goal configuration itself, depends on the probability value defined by the user.



The Basic Idea: Iteratively expand the tree

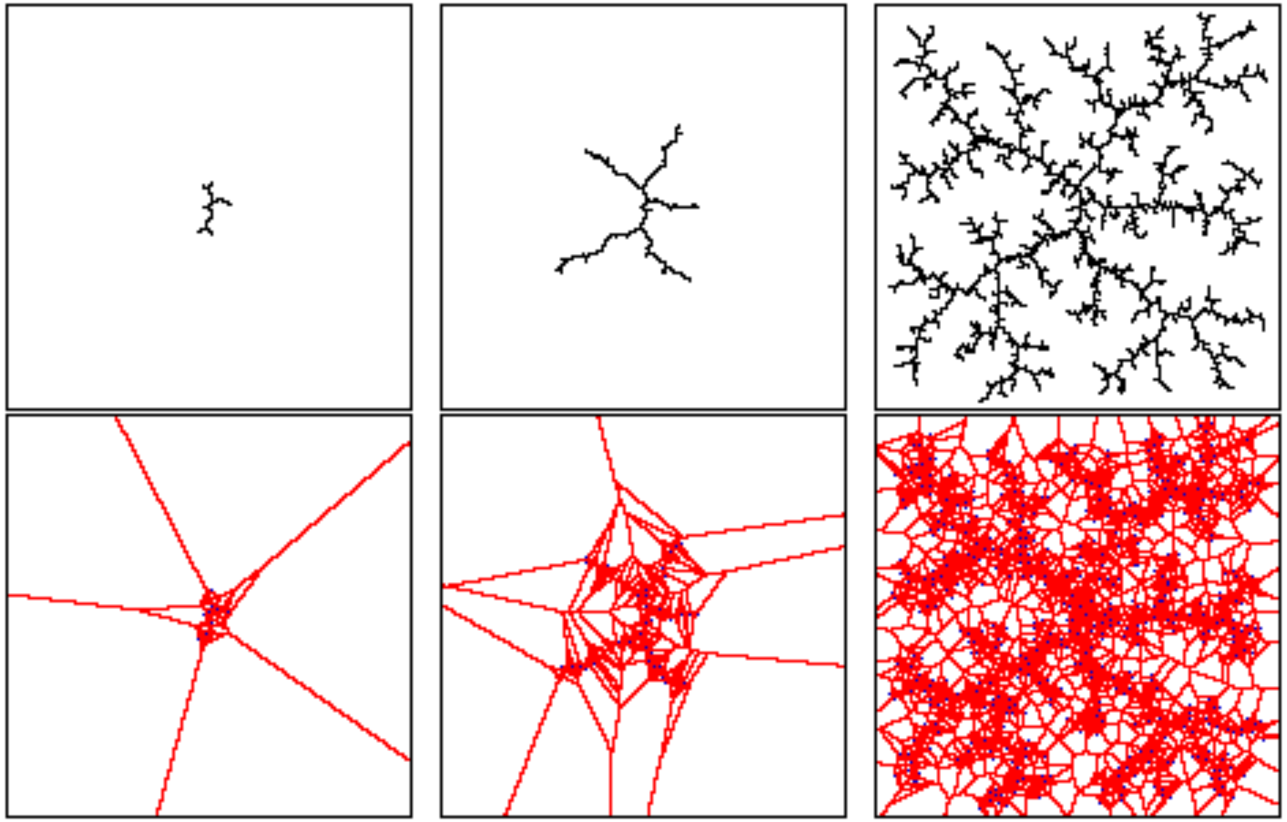
- Denote by T_k the tree at iteration k

- Randomly choose a configuration q_{rand}

- Choose $q_{near} = \arg\min_{q \in T_k} d(q, q_{rand})$
  - q_{near} is the nearest existing node in the tree to q_{rand}

- Create a new node, q_{new} by taking a small step from q_{near} toward q_{rand}

Why are RRT's rapidly exploring?



The probability of a node being selected for expansion (i.e. being a nearest neighbor to a new randomly picked point) is proportional to the area of its Voronoi region.

RRT

- Requires the following functions:
  - p = **RandomSample**()
    - Uniform random sampling of free configuration space
  - v = **Nearest**(p)
    - Given point in Cspace, find vertex on tree that is closest to that point
  - p’ = **Steer**(p, goal)
    - For a point p and a goal point, find p’ that is closer to the goal than p
  - **ObstacleFree**(p)
  - Check if a given Cspace point is in the free space

RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$

for $i = 1$ **to** N

$\quad G \leftarrow (V, E)$

$\quad x_{rand} \leftarrow RandomSample()$

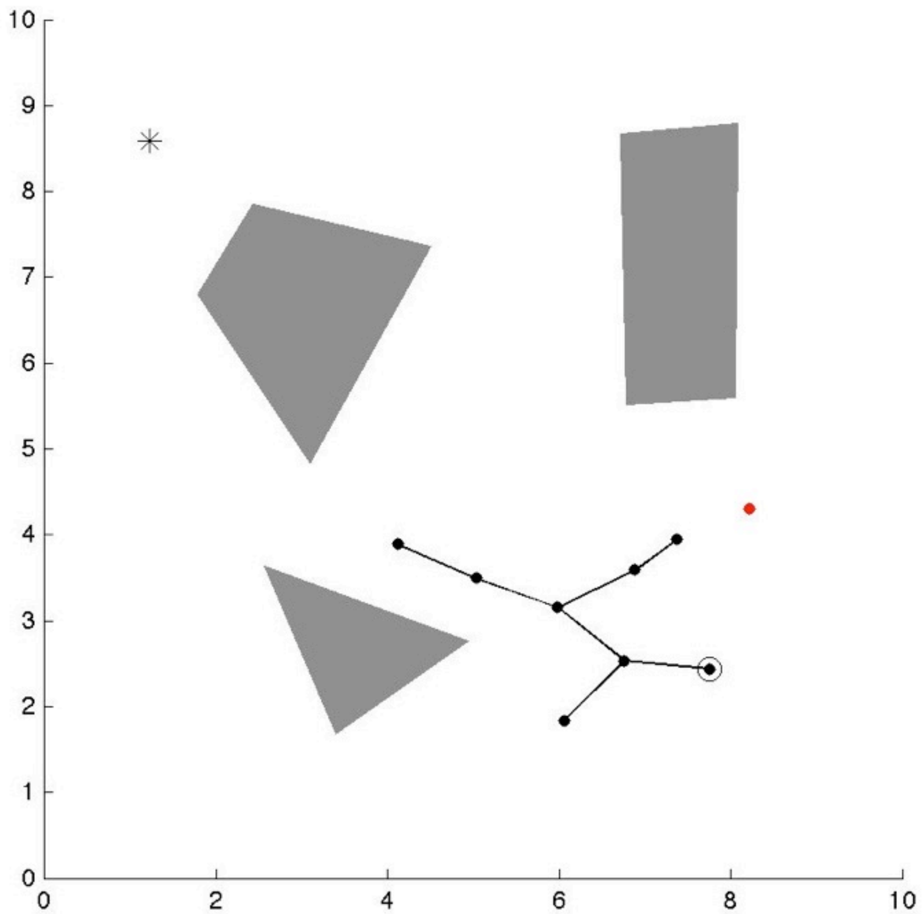
$\quad x_{nearest} \leftarrow Nearest(G, x_{rand})$

$\quad x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$

$\quad$ **if** $ObstacleFree(x_{nearest}, x_{new})$

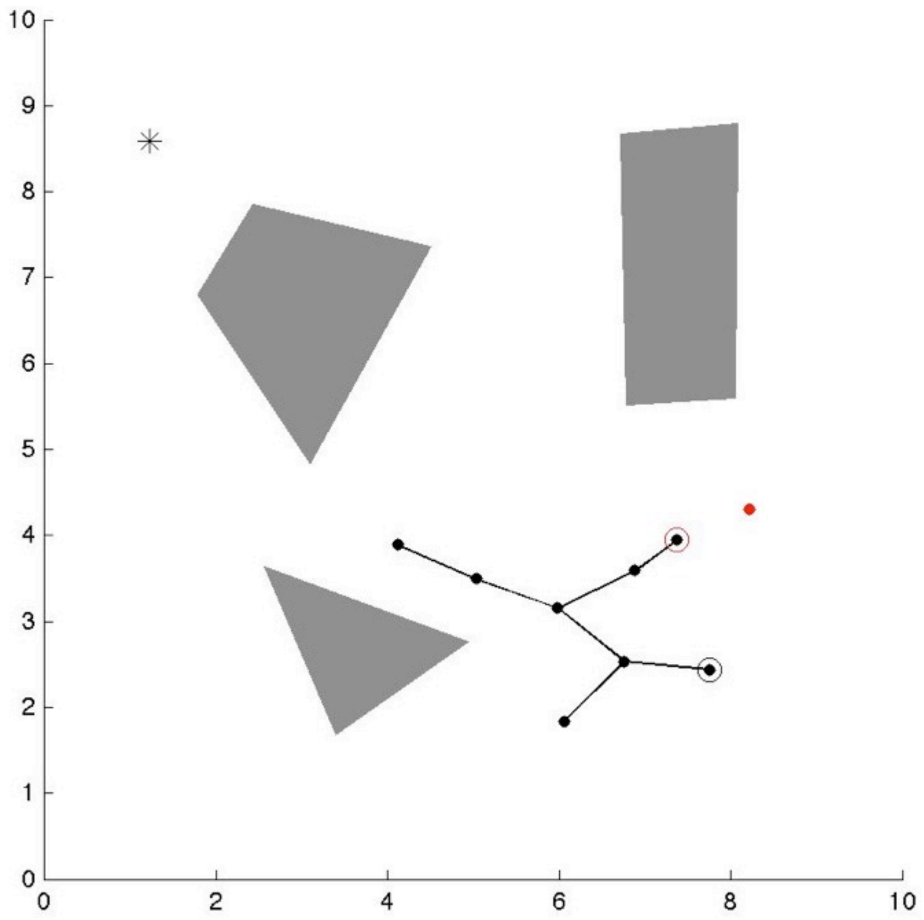
$\quad\quad V \leftarrow V \cup \{x_{new}\}$

