

# Introduction to Robotics

Marc Toussaint

September 3, 2016

*This is a direct concatenation and reformatting of all lecture slides and exercises from the Robotics course (winter term 2015/16, U Stuttgart), including a bullet point list to help prepare for exams.*

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Basic 3D Geometry</b>	<b>7</b>
Mobile robotics vs. Manipulation vs. Kinematic/Dynamic motion control (2:1)		
2.1	Basic 3D geometry & notation . . . . .	7
Coordinate frames and transforms (2:5) Homogeneous transformation (2:10) Composition of transforms (2:12)		
<b>3</b>	<b>Probabilities</b>	<b>9</b>
Probabilities as (subjective) information calculus (3:1) Frequentist vs Bayesian (3:3)		
3.1	Basic definitions . . . . .	9
Definitions based on sets (3:5) Random variables (3:6) Probability distribution (3:7) Joint distribution (3:8) Marginal (3:8) Conditional distribution (3:8) Bayes' Theorem (3:9) Multiple RVs, conditional independence (3:10)		
3.2	Distributions . . . . .	10
Gaussian (3:14) Dirac (3:16) Particle approximation of a distribution (3:17)		
3.3	Probability distributions . . . . .	11
Bernoulli and Binomial distributions (3:23) Beta (3:24) Multinomial (3:27) Dirichlet (3:28) Conjugate priors (3:32) Dirac distribution (3:35) Gaussian (3:36) Particle approximation of a distribution (3:39) Student's t, Exponential, Laplace, Chi-squared, Gamma distributions (3:42)		
<b>4</b>	<b>Mobile Robotics</b>	<b>16</b>
Darpa challenge (4:2)		
4.1	PART I: State Estimation . . . . .	17
Applying Bayes' rule to state estimation (4:14) Bayes Filter (4:15) Histogram Filter = Bayes Filter with Bins (4:19) Particle Filter = Bayes Filter with Particles (4:36) Kalman Filter = Bayes Filter with Gaussian (4:40) Kalman filter equations (4:42) Extended Kalman Filter (4:47) Unscented Transform (4:49)		
4.2	PART II: Simultaneous Localization and Mapping (SLAM) . . . . .	24
Definition of a map, definition of SLAM problem (4:57) Joint Bayes Filter over state and map (Kalman SLAM) (4:59) Particle SLAM (4:62) Particles carry their own history, their own map (4:62) Graph-based SLAM (4:68)		

<b>5 Kinematics</b>	<b>28</b>
5.1 Kinematics . . . . .	28
Composition of transforms (5:5) Forward kinematics (5:6) Joint types (5:7) Kinematic map (5:9) Jacobian (5:11)	
5.2 Inverse Kinematics . . . . .	30
Inverse kinematics as optimization problem (5:17) Inverse kinematics solution (5:20) Singularity (5:26) Null space, task space, operational space, joint space (5:27) Motion rate control (5:28) Sine motion profile (5:30) Joint space trajectory interpolation (5:32) Task space trajectory interpolation (5:33)	
5.3 Multiple tasks . . . . .	33
The 'big' task vector (5:38) Inverse kinematics for all tasks (5:39) Hierarchical inverse kinematics, nullspace motion (5:41) Reference of task maps and their Jacobians (5:43)	
<b>6 Dynamics</b>	<b>36</b>
Dynamic vs. non-dynamic robots (6:1) Definition of a dynamics equation (6:3)	
6.1 PID and a 1D point mass . . . . .	36
1D point mass (6:6) P gain (6:7) D gain (6:10) oscillatory-damped, critically damped, over-damped (6:13) Damping ratio, wave length (6:14) I gain (6:15) PID control (6:16)	
6.2 Dynamics of mechanical systems . . . . .	38
Euler-Lagrange equation (6:20) Pendulum example for Euler-Lagrange (6:21) General structure of Euler-Lagrange (6:22) Newton-Euler recursion (6:23)	
6.3 Controlling a dynamic robot . . . . .	39
General structure of robot dynamics (6:28) Inverse and forward dynamics (6:29) Following a reference trajectory in joint space (6:30) Following a reference trajectory in task space (6:31) Operational space control (6:32) Multiple tasks (6:35)	
<b>7 Path Planning</b>	<b>42</b>
Path finding examples (7:1) Feedback control vs path finding vs trajectory optimization (7:5)	
7.1 Background . . . . .	42
Bug algorithms (7:8) Potential fields (7:13) Dijkstra (7:18)	
7.2 Sample-based Path Finding . . . . .	45
Probabilistic Road Maps (PRMs) (7:28) Planning/testing local connections (7:32) Probabilistic completeness of PRMs (7:34) Other PRM sampling strategies (7:35) Rapidly Exploring Random Trees (RRTs) (7:39) Goal-oriented RRT (7:40) Bi-directional RRT (7:42)	
7.3 Non-holonomic systems . . . . .	49
Definition of non-holonomic (7:47) Path finding for a non-holonomic system (7:50) Control-based sampling (7:52) RRTs with differential constraints (7:58) Configuration state metrics (7:59) Side story: Dubins curves (7:60)	
<b>8 Path Optimization – briefly</b>	<b>53</b>
From inverse kinematics to path costs (8:2) General $k$ -order cost terms (8:3) Choice of optimizer (8:5)	
<b>9 Reinforcement Learning in Robotics – briefly</b>	<b>54</b>
Markov Decision Process (9:4) Relation to optimal control (9:5) Five approaches to RL (9:7) Imitation Learning (9:9) Inverse RL (9:12) Policy Search with Policy Gradients (9:16) Policy Search with Black-Box Optimization (9:21)	
<b>10 Control Theory</b>	<b>60</b>
Topics in control theory (10:3)	
10.1 Optimal control . . . . .	60
Discrete time finite horizon definition (10:7) Bellman equation (discrete time) (10:9) Continuous time finite horizon definition (10:10) Hamilton-Jacobi-Bellman equation (10:11) Infinite horizon case (10:12) Linear-Quadratic Optimal Control (10:15) Riccati differential eq = HJB eq in LQ case (10:17) Algebraic Riccati equation (ARE) for infinite horizon (10:19) Riccati equations (also discrete time) (10:19)	
10.2 Controllability . . . . .	63
Definition of controllability (10:27)	
10.3 Stability . . . . .	64
Definition of closed loop system equation (10:33) Lyapunov and exponential stability (10:35) Linear stability analysis using eigenvalues (10:36) Definition of Lyapunov function (10:40) Energy as Lyapunov function (10:42) Value as Lyapunov function (10:43)	

