ECE276B: Planning & Learning in Robotics
Lecture 1: Introduction

Nikolay Atanasov

natanasov@ucsd.edu

UC San Diego

JACOBS SCHOOL OF ENGINEERING

Electrical and Computer Engineering

Outline

What Is This Class About?

- **ECE276A**: sensing and estimation in robotics:
  - how to model robot motion and observations
  - how to estimate (distribution of) robot/environment state $\mathbf{x}_t$ from history of observations $\mathbf{z}_{0:t}$ and control inputs $\mathbf{u}_{0:t-1}$
- **ECE276B**: planning and decision making in robotics:
  - how to select control inputs $\mathbf{u}_{0:t-1}$ to accomplish a task
- **References** (optional):
  - Dynamic Programming and Optimal Control: Bertsekas
  - Planning Algorithms: LaValle (`https://lavalle.pl/planning/`)
  - Reinforcement Learning: Sutton & Barto (`http://incompleteideas.net/book/the-book.html`)
  - Calculus of Variations and Optimal Control Theory: Liberzon (`http://liberzon.csl.illinois.edu/teaching/cvoc.pdf`)

Teaching Team

Instructor:

- **Nikolay Atanasov**
- Associate Professor, ECE Department
- Email: natanasov@ucsd.edu

Teaching Assistant:

- **Brian Lee**
- Postdoc, ECE Department
- Email: kmblee@ucsd.edu

Teaching Assistant:

- **Yinzhuang Yi**
- PhD Student, ECE Department
- Email: yiyi@ucsd.edu

Teaching Assistant:

- **Tianji Tang**
- MS Student, ECE Department
- Email: tit006@ucsd.edu

Prerequisites

- **Probability theory**: random variables, probability density function, expectation, covariance, total probability, conditional probability, Bayes rule
- **Linear algebra and systems**: eigenvalues, symmetric positive definite matrices, linear equations, linear systems of ODEs, matrix exponential
- **Optimization**: unconstrained optimization, gradient descent
- **Programming**: extensive experience with at least one language (python/C++/Matlab), classes/objects, data structures (e.g., queue, list), data input/output processing, plotting
- It is up to you to judge if you are ready for this course!
  - Consult with your classmates who took ECE276A
  - Take a look at the material from last year: `https://natanaso.github.io/ece276b2024`
  - If the first assignment seems hard, the rest will be hard as well

Website, Assignments, Grading

- Course website: `https://natanaso.github.io/ece276b`
- Includes links to:
  - **Canvas**: lecture recordings
  - **Piazza**: course announcement, Q&A, discussion – check Piazza regularly
  - **Gradescope**: homework submission and grades
- Assignments:
  - 3 theoretical homeworks (16% of grade)
  - 3 programming assignments in **python** + project report:
    - Project 1: Dynamic Programming (18% of grade)
    - Project 2: Motion Planning (18% of grade)
    - Project 3: Optimal Control (18% of grade)
  - Final exam (30% of grade)
- Grading:
  - standard grade scale (93%+ = A) plus curve based on class performance (e.g., if the top students have grades in the 86% - 89% range, then this will correspond to letter grade A)
  - **no late submissions**: work submitted past the deadline receives 0 credit

Collaboration and Academic Integrity

- Every assignment in this course is **individual**
- You are encouraged to discuss the assignments with other students in general terms but the **work you do and turn in should be completely your own**
- An important element of academic integrity is fully and correctly acknowledging any materials taken from the work of others. You should provide references for papers and **acknowledge in writing people you discuss the assignments with** in your submissions.
- **Cheating will not be tolerated**
- Instances of academic dishonesty will be penalized via grade reduction and may be referred to the Office of Student Conduct for adjudication

Project Report: Examples

https://natanaso.github.io/ref/Lauri_MonteCarloTreeSearch_ICRA15_Workshop.pdf

https://natanaso.github.io/ref/Schlotfeldt_AdversarialInfoAcquisition_RSS18_Workshop.pdf

Active Object Recognition via Monte Carlo Tree Search

Mikko Lauri, Nikolay Atanasov, George J. Pappas, and Risto Ritala

***Abstract*—This paper considers object recognition with a camera, whose viewpoint can be controlled in order to improve the recognition results. The goal is to choose a multi-view camera trajectory in order to minimize the probability of having misclassified objects and incorrect orientation estimates. Instead of using *offline* dynamic programming, the resulting stochastic optimal control problem is addressed via an *online* Monte Carlo tree search algorithm, which can handle various constraints and provides exceptional performance in large state spaces. A key insight is to use an active hypothesis testing policy to select camera viewpoints during the rollout stage of the tree search.**

I. Introduction

The goal of this paper is to choose a sequence of views for an RGB-D camera in order to identify the class and orientation of an object of interest (see Fig. 1). Unlike many existing approaches, which consider a next-best-view problem [1], [2], [3], we plan a multi-view camera trajectory to minimize the probability of having misclassified objects and incorrect orientation estimates. In previous work [4], we addressed a similar stochastic optimal control problem by casting it as a partially-observable Markov decision process. A point-based approximate solver [5] was used to obtain a non-greedy policy offline. Since repeated observations of the object from the same viewpoint provide redundant information, it is desirable to disallow viewpoint revisiting. The drawback of computing a policy offline is that revisiting and occlusion constraints are hard to incorporate and if the environment were to change, the computed policy would no longer be useful. The idea of this paper is to apply Monte Carlo tree search (MCTS, [6], [7]) to the active object recognition problem. MCTS is a best-first *online* planning approach which can handle various constraints and has exceptional performance in large challenging domains such as game solving [8], [9] and belief-space planning in robotics [10], [11], [12].

II. Problem Formulation

Let the camera pose at time t be $x_t \in \mathcal{X} \subset SE(3)$, where $\mathcal{X}$ is a finite set of viewpoints on a sphere centered at the object's location (see Fig. 1). At time t, the camera can move to any of the viewpoints in $\mathcal{X}$ and pays a cost $g(x_{t-1}, x_t)$ which captures the energy expenditure. Let the true (unknown) class of the observed object be $c \in \mathcal{C}$. We formulate hypotheses about the class and orientation of the object:

$H(c, r)$: the object class is $c \in \mathcal{C}$ with orientation $r \in \mathcal{R}(c)$,

M. Lauri and R. Ritala are with the Department of Automation Science and Engineering, Tampere University of Technology, P.O. Box 692, FI-33101, Tampere, Finland, {mikko.lauri, risto.ritala}@tut.fi.
N. Atanasov and G. Pappas are with GRASP Lab, University of Pennsylvania, Philadelphia, PA 19104, USA, {atanasov, pappasg}@seas.upenn.edu. This work was supported by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

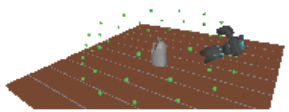


Fig. 1: Setup for the active object recognition problem. The camera position is restricted to a set of viewpoints (green) on a sphere centered at the object's location. The task is to choose a camera control policy, which minimizes the movement cost and the probability of misclassification.

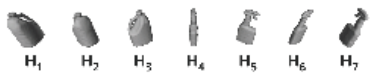


Fig. 2: An example set of hypotheses about the class and orientation of an unknown object

where $\mathcal{R}(c) \subset SO(3)$ is a small finite set of discrete[1] orientations for each class $c \in \mathcal{C}$. For notational convenience, let $i = 1, \ldots, M$ be an enumeration of the set $\{(c, r) \mid c \in \mathcal{C}, r \in \mathcal{R}(c)\}$ and denote the hypotheses by H_i (see Fig. 2).