$\quad\quad E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



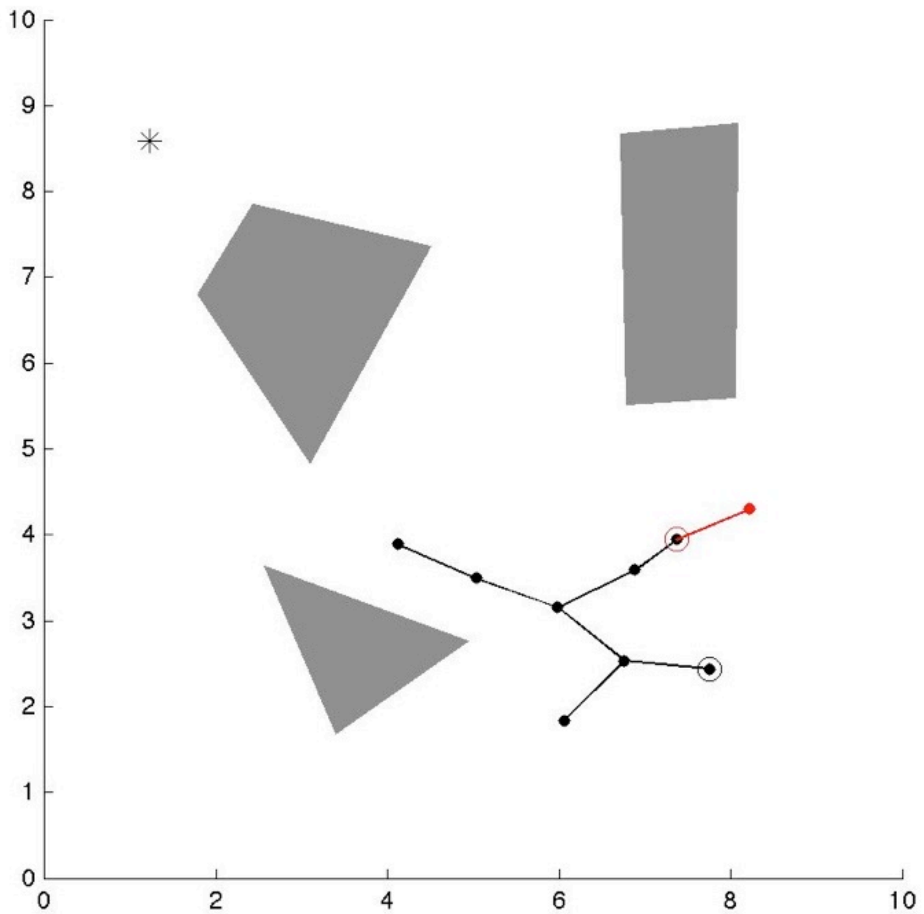
RRT

$$
\begin{array}{l}
V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset \\
\textbf{for } i = 1 \text{ to } N \\
\quad G \leftarrow (V, E) \\
\quad x_{rand} \leftarrow RandomSample() \\
\quad x_{nearest} \leftarrow Nearest(G, x_{rand}) \\
\quad x_{new} \leftarrow Steer(x_{nearest}, x_{rand}) \\
\quad \textbf{if } ObstacleFree(x_{nearest}, x_{new}) \\
\quad\quad V \leftarrow V \cup \{x_{new}\} \\
\quad\quad E \leftarrow E \cup \{(x_{nearest}, x_{new})\}
\end{array}
$$



RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $\boldsymbol{x_{new} \leftarrow Steer(x_{nearest}, x_{rand})}$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$

for $i = 1$ to N

$\quad G \leftarrow (V, E)$

$\quad x_{rand} \leftarrow RandomSample()$

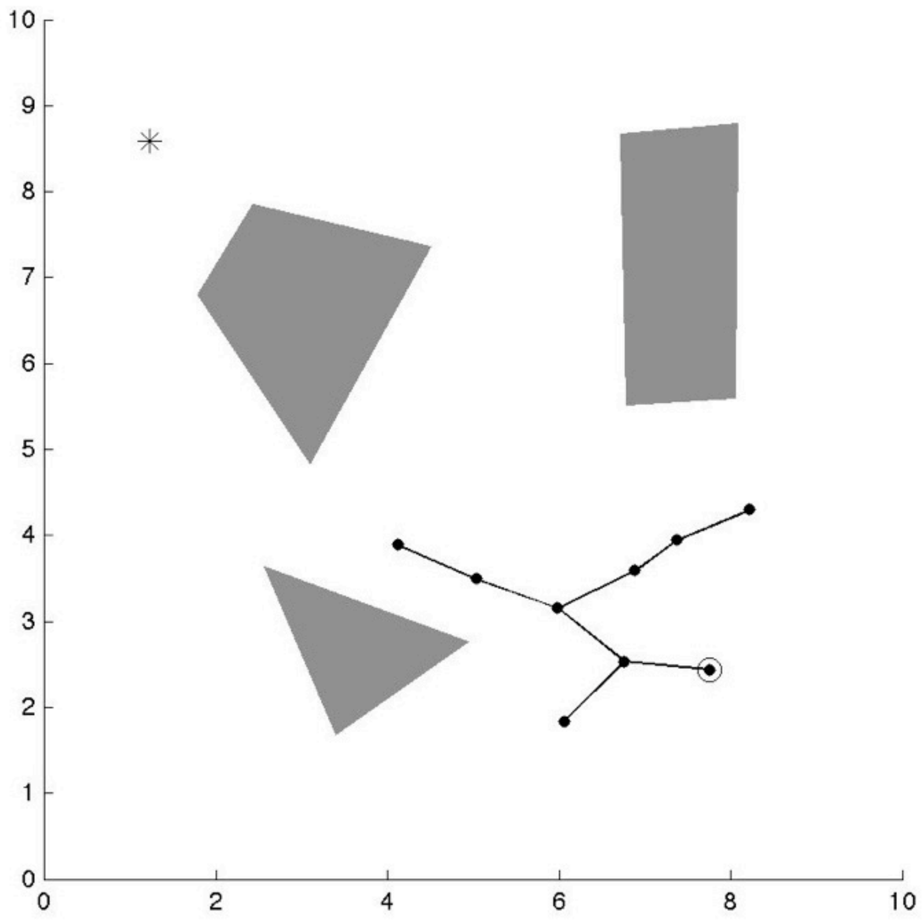
$\quad x_{nearest} \leftarrow Nearest(G, x_{rand})$

$\quad x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$

$\quad$ **if** $ObstacleFree(x_{nearest}, x_{new})$

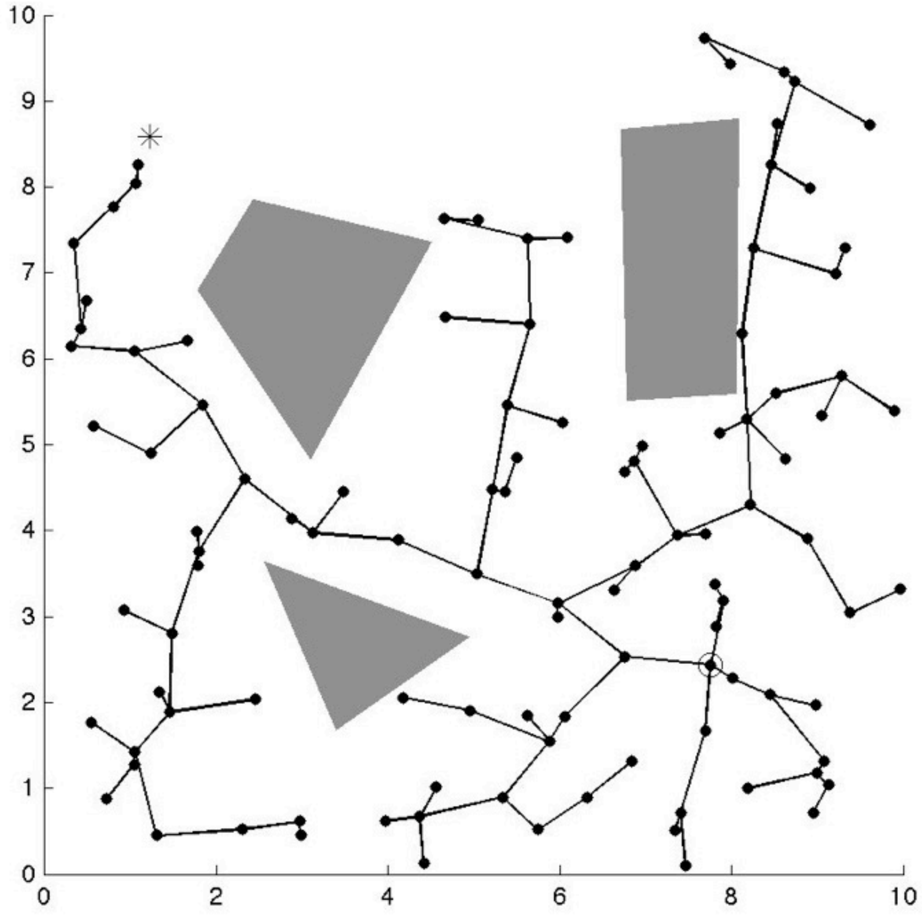
$\quad\quad V \leftarrow V \cup \{x_{new}\}$