<b>11 Exercises</b>	<b>67</b>
11.1 Exercise 1 . . . . .	67
11.2 Exercise 2 . . . . .	67
11.3 Exercise 3 . . . . .	68
11.4 Exercise 4 . . . . .	69
11.5 Exercise 5 . . . . .	70
11.6 Exercise 6 . . . . .	71
11.7 Exercise 7 . . . . .	72
11.8 Exercise 8 . . . . .	72
11.9 Exercise 09 . . . . .	73
11.10 Exercise 11 . . . . .	74
11.11 Exercise 11 . . . . .	75
<b>12 Bullet points to help learning</b>	<b>76</b>
12.1 Geometry . . . . .	76
12.2 Probability . . . . .	76
12.3 Mobile Robotics . . . . .	76
12.4 Kinematics . . . . .	76
12.5 Dynamics . . . . .	77
12.6 Path planning . . . . .	77
12.7 Reinforcement Learning . . . . .	77
12.8 Control Theory . . . . .	77



## 1 Introduction

### Instructors

Peter Englert

Vien Ngo

Stefan Otte

1:1

*Why Robotics?*

1:2



1:3



1:4

### Why Robotics?

- Commercial:

Industrial, health care, entertainment, agriculture, surgery, etc

- Critical view:

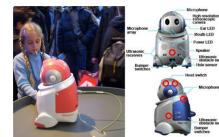
– International Committee for Robot Arms Control

<http://icrac.net/>

– Campaign to Stop Killer Robots

<http://www.stopkillerrobots.org/>

– Noel Sharkey's articles on robot ethics (Child care robots PePeRo.)



<http://www.nec.co.jp/products/robot/en/>

1:5

### Robotics as intelligence research

*AI in the real world*

AI: Machine Learning, probabilistic reasoning, optimization

Real World: Interaction, manipulation, perception, navigation, etc

1:6

### Why AI needs to go real world

- Motion was the driving force to develop intelligence
  - motion needs control & decision making ↔ fast information processing
  - motion needs anticipation & planning

- motion needs perception
  - motion needs spatial representations
- **Manipulation** requires to acknowledge the structure (geometry, physics, objects) of the real world. Classical AI does not

1:7

## Robotics as intelligence research

- Machine Learning and AI are **computational** disciplines, which had great success with statistical modelling, analysis of data sets, symbolic reasoning. But they have not solved *autonomous learning, acting & reasoning in real worlds*.
- Neurosciences and psychology are **descriptive** sciences, either on the biological or cognitive level, e.g. with great successes to describe and cure certain diseases. But they are not sufficient to create intelligent systems.
- Robotics is the only **synthetic** discipline to understand intelligent behavior in natural worlds. Robotics tells us what the actual problems are when trying to organize behavior in natural worlds.

1:8

## History

- little movie...

(<http://www.csail.mit.edu/videoarchive/history/aifilms>   <http://www.ai.sri.com/shakey/>)

1:9

## Four chapters

### • Kinematics & Dynamics

*goal: orchestrate joint movements for desired movement in task spaces*

Kinematic map, Jacobian, optimality principle of inverse kinematics, singularities, configuration/operational/null space, multiple simultaneous tasks, special task variables, trajectory interpolation, motion profiles; 1D point mass, damping & oscillation, PID, general dynamic systems, Newton-Euler, joint space control, reference trajectory following, optimal operational space control

### • Planning & optimization

*goal: planning around obstacles, optimizing trajectories*

Path finding vs. trajectory optimization, local vs. global, Dijkstra, Probabilistic Roadmaps, Rapidly Exploring Random Trees, differential constraints, metrics; trajectory optimization, general cost function, task variables, transition costs, gradient methods, 2nd order methods, Dynamic Programming

### • Control Theory

*theory on designing optimal controllers*

Topics in control theory, optimal control, HJB equation, infinite horizon case, Linear-Quadratic optimal control, Riccati equations (differential, algebraic, discrete-time), controllability, stability, eigenvalue analysis, Lyapunov function

### • Mobile robots

*goal: localize and map yourself*

State estimation, Bayes filter, odometry, particle filter, Kalman filter, Bayes smoothing, SLAM, joint Bayes filter, EKF SLAM, particle SLAM, graph-based SLAM

1:10

## Prerequisites

- Is this a practical or theoretical course?

*"There is nothing more practical than a good theory."*  
(Vapnik, others...)

- Essentially, the whole course is about

*reducing real-world problems to mathematical problems*

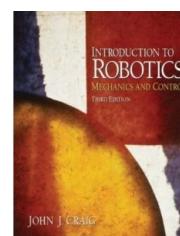
that can be solved efficiently

- Required: Basics in
  - Linear Algebra
  - Probability theory
  - Optimization

1:11

## Books

There is no reference book for this lecture. But a basic well-known standard text book is:

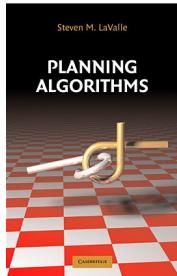


Craig, J.J.: *Introduction to robotics: mechanics and control*. Addison-Wesley New York, 1989. (3rd edition 2006)

1:12

## Books

An advanced text book on planning is this:



Steven M. LaValle: *Planning Algorithms*. Cambridge University Press, 2006.  
**online:**  
<http://planning.cs.uiuc.edu/>

1:13

## Online resources

- VideoLecture by Oussama Khatib: [http://videolectures.net/oussama\\_khatib/](http://videolectures.net/oussama_khatib/)  
(focus on kinematics, dynamics, control)
- Oliver Brock's lecture <http://www.robotics.tu-berlin.de/menue/teaching/>
- Stefan Schaal's lecture Introduction to Robotics: <http://www-clmc.usc.edu/Teaching/TeachingIntroductionToRoboticsSyllabus>  
(focus on control, useful: Basic Linear Control Theory (analytic solution to simple dynamic model → PID), chapter on dynamics)
- Chris Atkeson's "Kinematics, Dynamic Systems, and Control" <http://www.cs.cmu.edu/~cga/kdc/>  
(uses Schaal's slides and LaValle's book, useful: slides on 3d kinematics <http://www.cs.cmu.edu/~cga/kdc/ewhitman1.pptx>)
- CMU lecture "introduction to robotics" <http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/s16/main/index.html>  
(useful: PID control, simple BUGs algorithms for motion planning, non-holonomic constraints)
- *Handbook of Robotics* (partially online at Google books) <http://tiny.cc/u6tzl>
- LaValle's *Planning Algorithms* <http://planning.cs.uiuc.edu/>

1:14

## Organization

- Course Webpage:

<https://ipvs.informatik.uni-stuttgart.de/mlr/teaching/robotics>

- Slides, Exercises & Software (C++)

- Admin things, please first ask:

Carola Stahl, Carola.Stahl@ipvs.uni-stuttgart.de, Raum 2.217

- Tutorials: Wednesdays, 15:45-17:15 & 17:30-19:00 (0.108)

First exercise will be published today.