Offline, a 3-D model database is used to train a viewpoint-pose tree [4] by extracting point clouds from views on a sphere around each model. A set of Fast Point Feature Histograms [13] is extracted from each point cloud and the clouds are arranged in a tree structure according to their feature similarity (see [4] for details). Given a query point cloud, the best-matching cloud from the tree carries information about the class and orientation of the observed object and about the quality of the feature match. Thus, the tree provides an observation $z_t \in \mathcal{Z}$, consisting of the class, orientation, and confidence score of the top match. The model database is used to learn the probability density function (pdf) $q(\cdot \mid x, H_i)$ of z conditioned on any camera viewpoint $x \in \mathcal{X}$ and any hypothesis H_i, $i = 1, \ldots, M$.

Problem. *Given a camera pose $x_0 \in \mathcal{X}$, a prior $p_0 \in [0,1]^M$ on the true hypothesis H_i, and a planning horizon $T < \infty$, choose a sequence of functions $\mu_t : (\mathcal{Z} \times \mathcal{X})^{t+1} \to \mathcal{X}$ for $t = 0, \ldots, T-1$, which minimizes the average movement cost and the probability of an incorrect hypothesis:*

$$\begin{aligned} \min_{\mu_{0:T-1}} \ & \frac{1}{T} \sum_{t=1}^{T} g(x_{t-1}, x_t) + \lambda Pe(T) \\ \text{s.t. } & x_{t+1} = \mu_t(z_{0:t}, x_{0:t}), \quad t = 0, \ldots, T-1, \\ & x_{t+1} \notin \{x_0, \ldots, x_t\}, \quad t = 0, \ldots, T-1, \\ & z_t \sim q(\cdot \mid x_t, H_i), \quad t = 0, \ldots, T, \\ & p_t = b(p_{t-1}, z_t, x_t), \quad t = 1, \ldots, T, \end{aligned} \tag{1}$$

[1]After a hypothesis is chosen, the discrete orientation estimate can be refined by aligning the observed object surface to the corresponding model in the training database, e.g., by using the iterative closest point algorithm.

Adversarial Information Acquisition

Brent Schlotfeldt
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104
Email: brentsc@seas.upenn.edu

Nikolay Atanasov
Dept. of Electrical and Computer Engineering
University of California San Diego
La Jolla, CA 92093
Email: natanasov@ucsd.edu

George J. Pappas
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104
Email: pappasg@seas.upenn.edu

***Abstract*—This paper considers a non-cooperative two-player game modeling the problem of adversarial information acquisition in robotics. Each robot is equipped with a sensor, and is tasked with choosing control inputs that maximize an information (e.g. entropy) about the other player, while keeping its own state as uncertain to the other player as possible, subject to the available sensors. This adversarial information gathering problem has applications in surveillance, or search-and-rescue missions where the agent whose state is to be estimated may try to actively avoid the sensing robot. We formulate the problem of adversarial information acquisition, and provide an initial solution based on a variant of Monte Carlo Tree Search for simultaneous action games.**

I. Introduction

In this paper, we consider the problem of two robots interacting in an adversarial game where each robot attempts to estimate the state of its adversary, while keeping its own state hidden. This problem can have applications in search-and-rescue applications, where the agent to be found is mobile, and actively evades the sensing robot. In these problems, it is important to both accurately localize the target agent, while keeping one's own state hidden so that the target's ability to actively evade is reduced.

There is much prior work in the literature concerning the dynamics of pursuit-evasion in mobile robotic settings[3]. Approaches to the pursuit-evasion problem are split between considering the worst-case evader (adversary), and using probabilistic frameworks which consider the expected case. A common theme in the pursuit-evasion literature is the objective of reducing the distance to the evader to zero, or forcing the evader into a sensing footprint. In contrast, our problem is formulated using a probabilistic approach which optimizes an information theoretic quantity, namely entropy, about the distribution of the target to be tracked. Rather than closing distance to the target, our approach aims to produce the best estimate of the target's state, subject to the sensors available. Our previous work considers the information acquisition problem for target tracking [1], however this work assumes that the target being tracked moves independently of the sensing robot, and crucially is not trying to actively evade the sensing robot. In this work, the problem formulation is symmetric in the sense that the adversary is trying to maximize information gained about us, and also minimize the information we can gain about it.

We also draw attention to some biological inspiration for this problem. Motion camouflage [9] is a strategy utilized by dragonflies, which enables them to capture their prey by minimizing the optical flow of their motion. Mischiati and Krishnaprasad [8] consider the problem of mutual motion camouflage, where two agents each pursue each other, but attempt to maintain a constant bearing to avoid detection by the other agent. Our problem is related, but rather than considering pursuit-evasion, we consider the dynamics of an *adversarial information gathering game*.

In the most general case, the information acquisition game proposed is a stochastic game and is difficult to solve. McEneaney [7] discusses a class of stochastic games with finite-dimensional solutions and dynamic programming algorithms to solve them. With some assumptions on the motion and observation models of the agents in our game, the problem can be simplified to a deterministic game. McEneaney [6] introduces a curse-of-dimensionality free max-plus method for deterministic game problems, which is likely to be very applicable to the linear Gaussian version of the information-theoretic game introduced here. Additionally, Grünwald and Dawid [4] present a game-theoretic argument that maximizing entropy and minimizing worst-case expected loss are duals of each other. A comprehensive treatment on adversarial reasoning, is provided in the book by Kott and McEneaney [5]. The approach taken in this work is a variant of Monte Carlo Tree Search [11], for simultaneous action games. We present the details of this approach in Sec. III.

II. Problem Formulation

Consider a two-player partial information game with simultaneous moves. Each player $i \in \{1, 2\}$ has a state $x_{i,t}$ that evolves according to the following motion model:

$$x_{i,t+1} = f_i(x_{i,t}, u_{i,t}, w_{i,t}) \tag{1}$$

where $u_{i,t} \in \mathcal{U}_i$ is a *finite* space of admissible moves (control inputs) and $w_{i,t}$ is a random variable specifying the motion noise. Player i can observe its own state $x_{i,t}$ and chooses its moves with the objective of tracking the evolution of the state of the other player. Each player is equipped with a sensor used to collect information about the other player according to the following observation model:

$$z_{i,t} = h_i(x_{i,t}, x_{j,t}, v_{i,t}) \tag{2}$$

Course Schedule (Tentative)