$\quad\quad E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
 $G \leftarrow (V, E)$
 $x_{rand} \leftarrow RandomSample()$
 $x_{nearest} \leftarrow Nearest(G, x_{rand})$
 $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
 if $ObstacleFree(x_{nearest}, x_{new})$
  $V \leftarrow V \cup \{x_{new}\}$
  $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



RRT - Bias to Goal

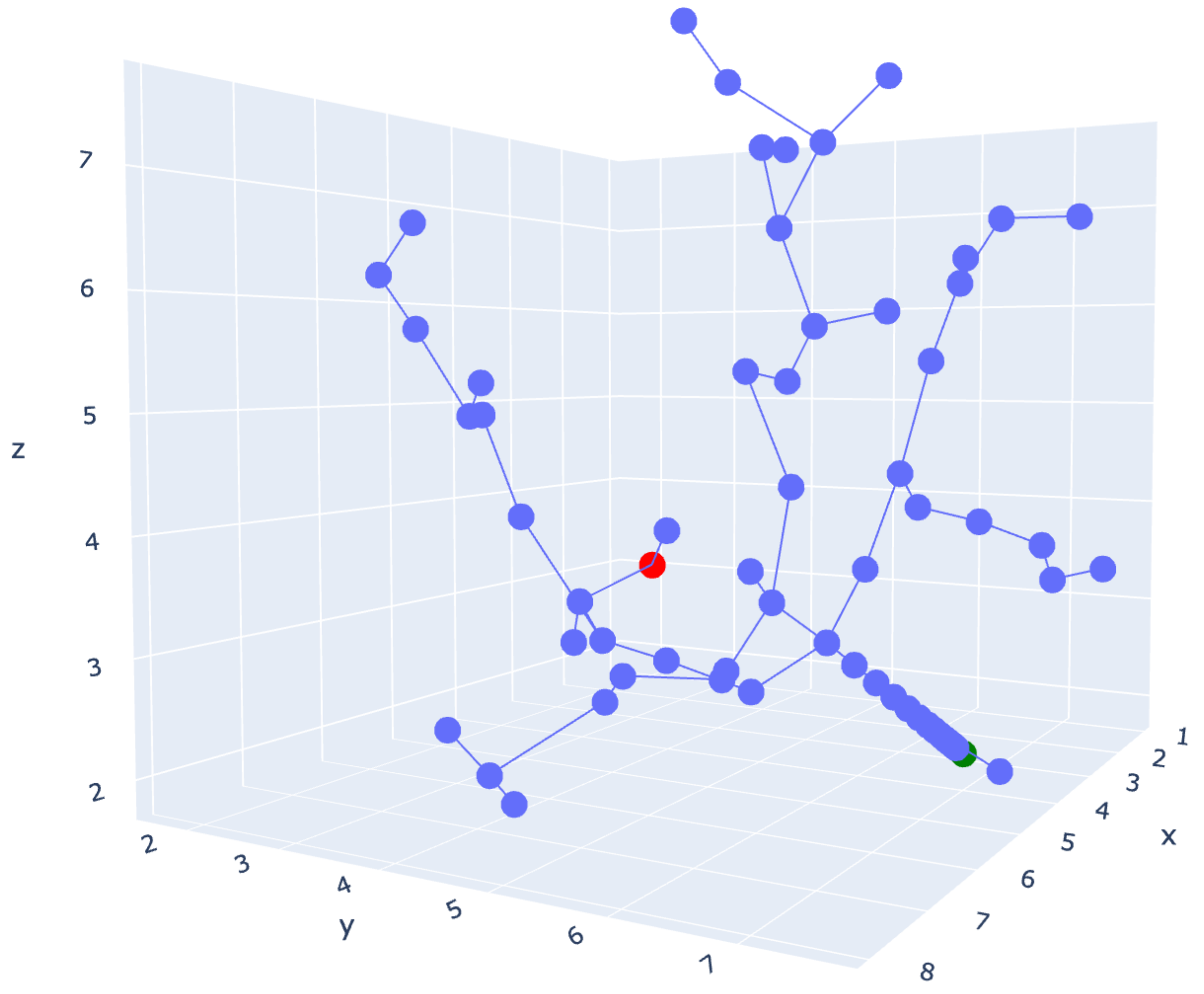
$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  with probability p
    $x_{rand} \leftarrow RandomSample()$
  otherwise
    $x_{rand} \leftarrow x_{goal}$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$

RRT for Drones

[Mardi Gras 2017 finals]

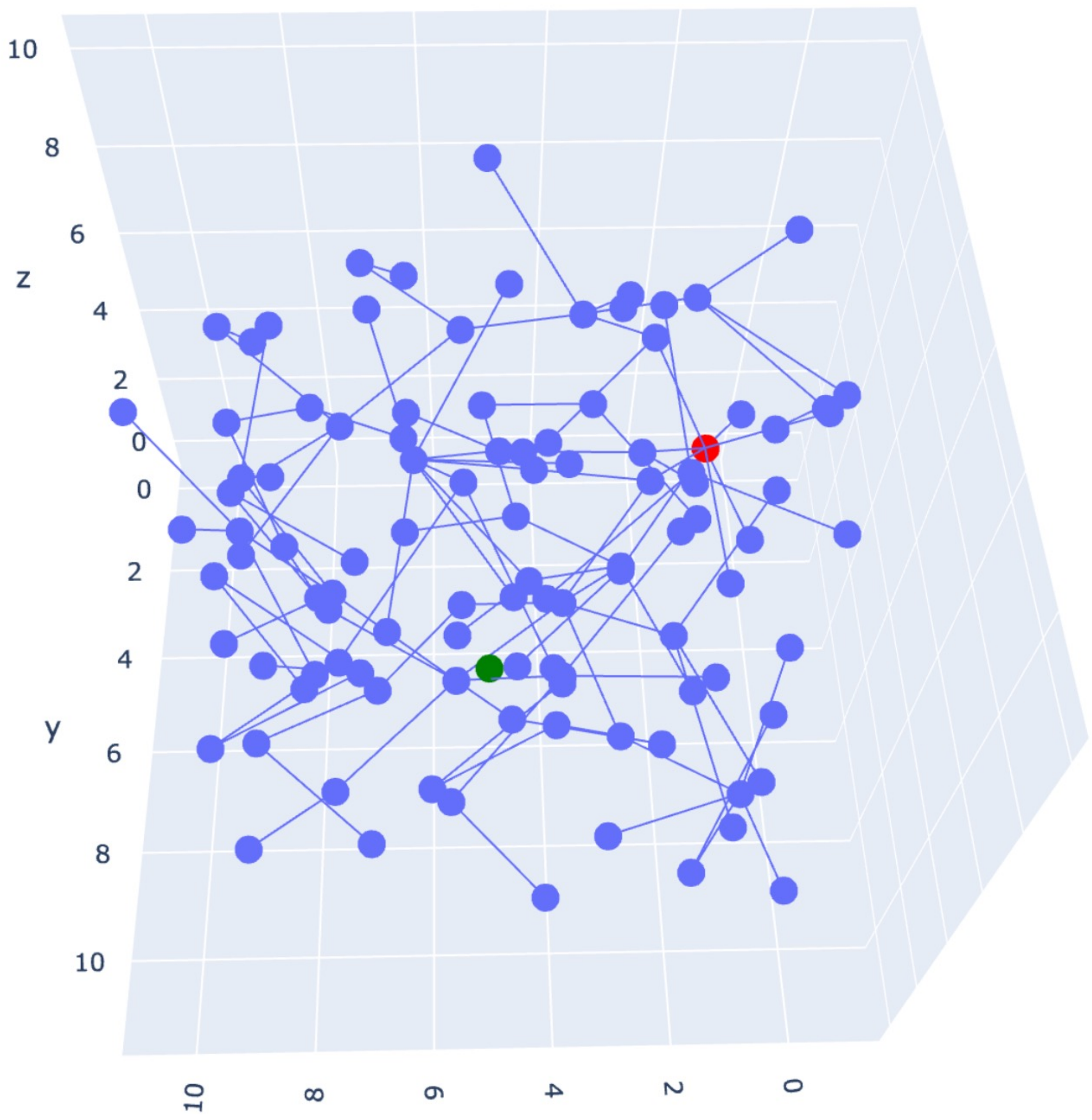
RRT for Drones in Project 6

- **RandomSample**()
    - Generate a point in 3D
- **Nearest**(p)
    - Find closest point on the evolving 3D tree
- p' = **Steer**(p, goal)
    - Steer the drone toward the target node
- **ObstacleFree**(p)
- No obstacles at first

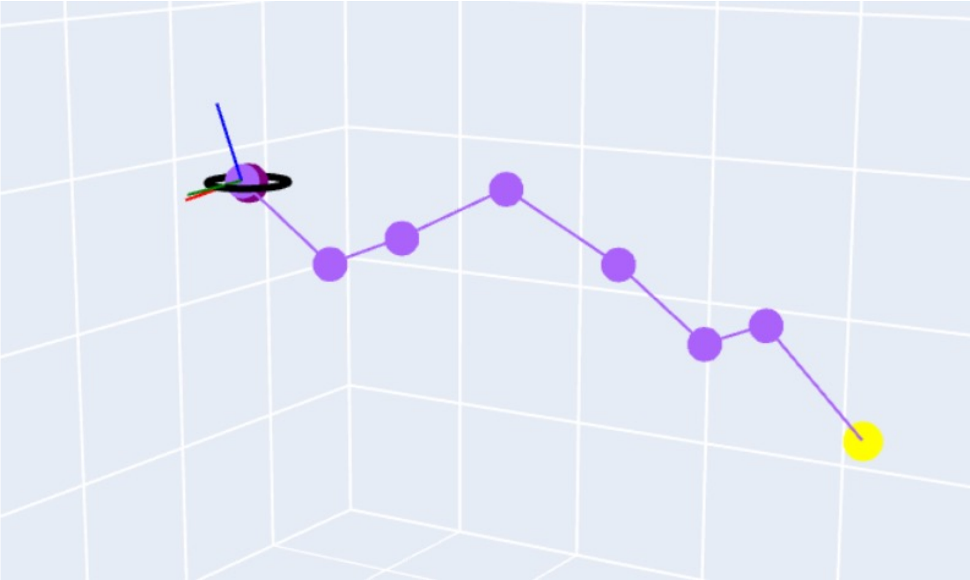


Adding Drone Dynamics

- **RandomSample**()
  - Generate a random pose
  - ENU nav frame, FLU body frame
- **Nearest**(p)
  - Find nearest pose on the tree
- p’ = **Steer**(p, goal)
  - fly the drone for a small duration in the direction of the target at the terminal velocity with maximum thrust of 20N.
- **ObstacleFree**(p)
- No obstacles at first