No tutorials in the first week.

- Rules for the tutorials:

- Doing the exercises is crucial!
- At the beginning of each tutorial:
  - sign into a list
  - mark which exercises you have (successfully) worked on
- Students are randomly selected to present their solutions
- You need 50% of completed exercises to be allowed to the exam
- Please check 2 weeks before the end of the term, if you can take the exam

1:15

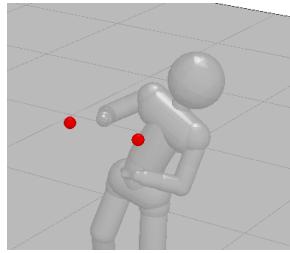
## 2 Basic 3D Geometry

- Two “types of robotics”:
  - Mobile robotics – is all about localization & mapping
  - Manipulation – is all about interacting with the world
  - Kinematic/Dynamic Motion Control: same as 2) without ever making it to interaction..
- Typical manipulation robots (and animals) are kinematic trees  
Their pose/state is described by all joint angles

2:1

### Basic motion generation problem

- Move all joints in a coordinated way so that the endeffector makes a desired movement



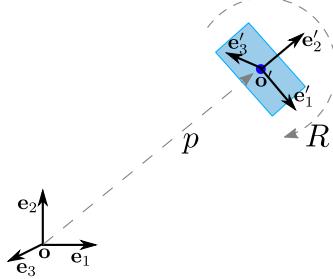
01-kinematics: ./x.exe -mode 2/3/4

2:2

### 2.1 Basic 3D geometry & notation

2:3

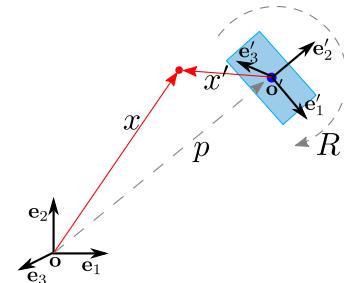
#### Pose (position & orientation)



- A *pose* is described by a translation  $p \in \mathbb{R}^3$  and a rotation  $R \in SO(3)$ 
  - $R$  is an *orthonormal* matrix (orthogonal vectors stay orthogonal, unit vectors stay unit)
  - $R^{-1} = R^T$
  - columns and rows are orthogonal unit vectors
  - $\det(R) = 1$
  - $R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}$

2:4

### Frame and coordinate transforms



- Let  $(o, e_{1:3})$  be the world frame,  $(o', e'_{1:3})$  be the body's frame. The new basis vectors are the *columns* in  $R$ , that is,  $e'_1 = R_{11}e_1 + R_{21}e_2 + R_{31}e_3$ , etc,
- $x$  = coordinates in world frame  $(o, e_{1:3})$
- $x'$  = coordinates in body frame  $(o', e'_{1:3})$
- $p$  = coordinates of  $o'$  in world frame  $(o, e_{1:3})$

$$x = p + Rx'$$

2:5

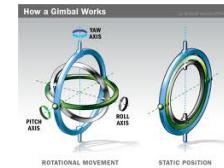
### Briefly: Alternative Rotation Representations

(See the “geometry notes” for more details!)

2:6

#### Euler angles

- Describe rotation by consecutive rotation about different axis:
  - 3-1-3 or 3-1-2 conventions, yaw-pitch-roll (3-2-1) in air flight
  - first rotate  $\phi$  about  $e_3$ , then  $\theta$  about the new  $e'_1$ , then  $\psi$  about the new  $e''_3$
- Gimbal Lock



- Euler angles have severe problem:
  - if two axes align: blocks 1 DoF of rotation!!
  - “singularity” of Euler angles
  - Example: 3-1-3 and second rotation 0 or  $\pi$

2:7

#### Rotation vector

- vector  $w \in \mathbb{R}^3$ 
  - length  $|w| = \theta$  is rotation angle (in radians)
  - direction of  $w$  = rotation axis ( $\underline{w} = w/\theta$ )
- Application on a vector  $v$  (Rodrigues' formula):

$$w \cdot v = \cos \theta v + \sin \theta (\underline{w} \times v) + (1 - \cos \theta) \underline{w}(\underline{w}^\top v)$$

- Conversion to matrix:

$$R(w) = \exp(\hat{w})$$

$$= \cos \theta I + \sin \theta \hat{w} \theta + (1 - \cos \theta) w w^\top / \theta^2$$

$$\hat{w} := \begin{pmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{pmatrix}$$

( $\hat{w}$  is called skew matrix, with property  $\hat{w}v = w \times v$ ;  $\exp(\cdot)$  is called exponential map)

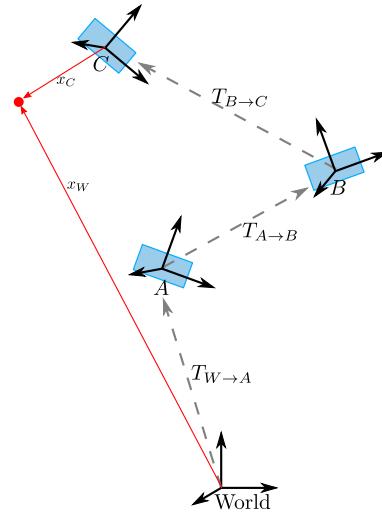
- Composition: convert to matrix first
- Drawback: singularity for small rotations

2:8

- $T_{A \rightarrow B}$  describes the coordinate transformation from  $x^B$  to  $x^A$ . That is, it describes the backward COORDINATE transformation
- Confused? Vectors (and frames) transform *covariant*, coordinates *contra-variant*. See “geometry notes” or Wikipedia for more details, if you like.