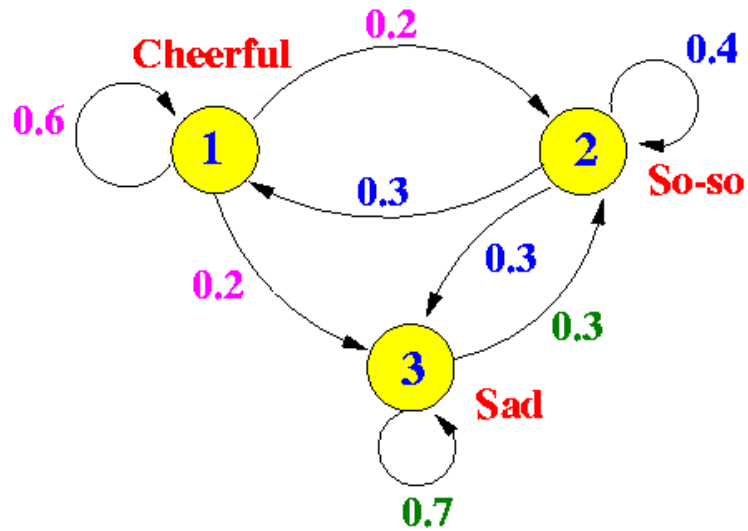
Date	Lecture	Materials	Assignments
Mar 31	Introduction		
Apr 02	Markov Chains	Grinstead-Snell Ch. 11	
Apr 07	Catch-up		
Apr 09	Markov Decision Processes	Bertsekas 1.1-1.2	HW1, PR1
Apr 14	Dynamic Programming	Bertsekas 1.3-1.4	
Apr 16	Deterministic Shortest Path	Bertsekas 2.1-2.3	
Apr 21	Configuration Space	LaValle 4.3, 6.2-6.3	
Apr 23	Search-based Planning	LaValle 2.1-2.3, JPS	
Apr 28	Catch-up		
Apr 30	Anytime Incremental Search	RTAA*, ARA*, AD*, Anytime Search	HW2, PR2
May 05	Sampling-based Planning	LaValle 5.5-5.6	
May 07	Catch-up		
May 12	Infinite-Horizon Optimal Control	Bertsekas 7.1-7.3, Sutton-Barto Ch 4	
May 14	Infinite-Horizon Optimal Control	Bertsekas 7.1-7.3, Sutton-Barto Ch 4	
May 19	Catch-up		
May 21	Model-Free Prediction	Sutton-Barto 6.1-6.3	HW3, PR3
May 26	Model-Free Control	Sutton-Barto 6.4-6.7	
May 28	Value Function Approximation	Sutton-Barto Ch. 9	
Jun 02	Linear Quadratic Control	Bertsekas 4.1	
Jun 04	Continuous-Time Optimal Control	Bertsekas Ch. 3, Liberzon Ch. 2.4 and Ch. 4	
Jun 11	Final Exam		

- Check website for updates: `https://natanaso.github.io/ece276b`

Outline

Markov Chain and Markov Decision Process

- **Markov Chain**: probabilistic model of system transitions
  - The state $\mathbf{x}_t$ can be discrete or continuous
  - The state transitions are random, determined by a transition matrix or a transition kernel

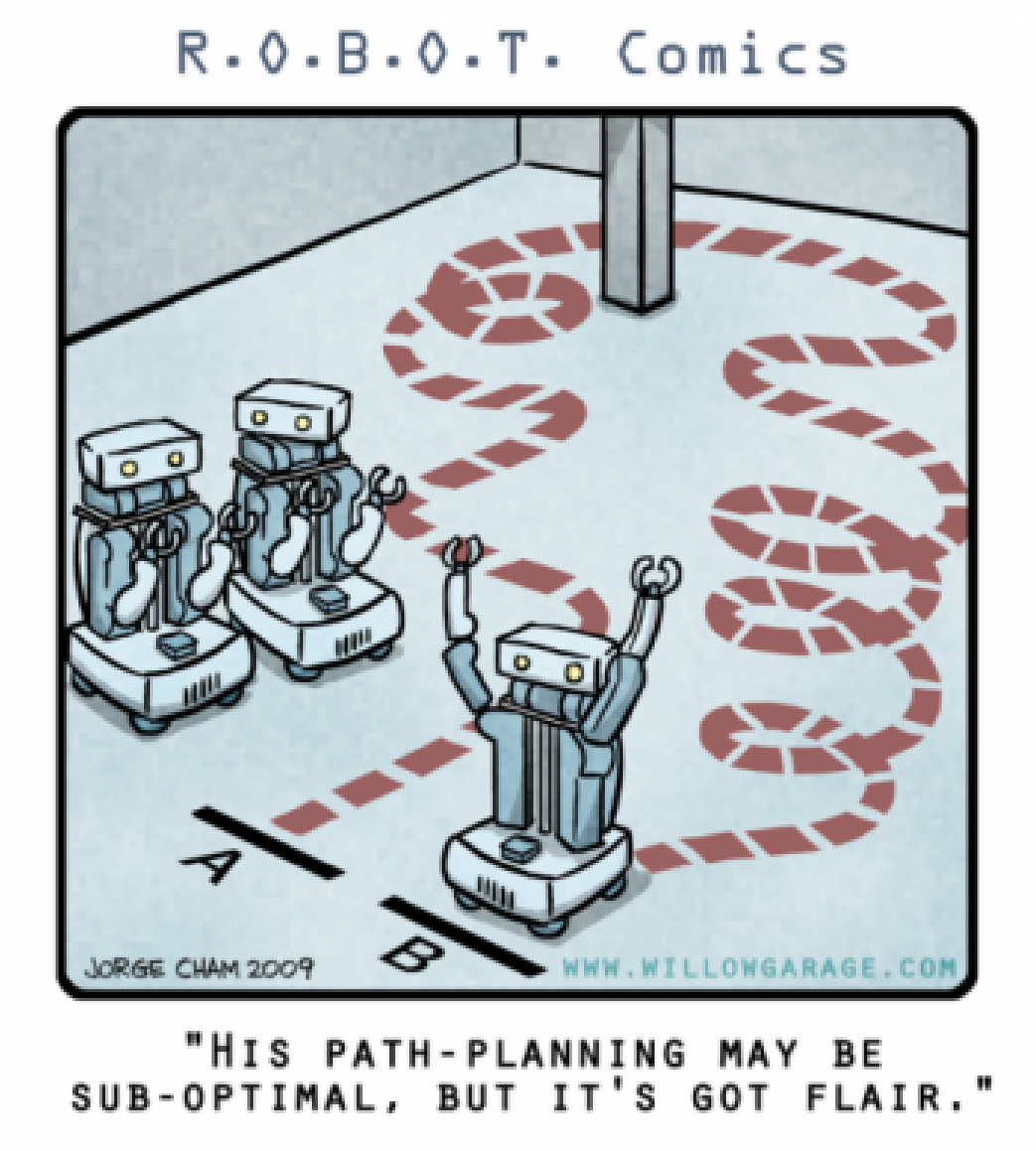


- **Markov Decision Process**: Markov chain whose transition probabilities are decided by control inputs $\mathbf{u}_t$
- Motion planning, optimal control, and reinforcement learning problems are formalized using a Markov decision process

$$P = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.3 & 0.4 & 0.3 \\ 0.0 & 0.3 & 0.7 \end{bmatrix}$$