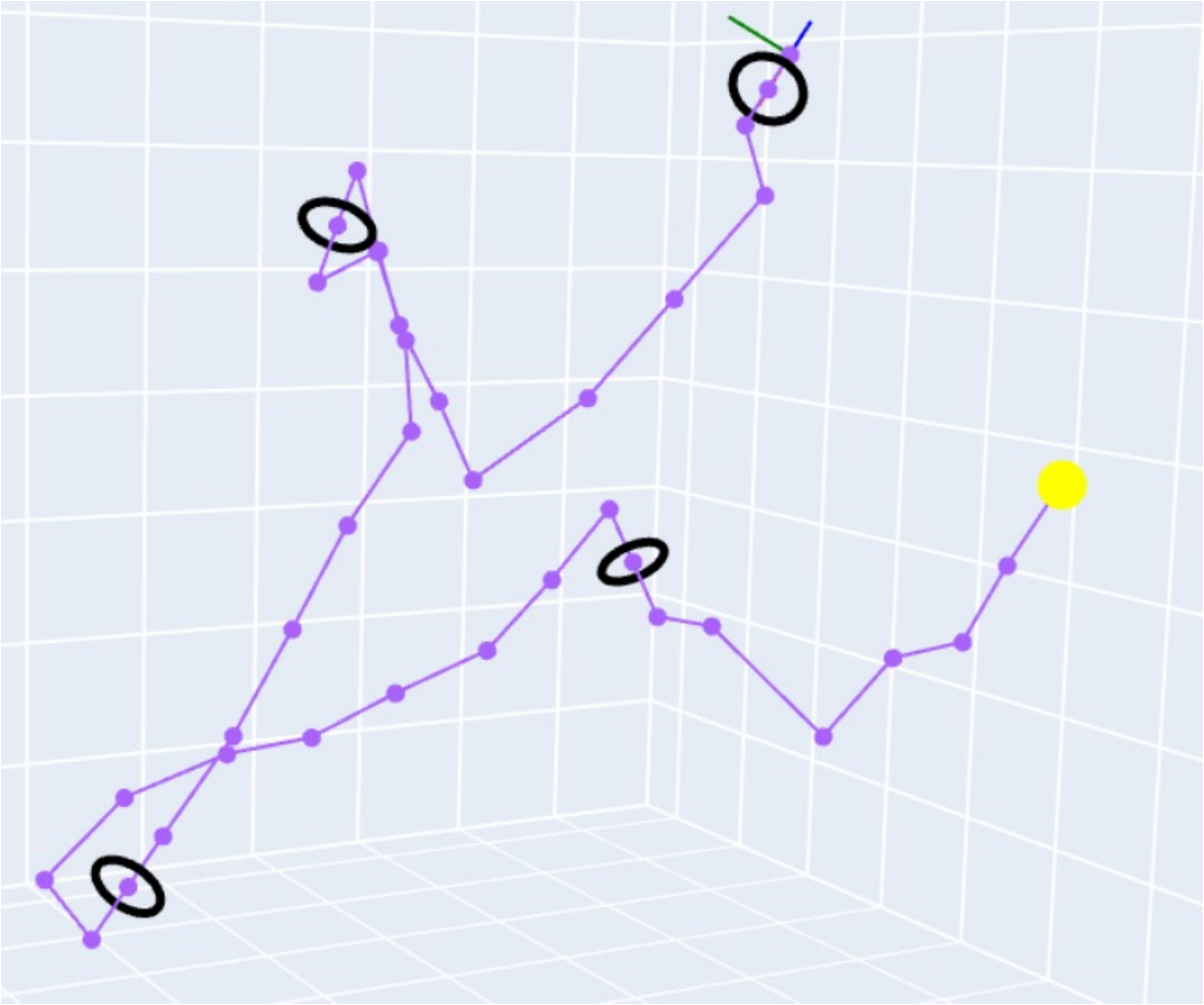


More realism



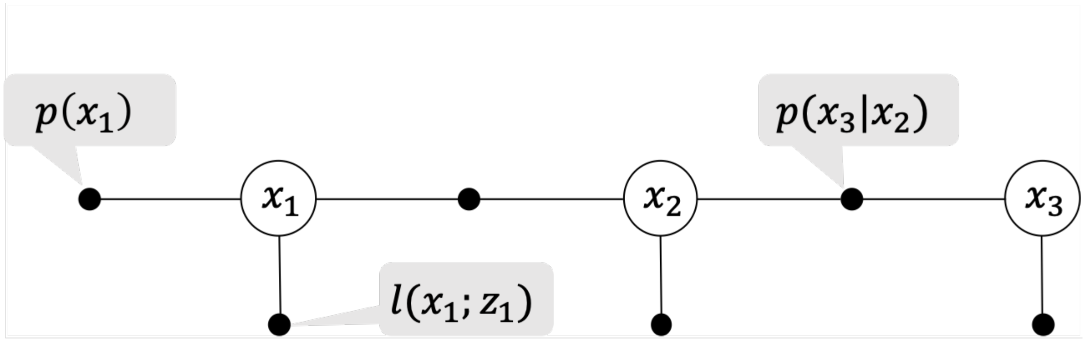
- Before we assumed we can apply thrust in any direction we want
- Realism 1: add effect of gravity!
  - Drone no longer flies where we want!
- Realism 2: no instantaneous attitude changes!
  - Allow only 10 degree changes in yaw, pitch, roll
  - Allow choosing thrust values
- New Steer function:
  - Check all 108 different combinations
  - Return result that gets us closest to the target!

Putting it all together: drone racing!

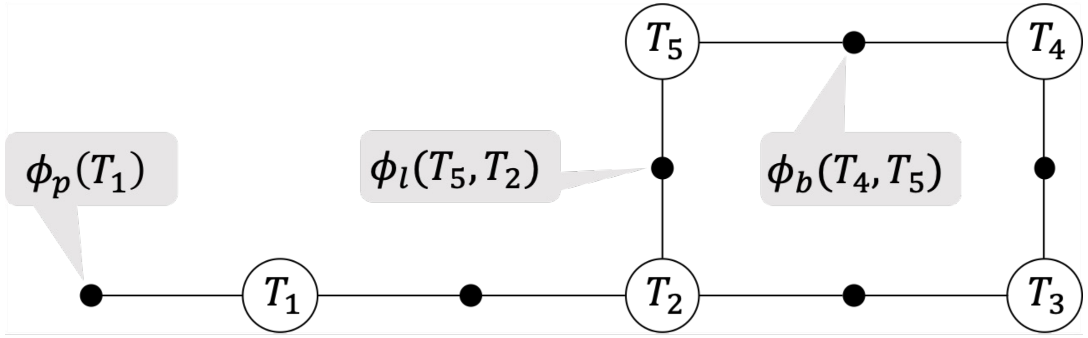


Trajectory Optimization

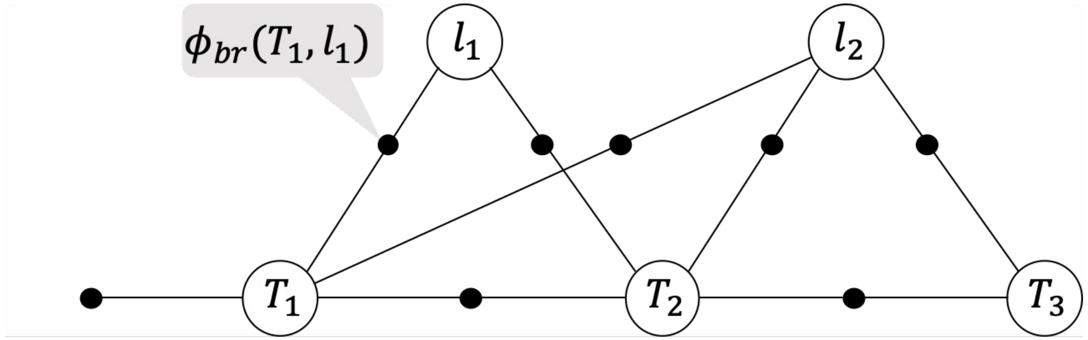
Factor graphs model both *perception* and *action*, from SLAM and 3D mapping to optimal control