2:11

## Composition of transforms



## Quaternion

- A quaternion is  $r \in \mathbb{R}^4$  with unit length  $|r| = r_0^2 + r_1^2 + r_2^2 + r_3^2 = 1$

$$r = \begin{pmatrix} r_0 \\ \bar{r} \end{pmatrix}, \quad r_0 = \cos(\theta/2), \quad \bar{r} = \sin(\theta/2) \underline{w}$$

where  $\underline{w}$  is the unit length rotation axis

- Conversion to matrix

$$R(r) = \begin{pmatrix} 1 - r_{22} - r_{33} & r_{12} - r_{03} & r_{13} + r_{02} \\ r_{12} + r_{03} & 1 - r_{11} - r_{33} & r_{23} - r_{01} \\ r_{13} - r_{02} & r_{23} + r_{01} & 1 - r_{11} - r_{22} \end{pmatrix}$$

$$r_{ij} = 2r_i r_j, \quad r_0 = \frac{1}{2} \sqrt{1 + \text{tr } R}$$

$$r_3 = (R_{21} - R_{12})/(4r_0), \quad r_2 = (R_{13} - R_{31})/(4r_0), \quad r_1 = (R_{32} - R_{23})/(4r_0)$$

- Composition

$$r \circ r' = \begin{pmatrix} r_0 r'_0 - \bar{r}^\top \bar{r}' \\ r_0 \bar{r}' + r'_0 \bar{r} + \bar{r}' \times \bar{r} \end{pmatrix}$$

- Application to vector  $v$ : convert to matrix first
- Benefits: fast composition. No sin/cos computations. **Use this!**

2:9

$$T_{W \rightarrow C} = T_{W \rightarrow A} T_{A \rightarrow B} T_{B \rightarrow C}$$

$$x^W = T_{W \rightarrow A} T_{A \rightarrow B} T_{B \rightarrow C} x^C$$

2:12

## Homogeneous transformations

- $x^A$  = coordinates of a point in frame  $A$

$x^B$  = coordinates of a point in frame  $B$

- Translation and rotation:  $x^A = t + Rx^B$

- Homogeneous transform  $T \in \mathbb{R}^{4 \times 4}$ :

$$T_{A \rightarrow B} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

$$x^A = T_{A \rightarrow B} x^B = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x^B \\ 1 \end{pmatrix} = \begin{pmatrix} Rx^B + t \\ 1 \end{pmatrix}$$

in homogeneous coordinates, we append a 1 to all coordinate vectors

2:10

## Is $T_{A \rightarrow B}$ forward or backward?

- $T_{A \rightarrow B}$  describes the translation and rotation of frame  $B$  relative to  $A$

That is, it describes the forward FRAME transformation (from  $A$  to  $B$ )

### 3 Probabilities

#### Probability Theory

- Why do we need probabilities?
  - Obvious: to express inherent (*objective*) stochasticity of the world
- But beyond this: (also in a “deterministic world”):
  - lack of knowledge!
  - hidden (latent) variables
  - expressing *uncertainty*
  - expressing *information* (and lack of information)
  - **Subjective Probability**
- Probability Theory: an information calculus

3:1

#### Outline

- Basic definitions
  - Random variables
  - joint, conditional, marginal distribution
  - Bayes’ theorem
- Probability distributions:
  - Gauss
  - Dirak & Particles

3:2

#### Probability: Frequentist and Bayesian

- Frequentist probabilities are defined in the limit of an infinite number of trials  
*Example:* “The probability of a particular coin landing heads up is 0.43”
- Bayesian (subjective) probabilities quantify degrees of belief  
*Example:* “The probability of it raining tomorrow is 0.3”
  - Not possible to repeat “tomorrow”

3:3

#### 3.1 Basic definitions

3:4

#### Probabilities & Sets

- Sample Space/domain  $\Omega$ , e.g.  $\Omega = \{1, 2, 3, 4, 5, 6\}$
- Probability  $P : A \subset \Omega \mapsto [0, 1]$   
e.g.,  $P(\{1\}) = \frac{1}{6}$ ,  $P(\{4\}) = \frac{1}{6}$ ,  $P(\{2, 5\}) = \frac{1}{3}$ ,

- Axioms:  $\forall A, B \subseteq \Omega$ 
  - Nonnegativity  $P(A) \geq 0$
  - Additivity  $P(A \cup B) = P(A) + P(B)$  if  $A \cap B = \emptyset$
  - Normalization  $P(\Omega) = 1$

#### Implications

$$\begin{aligned} 0 &\leq P(A) \leq 1 \\ P(\emptyset) &= 0 \\ A \subseteq B &\Rightarrow P(A) \leq P(B) \\ P(A \cup B) &= P(A) + P(B) - P(A \cap B) \\ P(\Omega \setminus A) &= 1 - P(A) \end{aligned}$$

3:5

#### Probabilities & Random Variables

- For a random variable  $X$  with discrete domain  $\text{dom}(X) = \Omega$  we write:
 
$$\begin{aligned} \forall x \in \Omega : 0 &\leq P(X=x) \leq 1 \\ \sum_{x \in \Omega} P(X=x) &= 1 \end{aligned}$$

Example: A dice can take values  $\Omega = \{1, \dots, 6\}$ .  
 $X$  is the random variable of a dice throw.  
 $P(X=1) \in [0, 1]$  is the probability that  $X$  takes value 1.
- A bit more formally: a random variable is a map from a measurable space to a domain (sample space) and thereby introduces a probability measure on the domain (“assigns a probability to each possible value”)

3:6

#### Probability Distributions

- $P(X=1) \in \mathbb{R}$  denotes a specific probability  
 $P(X)$  denotes the probability distribution (function over  $\Omega$ )

Example: A dice can take values  $\Omega = \{1, 2, 3, 4, 5, 6\}$ .  
By  $P(X)$  we describe the full distribution over possible values  $\{1, \dots, 6\}$ .  
These are 6 numbers that sum to one, usually stored in a *table*, e.g.:  
 $[\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}]$

- In implementations we typically represent distributions over discrete random variables as tables (arrays) of numbers