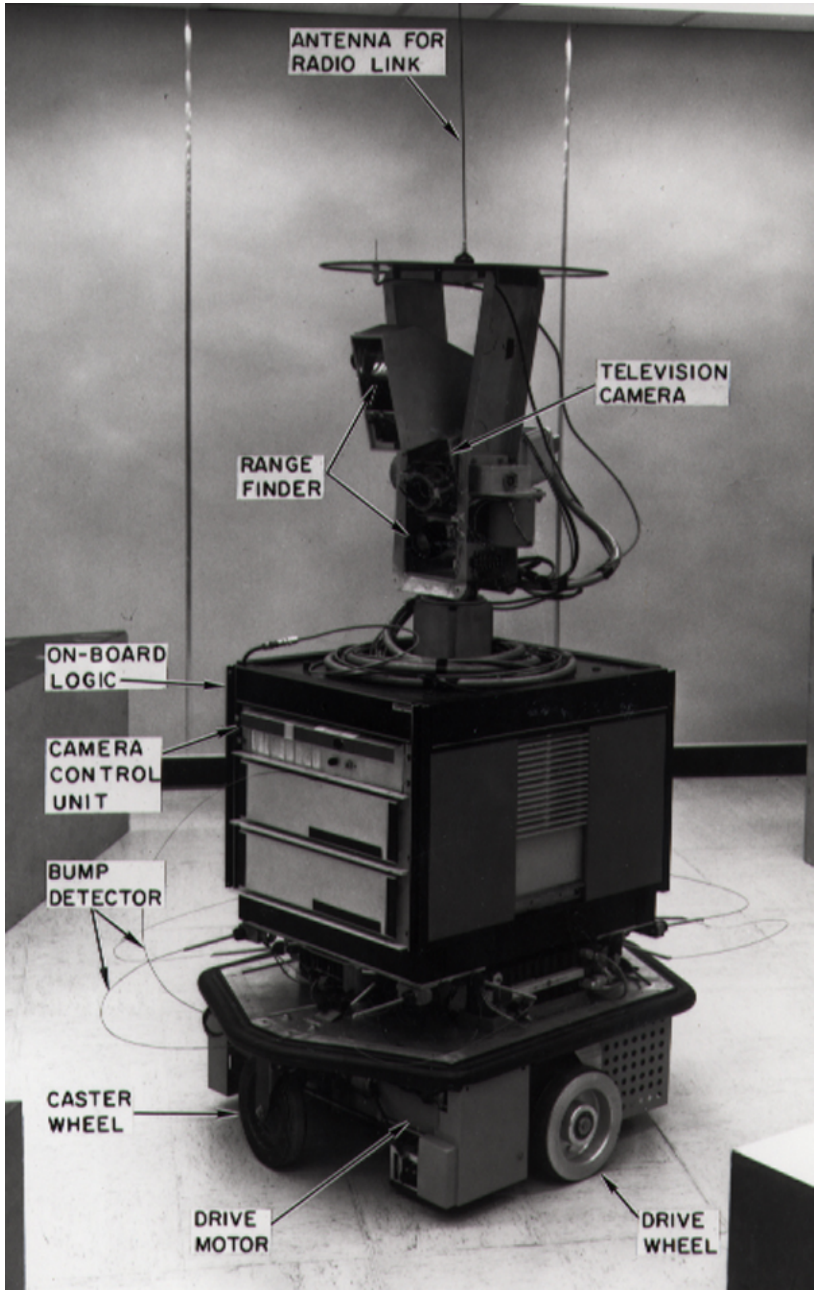
$$P_{ij} = \mathbb{P}(x_{t+1} = j \mid x_t = i)$$

Motion Planning

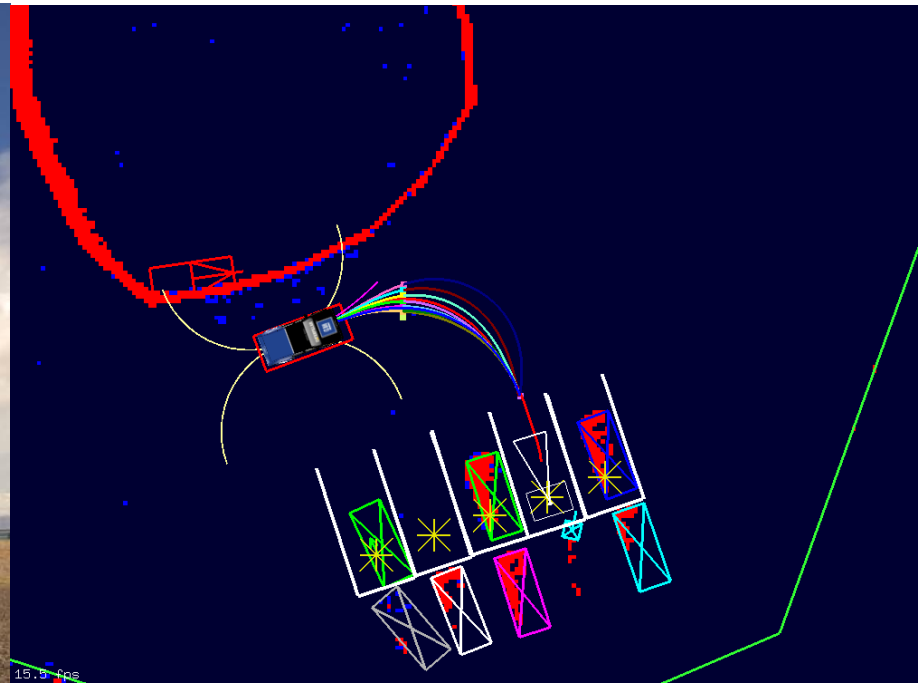


A* Search

- Developed by Hart, Nilsson and Raphael of Stanford Research Institute in 1968 for the Shakey robot
- MDP with deterministic transitions, i.e., directed graph
- Minimize cumulative transition costs subject to a goal constraint
- Graph search using a specific node visitation rule
- Video: `https://youtu.be/qXdn6ynwpiI?t=3m55s`