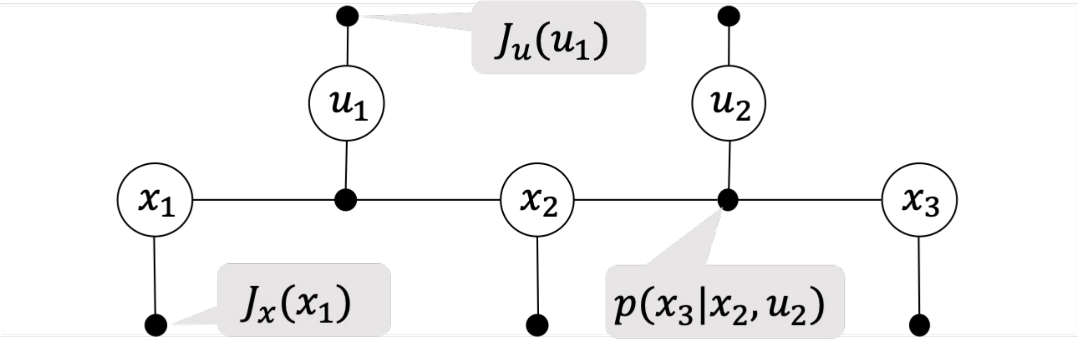
Localization



Pose SLAM



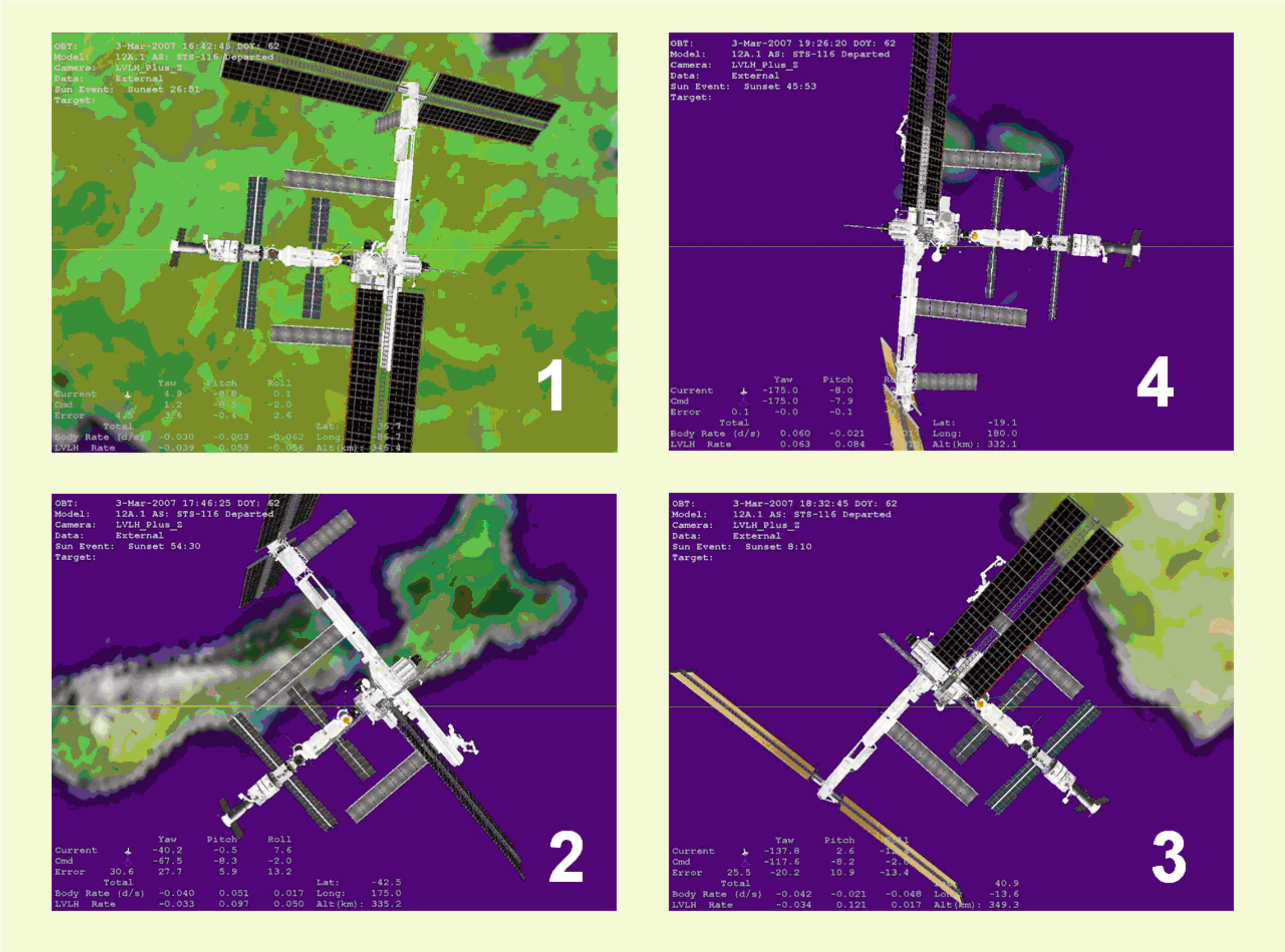
Landmarks-based SLAM



Optimal Control

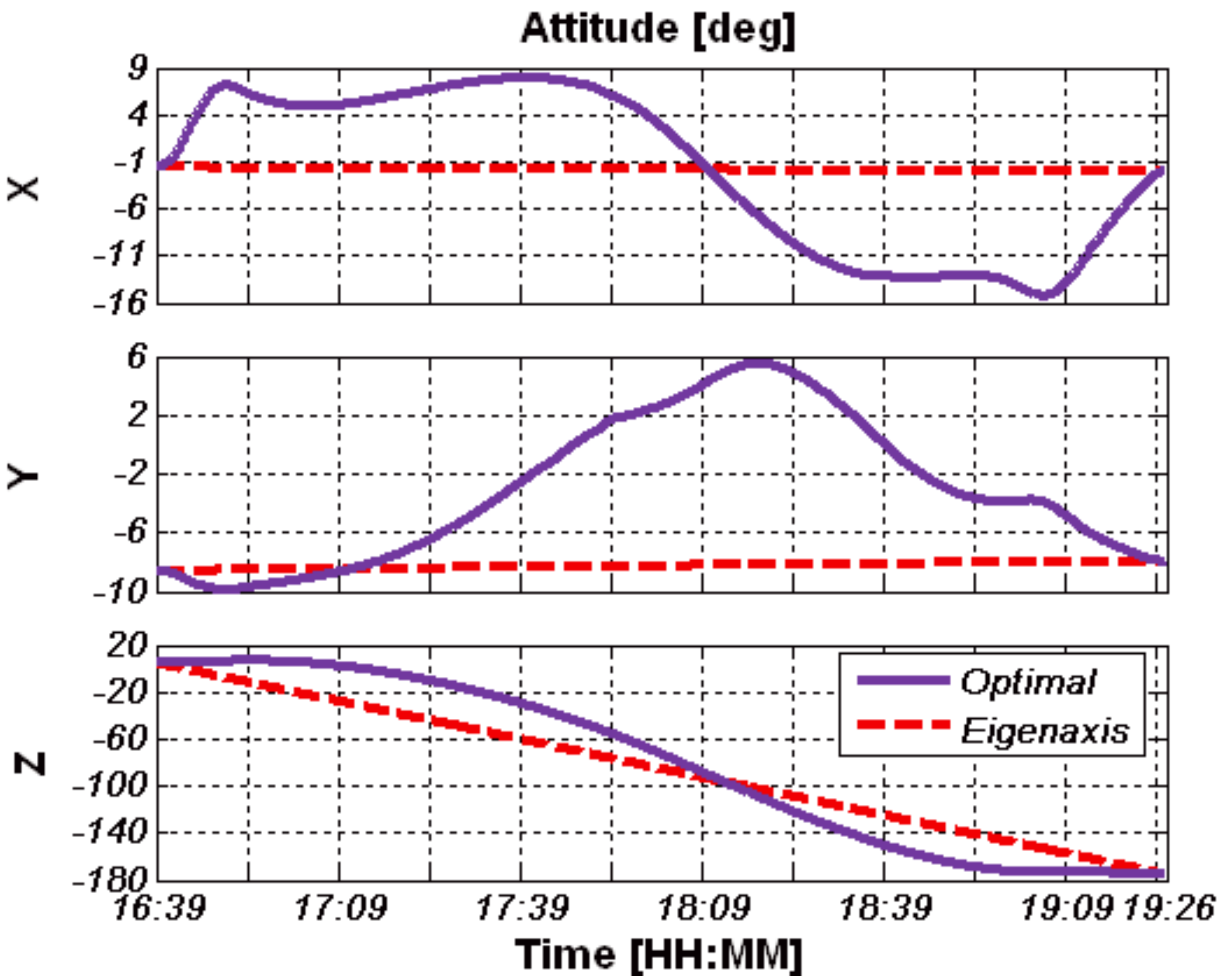
Pseudospectral Optimal Control

- Save NASA $1M !

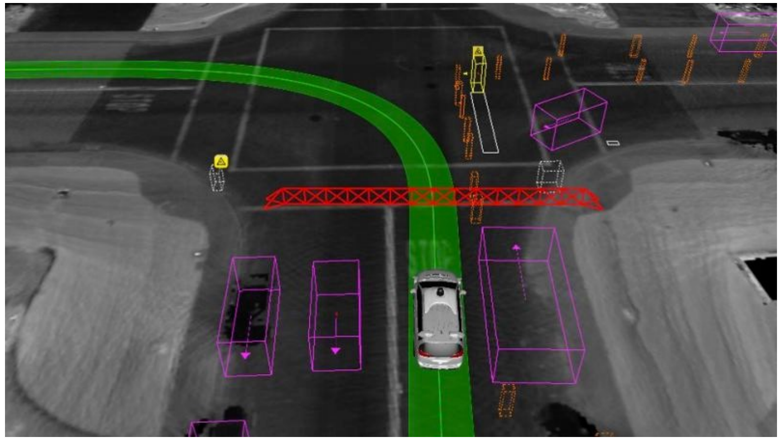


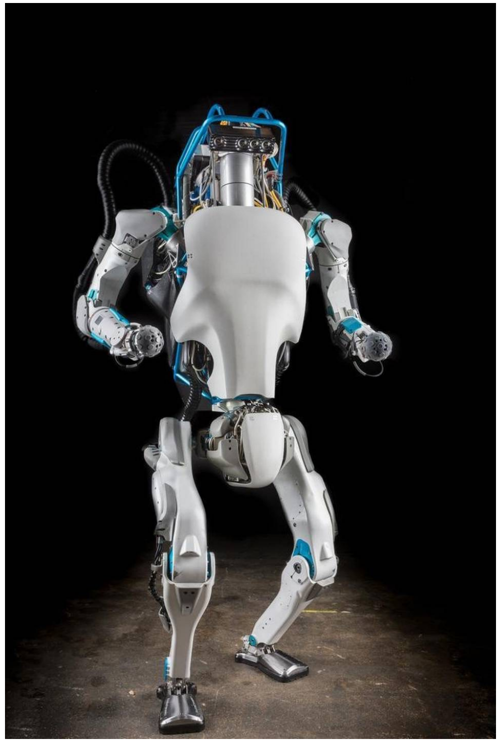
Pseudospectral Optimal Control

- Save NASA $1M !