#### Notation for summing over a RV:

In equation we often need to sum over RVs. We then write

$\sum_X P(X) \dots$   
as shorthand for the explicit notation  $\sum_{x \in \text{dom}(X)} P(X=x) \dots$

3:7

#### Know by heart!

Assume we have two random variables  $X$  and  $Y$

				$P(X=x, Y=y)$
$x$				
				$P_{xy}$

- Joint (product rule)

$$P(X, Y) = P(X) P(Y|X) = P(Y) P(X|Y)$$

- Marginal (sum rule)

$$P(X) = \sum_Y P(X, Y)$$

- Conditional:

$$P(X|Y) = \frac{P(X, Y)}{P(Y)}$$

The conditional is normalized:  $\forall Y : \sum_X P(X|Y) = 1$

- $X$  is *independent* of  $Y$  iff:  $P(X|Y) = P(X)$

(table thinking: all columns of  $P(X|Y)$  are equal)

3:8

## Know by heart!

- Implications of these definitions:

*Product rule:*  $P(X, Y) = P(X) P(Y|X) = P(Y) P(X|Y)$

$$P(X|Y) = \frac{P(Y|X) P(X)}{P(Y)}$$

$$P(H|E) = \frac{P(E|H) P(H)}{P(E)}$$

$$\text{posterior} = \frac{\text{likelihood} \cdot \text{prior}}{\text{normalization}}$$

3:9

## Multiple RVs:

- Analogously for  $n$  random variables  $X_{1:n}$  (stored as a rank  $n$  tensor)

*Joint:*  $P(X_{1:n})$

*Marginal:*  $P(X_1) = \sum_{X_{2:n}} P(X_{1:n})$ ,

*Conditional:*  $P(X_1|X_{2:n}) = \frac{P(X_{1:n})}{P(X_{2:n})}$

- $X$  is *conditionally independent* of  $Y$  given  $Z$  iff:

$$P(X|Y, Z) = P(X|Z)$$

- Product rule and Bayes' Theorem:

$$\begin{aligned} P(X_{1:n}) &= \prod_{i=1}^n P(X_i|X_{i+1:n}) & P(X, Z, Y) &= \\ && P(X|Y, Z) P(Y|Z) P(Z) &= \\ &\frac{P(X_1|X_{2:n})}{P(X_2|X_1, X_{3:n})} \frac{P(X_1|X_{3:n})}{P(X_2|X_{3:n})} & P(X|Y, Z) &= \frac{P(Y|X, Z) P(X|Z)}{P(Y|Z)} \\ && P(X, Y|Z) &= \frac{P(X, Z|Y) P(Y)}{P(Z)} \end{aligned}$$

3:10

## Discrete and continuous domains

- Let  $x$  be a continuous RV. The **probability density function (pdf)**  $p(x) \in [0, \infty)$  defines the probability

$$P(a \leq x \leq b) = \int_a^b p(x) dx \in [0, 1]$$

- In discrete domains: *probability mass function (PMF)*

- The **cumulative distribution function (cdf)**  $F(y) = P(x \leq y) = \int_{-\infty}^y dx p(x) \in [0, 1]$  is the cumulative integral with  $\lim_{y \rightarrow \infty} F(y) = 1$

3:11

## 3.2 Distributions

3:12

### Motivation for Gaussian distributions

- Gaussian Bandits
- Control theory, Stochastic Optimal Control
- State estimation, sensor processing, Gaussian filtering (Kalman filtering)
- Machine Learning
- etc
- rich vocabulary and easy to compute!

3:13

### Gaussian distribution

- 1-dim:  $\mathcal{N}(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}$

- $n$ -dim Gaussian in *normal form*:

$$\mathcal{N}(x | \mu, \Sigma) = \frac{1}{\sqrt{|2\pi\Sigma|^{1/2}}} \exp\left\{-\frac{1}{2}(x - \mu)^\top \Sigma^{-1} (x - \mu)\right\}$$

with **mean**  $\mu$  and **covariance** matrix  $\Sigma$ . In *canonical form*:

$$\mathcal{N}[x | a, A] = \frac{\exp\left\{-\frac{1}{2}a^\top A^{-1}a\right\}}{\sqrt{|2\pi A^{-1}|^{1/2}}} \exp\left\{-\frac{1}{2}x^\top A x + x^\top a\right\} \quad (1)$$

with **precision** matrix  $A = \Sigma^{-1}$  and coefficient  $a = \Sigma^{-1}\mu$  (and mean  $\mu = A^{-1}a$ ).

- Gaussian identities: see <http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gaussians.pdf>

3:14

## Gaussian identities

Symmetry:  $\mathcal{N}(x | a, A) = \mathcal{N}(a | x, A) = \mathcal{N}(x - a | 0, A)$

Product:

$$\begin{aligned}\mathcal{N}(x | a, A) \mathcal{N}(x | b, B) &= \mathcal{N}[x | A^{-1}a + B^{-1}b, A^{-1} + B^{-1}] \mathcal{N}(a | b, A + B) \\ \mathcal{N}[x | a, A] \mathcal{N}[x | b, B] &= \mathcal{N}[x | a + b, A + B] \mathcal{N}(A^{-1}a | B^{-1}b, A^{-1} + B^{-1})\end{aligned}$$

"Propagation":

$$\int_y \mathcal{N}(x | a + Fy, A) \mathcal{N}(y | b, B) dy = \mathcal{N}(x | a + Fb, A + FB^T)$$

Transformation:

$$\mathcal{N}(Fx + f | a, A) = \frac{1}{|F|} \mathcal{N}(x | F^{-1}(a - f), F^{-1}AF^T)$$

Marginal & conditional:

$$\mathcal{N}\left(\begin{array}{c} x \\ y \end{array} \middle| \begin{array}{cc} a & C \\ b & B^T \\ C^T & B \end{array}\right) = \mathcal{N}(x | a, A) \cdot \mathcal{N}(y | b + C^T A^{-1}(x - a), B - C^T A^{-1}C)$$

More Gaussian identities: see <http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gaussians.pdf>