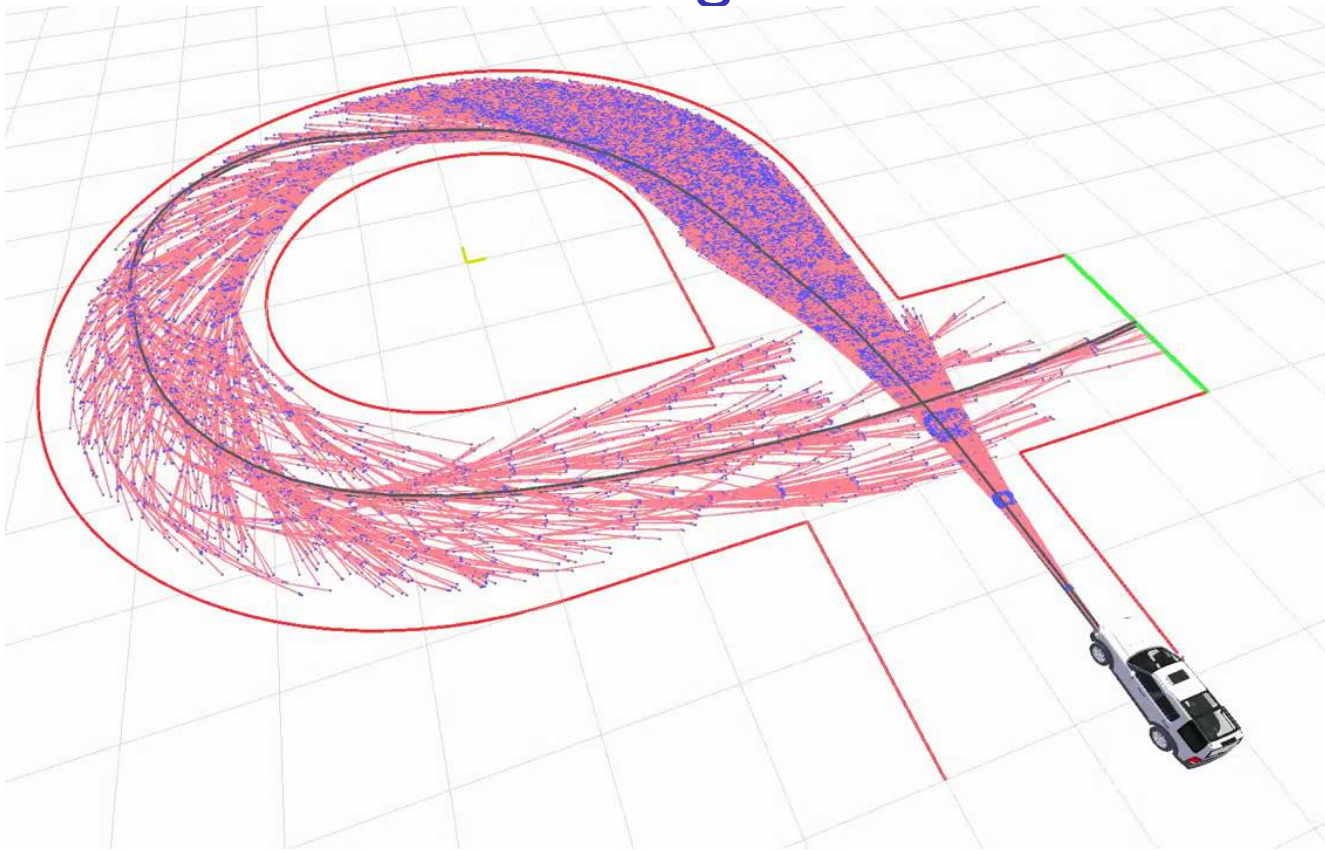


Search-Based Motion Planning



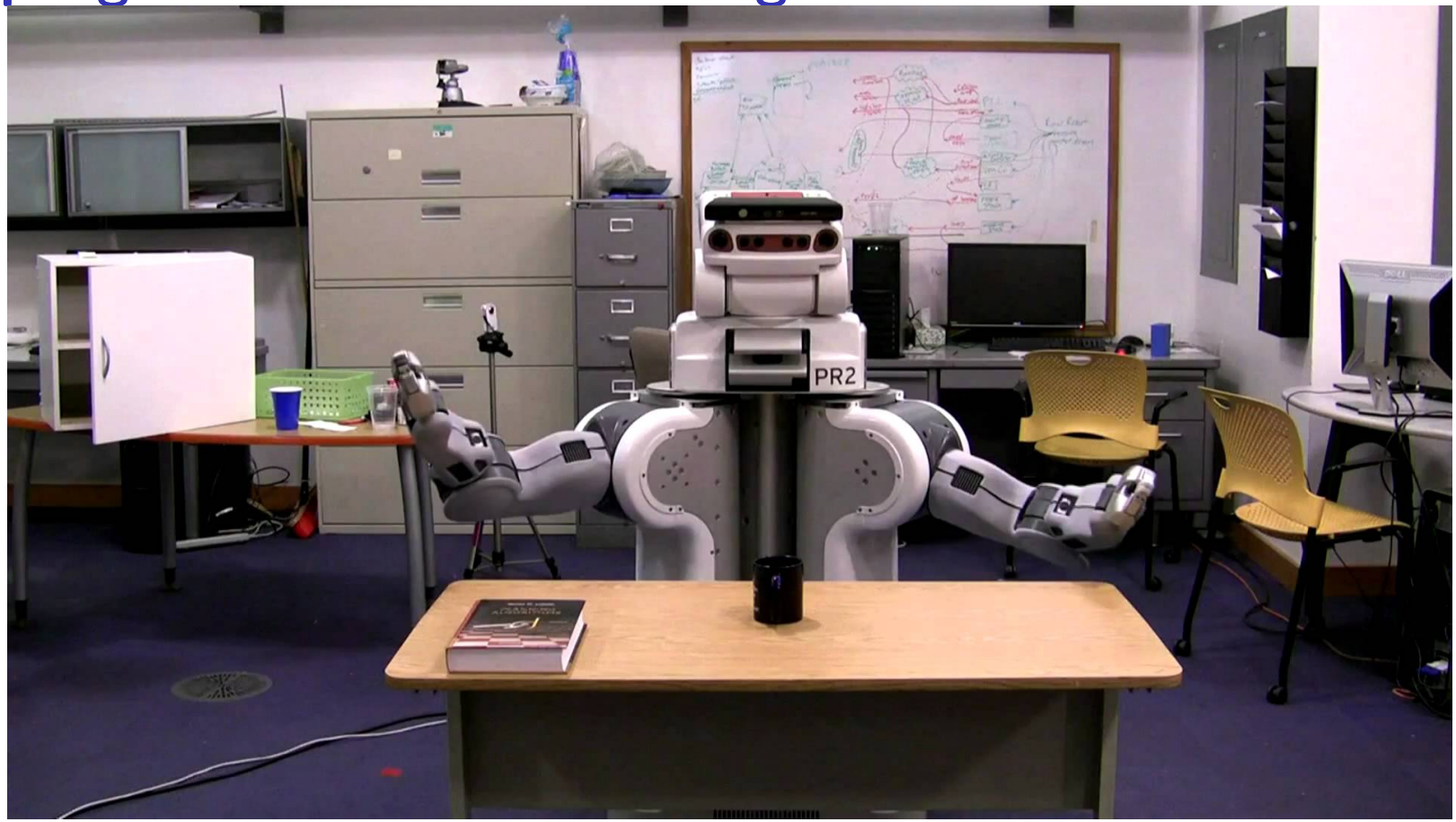
- CMU's autonomous car used search-based motion planning in the DARPA Urban Challenge in 2007
- Video: `https://www.youtube.com/watch?v=4hFhl0Oi8KI`
- Video: `https://www.youtube.com/watch?v=qXZt-B7iUyw`
- Paper: Likhachev and Ferguson, "Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles," IJRR, 2009, `http://journals.sagepub.com/doi/pdf/10.1177/0278364909340445`