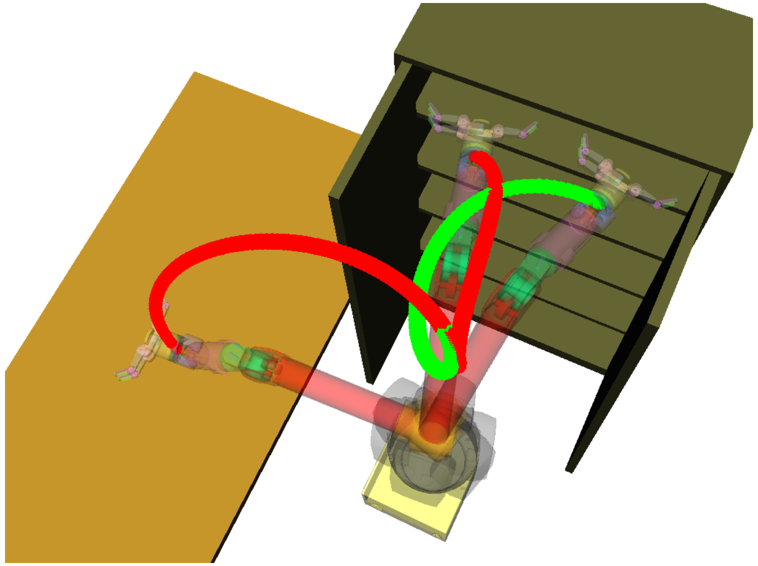


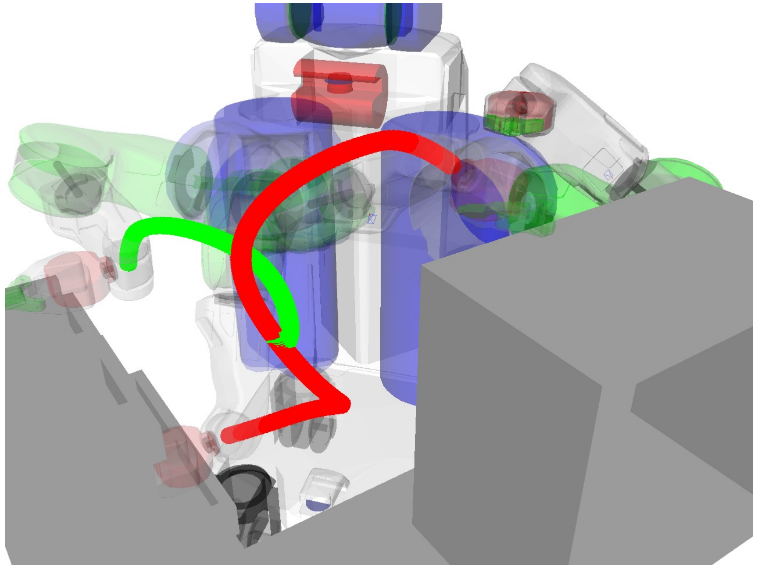
Motion Planning is one of the key capabilities for autonomous systems





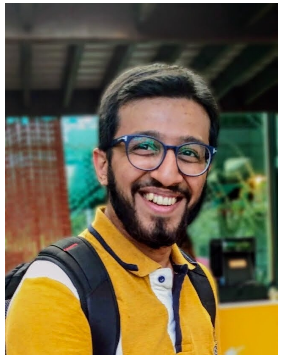
Factor graphs turn out to be an excellent framework in which to innovate in motion planning [Mukadam et al. IJRR ’18]





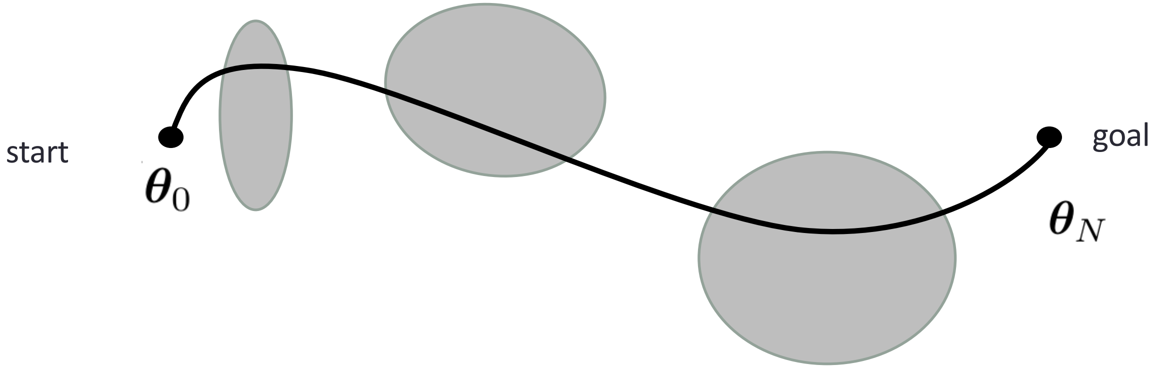
- Factors for:
  - Trajectory prior factors
  - Overall task-related objective
  - Obstacle avoidance, joint limits, etc…
- Fast incremental replanning using GTSAM

Gaussian-Process Motion Planning (GPMP) formulates motion planning as Probabilistic Inference in a space of smooth trajectories



Trajectory Prior Collision-free Likelihood

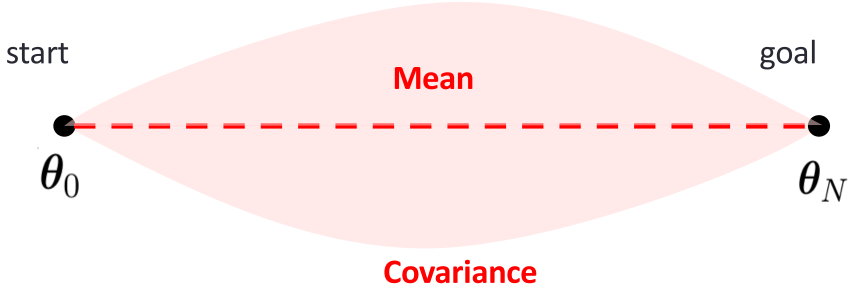
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left\{ P(\boldsymbol{\theta}) \prod_i P(c_i|\boldsymbol{\theta}_i) \right\}$$

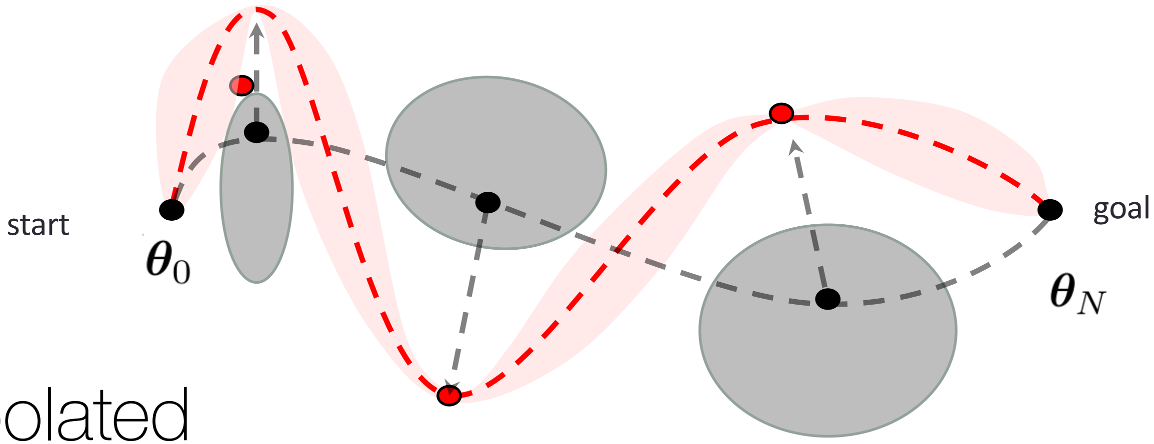


With Jing Dong, Mustafa Mukadam,& Byron Boots
Robotics: Science and Systems, 2016

Trajectory as Gaussian Process (GP)