3:15

## Dirac Delta / Point Particle

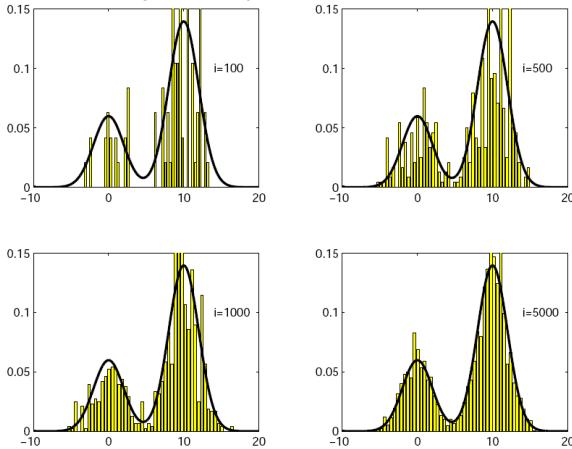
**Dirac or  $\delta$**  ("point particle"):  $\delta(x) = 0$  except at  $x = 0$ ,  $\int \delta(x) dx = 1$

$\delta(x) = \frac{\partial}{\partial x} H(x)$  where  $H(x) = [x \geq 0]$  is the Heavyside step function

3:16

## Particle Approximation of a Distribution

We approximate a distribution  $p(x)$  over a continuous domain  $\mathbb{R}^n$  with a histogram of  $N$  particles:



3:17

## Particle Approximation of a Distribution

We approximate a distribution  $p(x)$  over a continuous domain  $\mathbb{R}^n$

- A particle distribution  $q(x)$  is a weighed set  $S = \{(x^i, w^i)\}_{i=1}^N$  of  $N$  particles

- each particle has a "location"  $x^i \in \mathbb{R}^n$  and a weight  $w^i \in \mathbb{R}$
- weights are normalized,  $\sum_i w^i = 1$

$$q(x) := \sum_{i=1}^N w^i \delta(x - x^i)$$

where  $\delta(x - x^i)$  is the  $\delta$ -distribution.

- Given weighted particles, we can estimate for any (smooth)  $f$ :

$$\langle f(x) \rangle_p = \int_x f(x)p(x)dx \approx \sum_{i=1}^N w^i f(x^i)$$

See *An Introduction to MCMC for Machine Learning* [www.cs.ubc.ca/~nando/papers/mlintro.pdf](http://www.cs.ubc.ca/~nando/papers/mlintro.pdf)

3:18

## Motivation for particle distributions

- Numeric representation of "difficult" distributions
  - Very general and versatile
  - But often needs many samples
- Distributions over games (action sequences), sample based planning, MCTS
- State estimation, particle filters
- etc

3:19

## Recap

sum rule

product rule

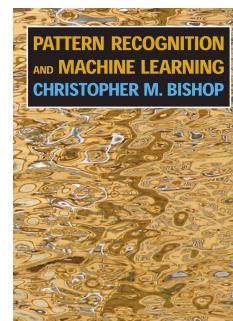
bayes theorem

normal distribution

particle approximation

3:20

## 3.3 Probability distributions



Bishop, C. M.: *Pattern Recognition and Machine Learning*. Springer, 2006  
<http://research.microsoft.com/en-us/um/people/cmbishop/prml/>

3:21

- We skip Bernoulli & Beta, Multinomial & Dirichlet..

3:22

## Bernoulli & Binomial

- We have a binary random variable  $x \in \{0, 1\}$  (i.e.  $\text{dom}(x) = \{0, 1\}$ )

The *Bernoulli* distribution is parameterized by a single scalar  $\mu$ ,

$$P(x=1 | \mu) = \mu, \quad P(x=0 | \mu) = 1 - \mu$$

$$\text{Bern}(x | \mu) = \mu^x (1 - \mu)^{1-x}$$

- We have a data set of random variables  $D = \{x_1, \dots, x_n\}$ , each  $x_i \in \{0, 1\}$ . If each  $x_i \sim \text{Bern}(x_i | \mu)$  we have

$$P(D | \mu) = \prod_{i=1}^n \text{Bern}(x_i | \mu) = \prod_{i=1}^n \mu^{x_i} (1 - \mu)^{1-x_i}$$

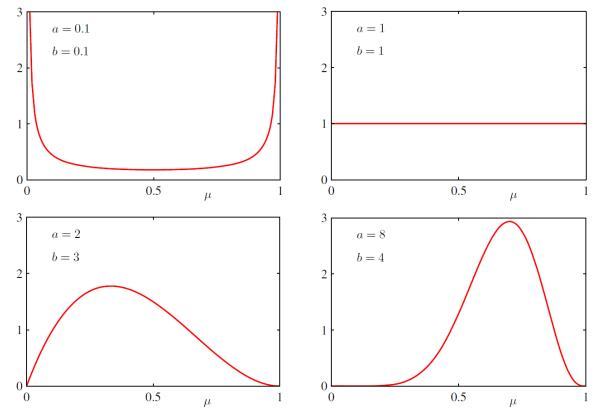
$$\underset{\mu}{\operatorname{argmax}} \log P(D | \mu) = \underset{\mu}{\operatorname{argmax}} \sum_{i=1}^n x_i \log \mu + (1 - x_i) \log(1 - \mu) = \frac{1}{n} \sum_{i=1}^n x_i$$

- The *Binomial distribution* is the distribution over the count  $m = \sum_{i=1}^n x_i$

$$\text{Bin}(m | n, \mu) = \binom{n}{m} \mu^m (1 - \mu)^{n-m}, \quad \binom{n}{m} = \frac{n!}{(n-m)! m!}$$

3:23

## Beta



from Bishop

3:26

## Beta

### How to express uncertainty over a Bernoulli parameter $\mu$

- The *Beta distribution* is over the interval  $[0, 1]$ , typically the parameter  $\mu$  of a Bernoulli:

$$\text{Beta}(\mu | a, b) = \frac{1}{B(a, b)} \mu^{a-1} (1 - \mu)^{b-1}$$

with mean  $\langle \mu \rangle = \frac{a}{a+b}$  and mode  $\mu^* = \frac{a-1}{a+b-2}$  for  $a, b > 1$

- The crucial point is:

- Assume we are in a world with a “Bernoulli source” (e.g., binary bandit), but don’t know its parameter  $\mu$
- Assume we have a *prior* distribution  $P(\mu) = \text{Beta}(\mu | a, b)$
- Assume we collected some data  $D = \{x_1, \dots, x_n\}$ ,  $x_i \in \{0, 1\}$ , with counts  $a_D = \sum_i x_i$  of  $[x_i = 1]$  and  $b_D = \sum_i (1 - x_i)$  of  $[x_i = 0]$
- The posterior is

$$\begin{aligned} P(\mu | D) &= \frac{P(D | \mu)}{P(D)} P(\mu) \propto \text{Bin}(D | \mu) \text{Beta}(\mu | a, b) \\ &\propto \mu^{a_D} (1 - \mu)^{b_D} \mu^{a-1} (1 - \mu)^{b-1} = \mu^{a-1+a_D} (1 - \mu)^{b-1+b_D} \\ &= \text{Beta}(\mu | a + a_D, b + b_D) \end{aligned}$$

3:24

## Beta

The prior is  $\text{Beta}(\mu | a, b)$ , the posterior is  $\text{Beta}(\mu | a + a_D, b + b_D)$

- Conclusions:

- The semantics of  $a$  and  $b$  are counts of  $[x_i = 1]$  and  $[x_i = 0]$ , respectively
- The Beta distribution is conjugate to the Bernoulli (explained later)
- With the Beta distribution we can represent beliefs (state of knowledge) about uncertain  $\mu \in [0, 1]$  and know how to update this belief given data

3:25

## Multinomial

- We have an integer random variable  $x \in \{1, \dots, K\}$

The probability of a single  $x$  can be parameterized by  $\mu = (\mu_1, \dots, \mu_K)$

$$P(x=k | \mu) = \mu_k$$

with the constraint  $\sum_{k=1}^K \mu_k = 1$  (probabilities need to be normalized)

- We have a data set of random variables  $D = \{x_1, \dots, x_n\}$ , each  $x_i \in \{1, \dots, K\}$ . If each  $x_i \sim P(x_i | \mu)$  we have

$$P(D | \mu) = \prod_{i=1}^n \mu_{x_i} = \prod_{i=1}^n \prod_{k=1}^K \mu_k^{[x_i=k]} = \prod_{k=1}^K \mu_k^{m_k}$$

where  $m_k = \sum_{i=1}^n [x_i = k]$  is the count of  $[x_i = k]$ . The ML estimator is

$$\underset{\mu}{\operatorname{argmax}} \log P(D | \mu) = \frac{1}{n} (m_1, \dots, m_K)$$

- The *Multinomial distribution* is this distribution over the counts  $m_k$

$$\text{Mult}(m_1, \dots, m_K | n, \mu) \propto \prod_{k=1}^K \mu_k^{m_k}$$

3:27

## Dirichlet

### How to express uncertainty over a Multinomial parameter $\mu$

- The *Dirichlet distribution* is over the  $K$ -simplex, that is, over  $\mu_1, \dots, \mu_K \in [0, 1]$  subject to the constraint  $\sum_{k=1}^K \mu_k = 1$ :

$$\text{Dir}(\mu | \alpha) \propto \prod_{k=1}^K \mu_k^{\alpha_k - 1}$$

It is parameterized by  $\alpha = (\alpha_1, \dots, \alpha_K)$ , has mean  $\langle \mu_i \rangle = \frac{\alpha_i}{\sum_j \alpha_j}$  and mode  $\mu_i^* = \frac{\alpha_i - 1}{\sum_j \alpha_j - K}$  for  $\alpha_i > 1$ .

- The crucial point is:

- Assume we are in a world with a “Multinomial source” (e.g., an integer bandit), but don’t know its parameter  $\mu$
- Assume we have a *prior* distribution  $P(\mu) = \text{Dir}(\mu | \alpha)$

- Assume we collected some data  $D = \{x_1, \dots, x_n\}$ ,  $x_i \in \{1, \dots, K\}$ , with counts  $m_k = \sum_i [x_i = k]$
- The posterior is

$$\begin{aligned} P(\mu | D) &= \frac{P(D | \mu)}{P(D)} P(\mu) \propto \text{Mult}(D | \mu) \text{Dir}(\mu | a, b) \\ &\propto \prod_{k=1}^K \mu_k^{m_k} \prod_{k=1}^K \mu_k^{\alpha_k - 1} = \prod_{k=1}^K \mu_k^{\alpha_k - 1 + m_k} \\ &= \text{Dir}(\mu | \alpha + m) \end{aligned}$$

3:28

## Dirichlet

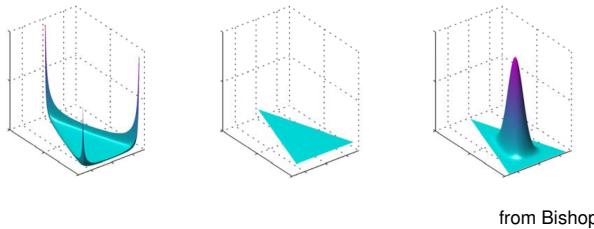
The prior is  $\text{Dir}(\mu | \alpha)$ , the posterior is  $\text{Dir}(\mu | \alpha + m)$

- Conclusions:
  - The semantics of  $\alpha$  is the counts of  $[x_i = k]$
  - The Dirichlet distribution is conjugate to the Multinomial
  - With the Dirichlet distribution we can represent beliefs (state of knowledge) about uncertain  $\mu$  of an integer random variable and know how to update this belief given data

3:29

## Dirichlet

Illustrations for  $\alpha = (0.1, 0.1, 0.1)$ ,  $\alpha = (1, 1, 1)$  and  $\alpha = (10, 10, 10)$ :



3:30

## Motivation for Beta & Dirichlet distributions

- Bandits:
  - If we have binary [integer] bandits, the Beta [Dirichlet] distribution is a way to represent and update beliefs
  - The belief space becomes discrete: The parameter  $\alpha$  of the prior is continuous, but the posterior updates live on a discrete "grid" (adding counts to  $\alpha$ )
  - We can in principle do belief planning using this
- Reinforcement Learning:
  - Assume we know that the world is a finite-state MDP, but do not know its transition probability  $P(s' | s, a)$ . For each  $(s, a)$ ,  $P(s' | s, a)$  is a distribution over the integer  $s'$
  - Having a separate Dirichlet distribution for each  $(s, a)$  is a way to represent our belief about the world, that is, our belief about  $P(s' | s, a)$
  - We can in principle do belief planning using this → Bayesian Reinforcement Learning
- Dirichlet distributions are also used to model texts (word distributions in text), images, or mixture distributions in general