Sampling-Based Motion Planning



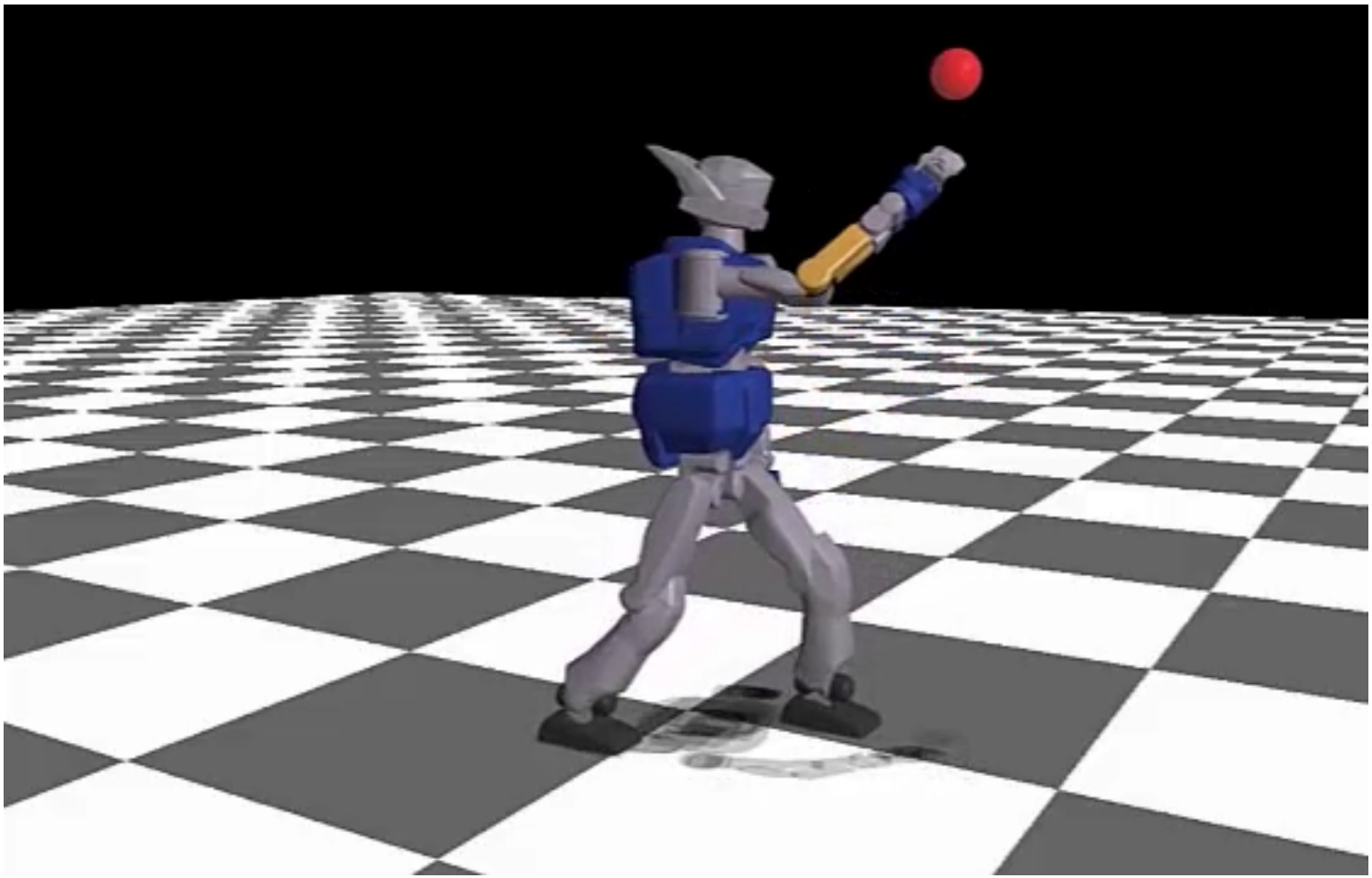
- RRT* algorithm on a high-fidelity car model – 270 degree turn
- Video: `https://www.youtube.com/watch?v=p3nZHnOWhrg`
- Video: `https://www.youtube.com/watch?v=LKL5qRBiJaM`
- Karaman and Frazzoli, "Sampling-based algorithms for optimal motion planning," IJRR, 2011, `http://journals.sagepub.com/doi/pdf/10.1177/0278364911406761`

Sampling-Based Motion Planning



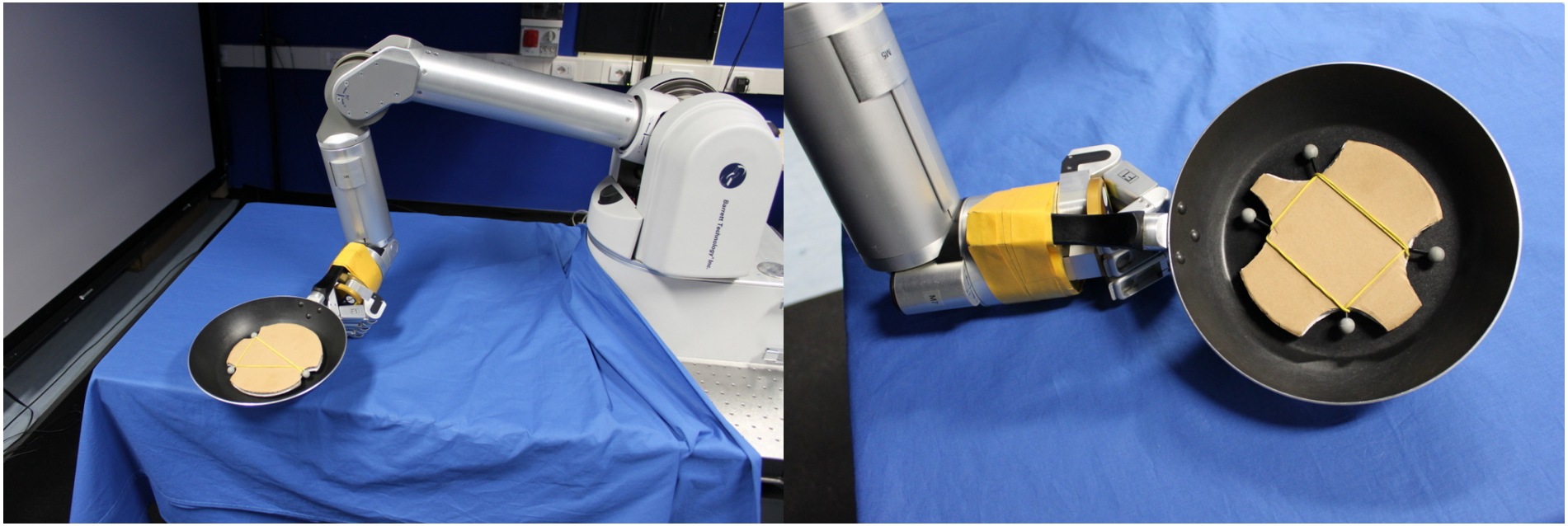
- RRT algorithm on the PR2 – planning with both arms (12 DOF)
- Video: `https://www.youtube.com/watch?v=vW74bC-Ygb4`
- Karaman and Frazzoli, "Sampling-based algorithms for optimal motion planning," IJRR, 2011, `http://journals.sagepub.com/doi/pdf/10.1177/0278364911406761`

Optimal Control using Dynamic Programming



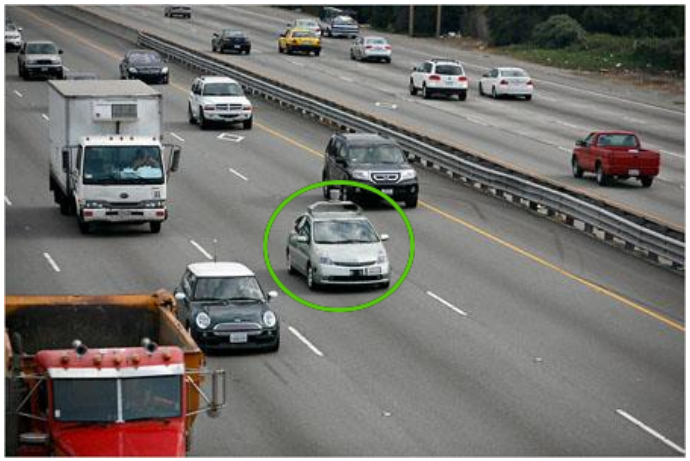
- Video: `https://www.youtube.com/watch?v=tCQSSkBH2NI`
- Tassa, Mansard and Todorov, "Control-limited Differential Dynamic Programming," ICRA, 2014, `http://ieeexplore.ieee.org/document/6907001/`