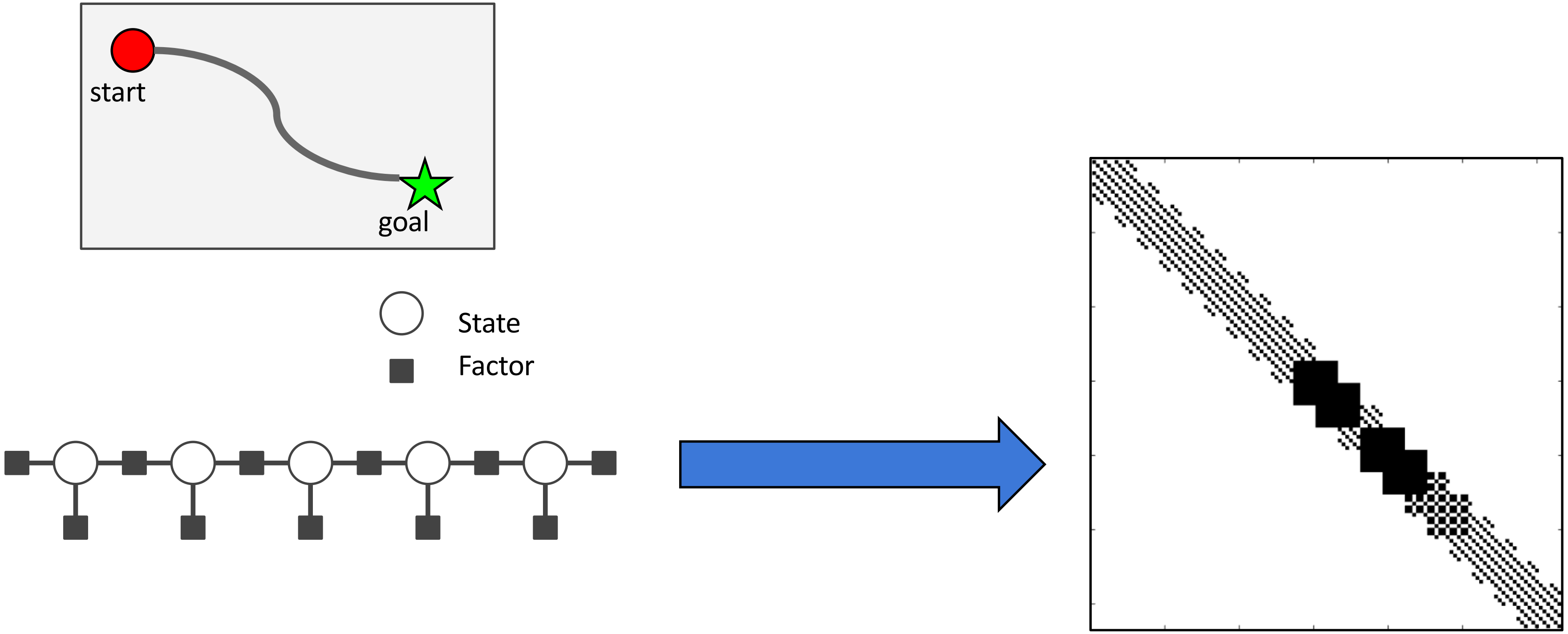
$$\boldsymbol{\theta}(t) \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\mathcal{K}}(t, t'))$$



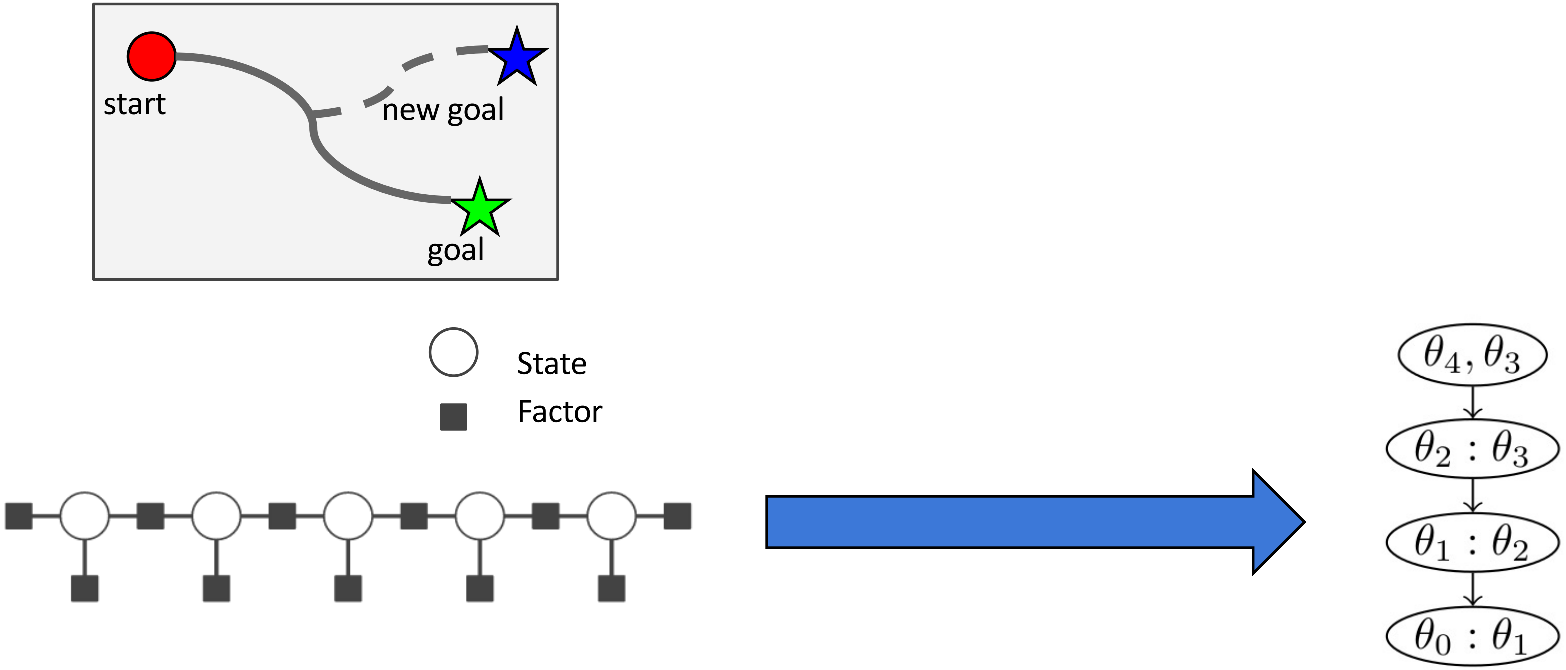


- Trajectory is interpolated
- Represented by a few states

GPMP2 uses factor graphs and sparsity to provide an efficient least-square MP solution

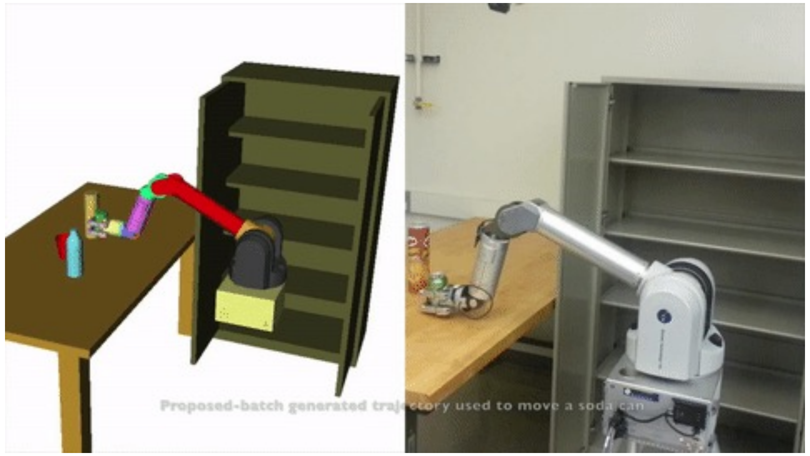


iGPMP2 uses the Bayes tree to efficiently re-plan, exploiting tricks we learned in incremental SLAM[2]

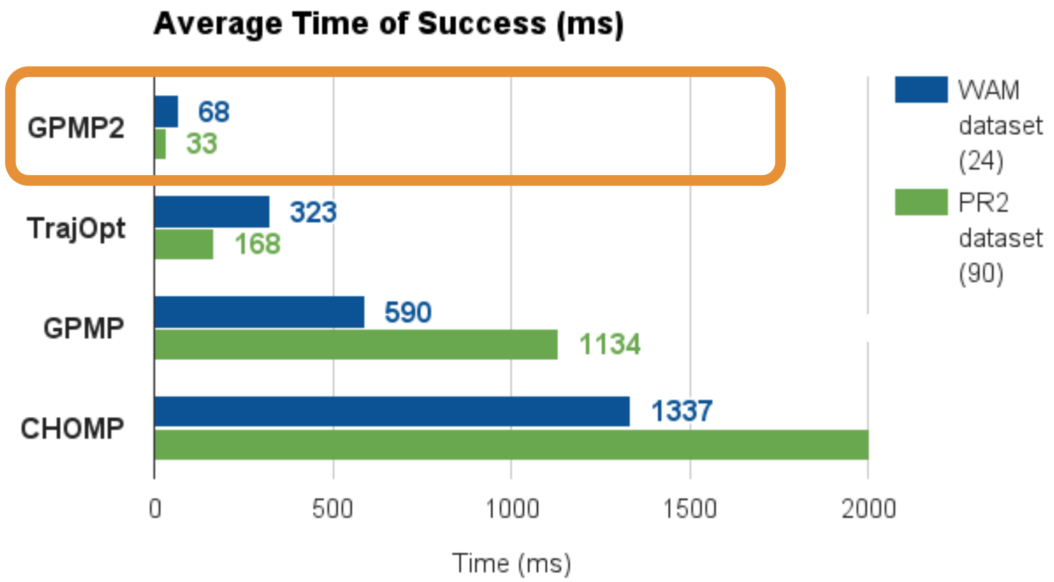


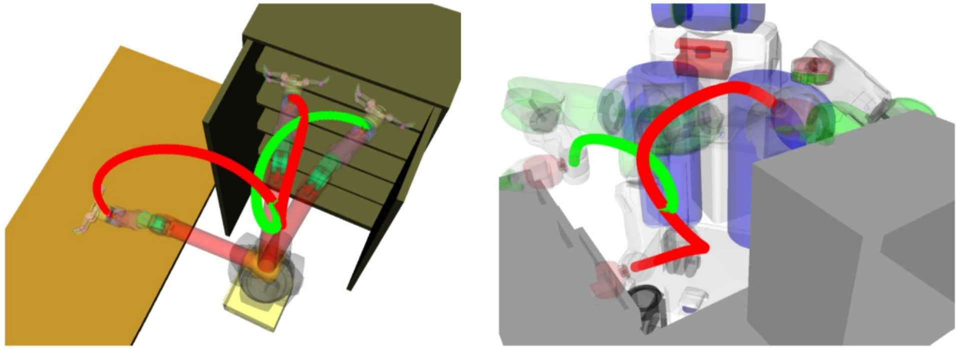
[2] Kaess et al. iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree, The *International Journal of Robotics Research* (2011)