Model-Free Reinforcement Learning



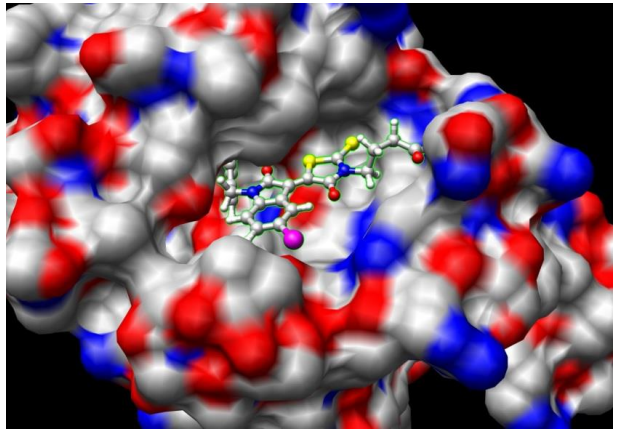
- A robot learns to flip pancakes
- Video: `https://www.youtube.com/watch?v=W_gxLKSsSIE`
- Kormushev, Calinon and Caldwell, "Robot Motor Skill Coordination with EM-based Reinforcement Learning," IROS, 2010, `http://www.dx.doi.org/10.1109/IROS.2010.5649089`

Applications of Optimal Control & Reinforcement Learning



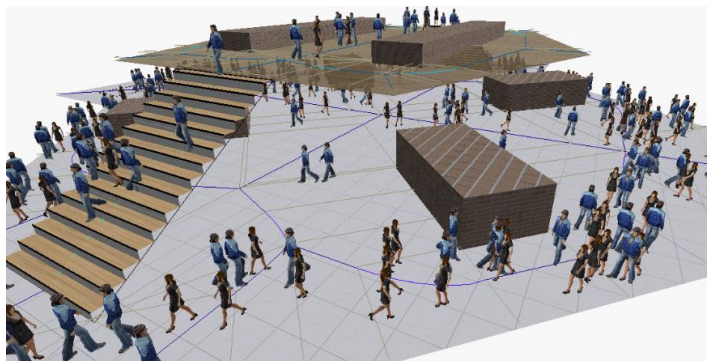
(a) Autonomous Driving

(b) Marketing

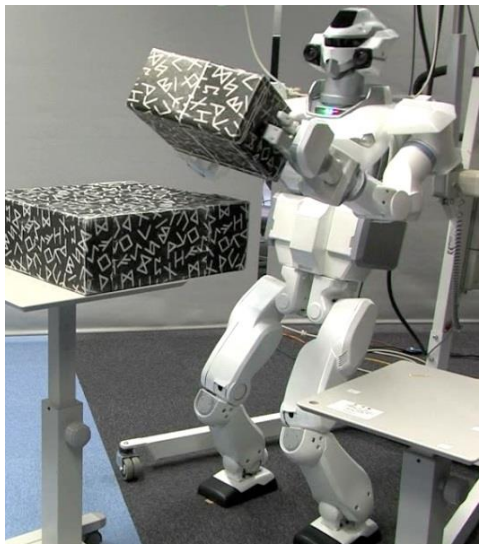


(c) Computational Biology

(d) Games



(e) Character Animation



(f) Robotics

Outline

Model

- discrete **time** $t \in \{0, \dots, T\}$ with finite or infinite **horizon** T
- **state** $\mathbf{x}_t \in \mathcal{X}$ and **state space** $\mathcal{X}$
- **control** $\mathbf{u}_t \in \mathcal{U}$ and **control space** $\mathcal{U}$
- **motion noise** $\mathbf{w}_t$: random vector with known probability density function (pdf), independent of $\mathbf{w}_\tau$ for $\tau \neq t$ conditioned on $\mathbf{x}_t$ and $\mathbf{u}_t$
- **motion model**: a function f or equivalently a pdf p_f describing the change in the state $\mathbf{x}_t$ when a control input $\mathbf{u}_t$ is applied:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \qquad \text{or} \qquad \mathbf{x}_{t+1} \sim p_f(\cdot \mid \mathbf{x}_t, \mathbf{u}_t)$$

  - **Markov assumption**: $\mathbf{x}_{t+1}$ conditioned on $\mathbf{u}_t$ and $\mathbf{x}_t$ is independent of all other variables

Control Policy

- **control policy**: function $\pi_t : \mathcal{X} \mapsto \mathcal{U}$ that maps state $\mathbf{x}$ at time t to control input $\mathbf{u}$
- A policy defines fully at any time t and any state $\mathbf{x}$ which control $\mathbf{u}$ to apply
- A policy can be:
  - **stationary** ($\pi_0 \equiv \pi_1 \equiv \cdots$) or **non-stationary** ($\pi_0 \not\equiv \pi_1 \not\equiv \cdots$)
  - **deterministic** ($\mathbf{u}_t = \pi_t(\mathbf{x}_t)$) or **stochastic** ($\mathbf{u}_t \sim \pi_t(\cdot \mid \mathbf{x}_t)$)
  - **open-loop** ($\mathbf{u}_t$ is selected independent of $\mathbf{x}_t$) or **closed-loop** ($\mathbf{u}_t = \pi_t(\mathbf{x}_t)$ depends on $\mathbf{x}_t$)
- A control policy induces a transition from state $\mathbf{x}_t$ at time t with control input $\mathbf{u}_t = \pi_t(\mathbf{x}_t)$ to state $\mathbf{x}_{t+1} \sim p_f(\cdot \mid \mathbf{x}_t, \mathbf{u}_t)$ according to the motion model $p_f(\cdot \mid \mathbf{x}_t, \mathbf{u}_t)$

Optimal Control Problem

- **stage cost** $\ell(\mathbf{x}, \mathbf{u})$ measures the cost of applying control $\mathbf{u}$ in state $\mathbf{x}$
- **terminal cost** $\mathrm{q}(\mathbf{x})$ measures the cost of terminating at state $\mathbf{x}$
- **value function** $V_t^\pi(\mathbf{x})$ of policy π is the expected long-term cost of starting at state $\mathbf{x}$ at time t and following transitions induced by $\pi_t, \pi_{t+1}, \ldots, \pi_{T-1}$:

$$V_t^\pi(\mathbf{x}) := \mathbb{E}_{\mathbf{x}_{t+1:T}}\left[\underbrace{\mathrm{q}(\mathbf{x}_T)}_{\text{terminal cost}} + \sum_{\tau=t}^{T-1} \underbrace{\ell(\mathbf{x}_\tau, \pi_\tau(\mathbf{x}_\tau))}_{\text{stage cost}} \,\middle|\, \mathbf{x}_t = \mathbf{x} \right]$$

- **optimal control problem**: given initial state $\mathbf{x}$ at time t, determine a policy that minimizes the value function $V_t^\pi(\mathbf{x})$:
  - **optimal value**: $V_t^*(\mathbf{x}) = \min_\pi V_t^\pi(\mathbf{x})$
  - **optimal policy**: $\pi^*(\mathbf{x}) \in \arg\min_\pi V_t^\pi(\mathbf{x})$

Optimal Control Problem Types

- **deterministic** (no motion noise) vs **stochastic** (with motion noise)
- **fully observable** ($\mathbf{z}_t = \mathbf{x}_t$) vs **partially observable** ($\mathbf{z}_t \sim p_h(\cdot|\mathbf{x}_t)$)
  - Markov Decision Process (MDP) vs Partially Observable Markov Decision Process (POMDP)
- **stationary** vs **non–stationary** (time-dependent motion $p_{f,t}$ and cost ℓ_t)
- **discrete** vs **continuous** state space $\mathcal{X}$
  - tabular approach vs function approximation
- **discrete** vs **continuous** control space $\mathcal{U}$:
  - tabular approach vs optimization
- **discrete** vs **continuous** time t
- **finite** vs **infinite** horizon T
- reinforcement learning (p_f, ℓ, $\mathfrak{q}$ are unknown):
  - **Model-based RL**: explicitly approximate the models $\hat{p}_f$, $\hat{\ell}$, $\hat{\mathfrak{q}}$ from data and apply optimal control algorithms
  - **Model-free RL**: directly approximate V_t^* and π_t^* without approximating the motion or cost models

Naming Conventions

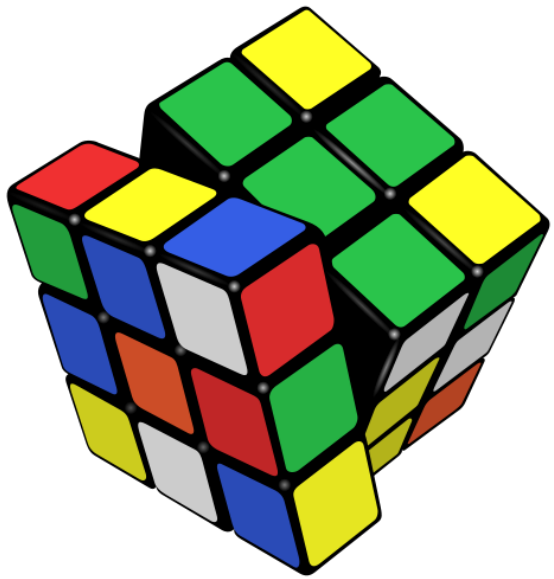
- The problem is called:
  - **Motion planning** (MP): when the motion model p_f is known and deterministic and the cost functions ℓ, $\mathfrak{q}$ are known
  - **Optimal control** (OC): when the motion model p_f is known but may be stochastic and the cost functions ℓ, $\mathfrak{q}$ are known
  - **Reinforcement learning** (RL): when the motion model p_f and cost functions ℓ, $\mathfrak{q}$ are unknown but samples $\mathbf{x}_t$, $\ell(\mathbf{x}_t, \mathbf{u}_t)$, $\mathfrak{q}(\mathbf{x}_t)$ can be obtained from them
- Naming conventions differ:
  - **OC**: minimization, cost, state $\mathbf{x}$, control $\mathbf{u}$, policy μ
  - **RL**: maximization, reward, state $\mathbf{s}$, action $\mathbf{a}$, policy π
  - **ECE276B**: minimization, cost, state $\mathbf{x}$, control $\mathbf{u}$, policy π

Example: Inventory Control

- Consider keeping an item stocked in a warehouse:
  - If there is too little, we may run out (not preferred)
  - If there is too much, the storage cost will be high (not preferred)
- Model:
  - $x_t \in \mathbb{R}$: available stock at the beginning of time period t
  - $u_t \in \mathbb{R}_{\geq 0}$: stock ordered and immediately delivered at the beginning of time period t (supply)
  - w_t: random demand during time period t with known pdf. Assume excess demand is back-logged, i.e., corresponds to negative stock x_t.
  - **Motion model**: $x_{t+1} = f(x_t, u_t, w_t) := x_t + u_t - w_t$
  - **Cost function**: $\mathbb{E}\left[q(x_T) + \sum_{t=0}^{T-1} \left(r(x_t) + cu_t - pw_t\right)\right]$ where
    - pw_t: revenue
    - cu_t: cost of items
    - $r(x_t)$: penalizes too much stock or negative stock
    - $q(x_T)$: remaining items we cannot sell or demand that we cannot meet

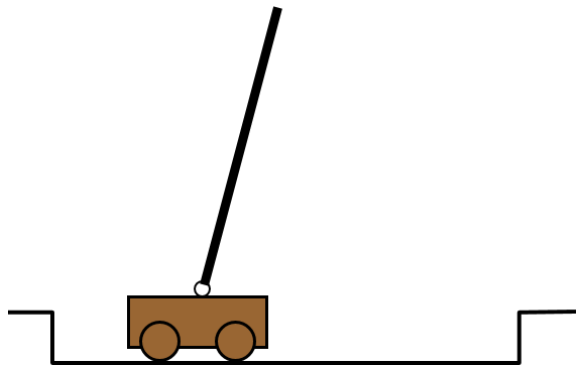
Example: Rubik's Cube

- Invented in 1974 by Ernõ Rubik
- Model:
  - State space size: $\sim 4.33 \times 10^{19}$
  - Control space size: 12
  - Cost: 1 for each time step
  - Deterministic, fully observable
- The cube can be solved in 20 or fewer moves



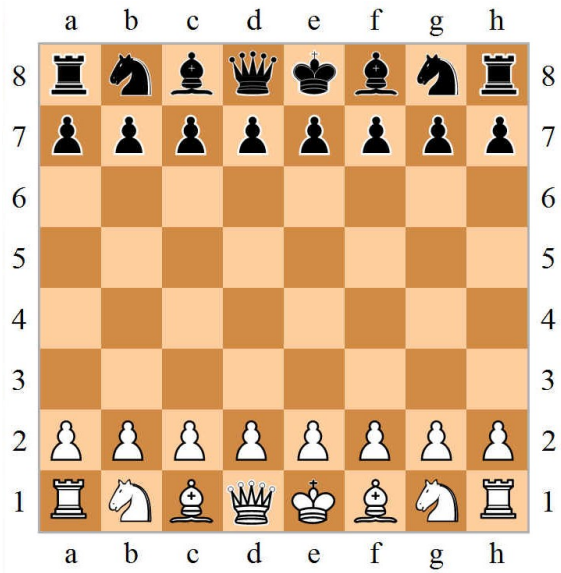
Example: Cart-Pole Problem

- Move a cart left, right to keep a pole balanced
- Model:
  - State space: 4-D continuous $(x, \dot{x}, \theta, \dot{\theta})$
  - Control space: $\{-N, N\}$
  - Cost:
    - 0 when in the goal region
    - 1 when outside the goal region
    - 100 when outside the feasible region
  - Deterministic, fully observable



Example: Chess

- Model:
  - State space size: $\sim 10^{47}$
  - Control space size: from 0 to 218
  - Cost: 0 each step, $\{-1, 0, 1\}$ at the end of the game
  - Deterministic, opponent-dependent state transitions (can be modeled as a game)
- The game tree size (all possible policies) is 10^{123}



Example: Grid World Navigation

- Navigate to a goal without crashing into obstacles
- Model:
  - State space: 2-D robot position
  - Control space: $\mathcal{U} = \{left, right, up, down\}$
  - Cost: 1 until the goal is reached, ∞ if an obstacles is hit
  - Can be deterministic or stochastic; fully or partially observable