Results

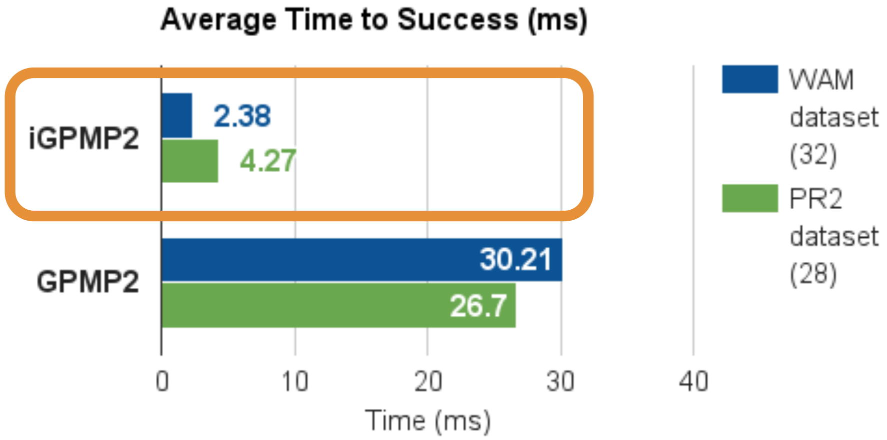


Planning Experiments



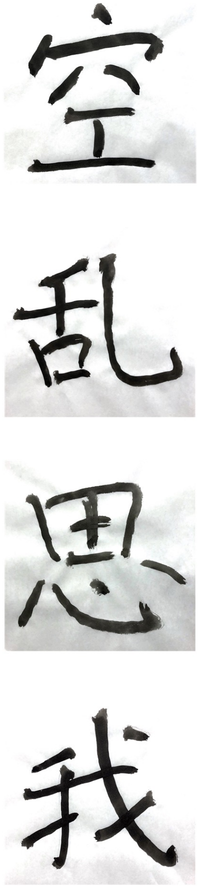


Re-planning Experiments

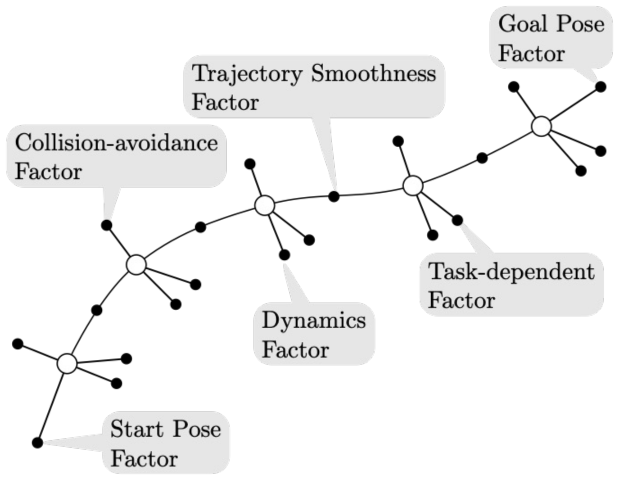


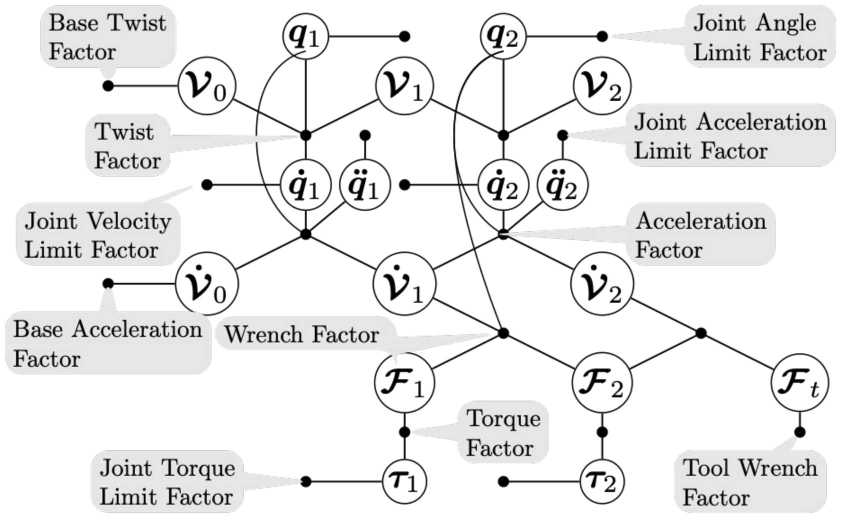
We used factor-graph-based motion planning to perform robot calligraphy [Wang et al. IROS ’20]

空
乱
思
我



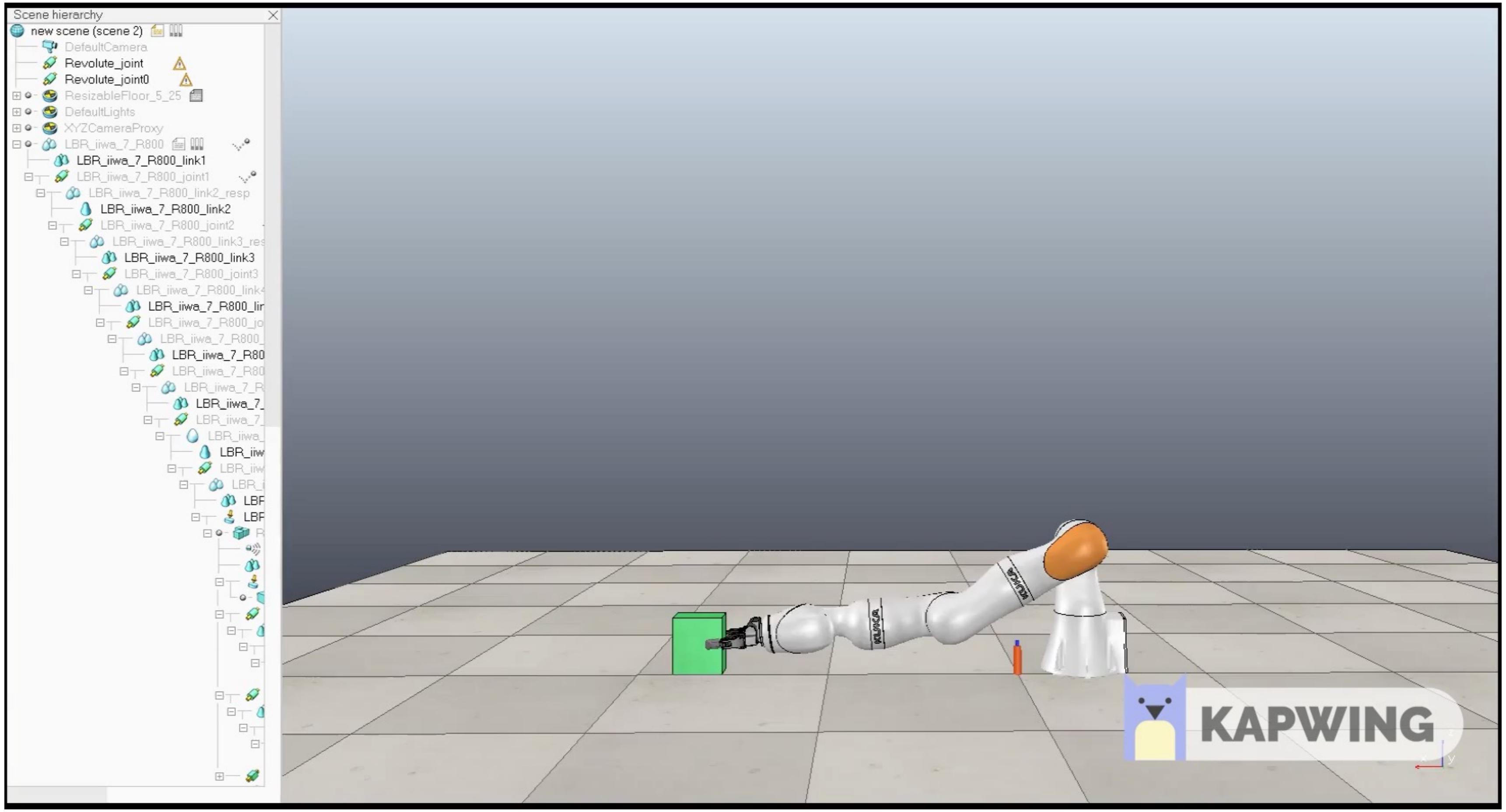
We used factors to encode robot dynamics and applied to kino-dynamic motion planning [Xie+'20]





- Recipe:
  - Take modern dynamics formulation
  - Turn into factor graph
  - Optimize with sparse (incremental) solvers

This example shows how kino-dynamics motion planning respects torque limits for weight-lifting



A sophisticated applications involves a jumping robot with pneumatic muscles, using kino-dynamic planning [IROS'21].

Factor Graph-Based Trajectory Optimization for a Pneumatically-Actuated Jumping Robot

Georgia Institute of Technology

Lucas Tiziani, Yetong Zhang, Frank Dellaert, and Frank L. Hammond III

Borg Lab

Georgia Tech ARMLAB ADAPTIVE ROBOTIC MANIPULATION LABORATORY