3:31

## Conjugate priors

- Assume you have data  $D = \{x_1, \dots, x_n\}$  with likelihood

$$P(D | \theta)$$

that depends on an uncertain parameter  $\theta$

Assume you have a prior  $P(\theta)$

- The prior  $P(\theta)$  is **conjugate** to the likelihood  $P(D | \theta)$  iff the posterior

$$P(\theta | D) \propto P(D | \theta) P(\theta)$$

is in the *same distribution class* as the prior  $P(\theta)$

- Having a conjugate prior is very convenient, because then you know how to update the belief given data

3:32

## Conjugate priors

likelihood	conjugate
Binomial $\text{Bin}(D   \mu)$	Beta $\text{Beta}(\mu   a, b)$
Multinomial	Dirichlet $\text{Dir}(\mu   \alpha)$
$\text{Mult}(D   \mu)$	
Gauss $\mathcal{N}(x   \mu, \Sigma)$	Gauss $\mathcal{N}(\mu   \mu_0, A)$
1D $\mathcal{N}(x   \mu, \lambda^{-1})$	Gauss $\text{Gamma}(\lambda   a, b)$
$nD$	
$\mathcal{N}(x   \mu, \Lambda^{-1})$	Wishart $\text{Wish}(\Lambda   W, \nu)$
$nD$	Gauss
$\mathcal{N}(x   \mu, \Lambda^{-1})$	Gauss-Wishart
	$\mathcal{N}(\mu   \mu_0, (\beta \Lambda)^{-1}) \text{ Wish}(\Lambda   W, \nu)$

3:33

## Distributions over continuous domain

3:34

## Distributions over continuous domain

- Let  $x$  be a continuous RV. The **probability density function (pdf)**  $p(x) \in [0, \infty)$  defines the probability

$$P(a \leq x \leq b) = \int_a^b p(x) dx \in [0, 1]$$

The **(cumulative) probability distribution**  $F(y) = P(x \leq y) = \int_{-\infty}^y dx p(x) \in [0, 1]$  is the cumulative integral with  $\lim_{y \rightarrow \infty} F(y) = 1$

(In discrete domain: *probability distribution* and *probability mass function*  $P(x) \in [0, 1]$  are used synonymously.)

- Two basic examples:

**Gaussian:**  $\mathcal{N}(x | \mu, \Sigma) = \frac{1}{(2\pi|\Sigma|)^{1/2}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1} (x-\mu)}$

**Dirac or  $\delta$  ("point particle")**  $\delta(x) = 0$  except at  $x = 0$ ,  $\int \delta(x) dx = 1$

$\delta(x) = \frac{\partial}{\partial x} H(x)$  where  $H(x) = [x \geq 0]$  = Heavyside step function

3:35

## Gaussian distribution

- 1-dim:  $\mathcal{N}(x | \mu, \sigma^2) = \frac{1}{|2\pi\sigma^2|^{1/2}} e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}$

- n-dim Gaussian in *normal form*:

$$\mathcal{N}(x | \mu, \Sigma) = \frac{1}{|2\pi\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu)^\top \Sigma^{-1} (x - \mu)\right\}$$

with **mean**  $\mu$  and **covariance** matrix  $\Sigma$ . In *canonical form*:

$$\mathcal{N}[x | a, A] = \frac{\exp\left\{-\frac{1}{2}a^\top A^{-1}a\right\}}{|2\pi A^{-1}|^{1/2}} \exp\left\{-\frac{1}{2}x^\top A x + x^\top a\right\} \quad (2)$$

with **precision** matrix  $A = \Sigma^{-1}$  and coefficient  $a = \Sigma^{-1}\mu$  (and mean  $\mu = A^{-1}a$ ).

- Gaussian identities: see <http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gaussians.pdf>

3:36

## Gaussian identities

Symmetry:  $\mathcal{N}(x | a, A) = \mathcal{N}(a | x, A) = \mathcal{N}(x - a | 0, A)$

Product:

$$\begin{aligned} \mathcal{N}(x | a, A) \mathcal{N}(x | b, B) &= \mathcal{N}[x | A^{-1}a + B^{-1}b, A^{-1} + B^{-1}] \mathcal{N}(a | b, A + B) \\ \mathcal{N}[x | a, A] \mathcal{N}[x | b, B] &= \mathcal{N}[x | a + b, A + B] \mathcal{N}(A^{-1}a | B^{-1}b, A^{-1} + B^{-1}) \end{aligned}$$

"Propagation":

$$\int_y \mathcal{N}(x | a + Fy, A) \mathcal{N}(y | b, B) dy = \mathcal{N}(x | a + Fb, A + FBF^\top)$$

Transformation:

$$\mathcal{N}(Fx + f | a, A) = \frac{1}{|F|} \mathcal{N}(x | F^{-1}(a - f), F^{-1}AF^{-\top})$$

Marginal & conditional:

$$\mathcal{N}\left(\begin{array}{c} x \\ y \end{array} \middle| \begin{array}{cc} a & A \\ b & C^\top \\ \hline C & B \end{array}\right) = \mathcal{N}(x | a, A) \cdot \mathcal{N}(y | b + C^\top A^{-1}(x - a), B - C^\top A^{-1}C)$$

More Gaussian identities: see <http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gaussians.pdf>

3:37

## Motivation for Gaussian distributions

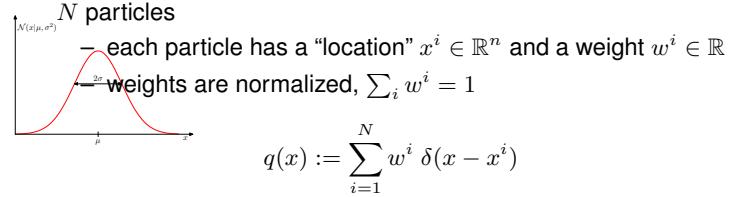
- Gaussian Bandits
- Control theory, Stochastic Optimal Control
- State estimation, sensor processing, Gaussian filtering (Kalman filtering)
- Machine Learning
- etc

3:38

## Particle Approximation of a Distribution

- We approximate a distribution  $p(x)$  over a continuous domain  $\mathbb{R}^n$

- A particle distribution  $q(x)$  is a weighed set  $\mathcal{S} = \{(x^i, w^i)\}_{i=1}^N$  of



where  $\delta(x - x^i)$  is the  $\delta$ -distribution.

- Given weighted particles, we can estimate for any (smooth)  $f$ :

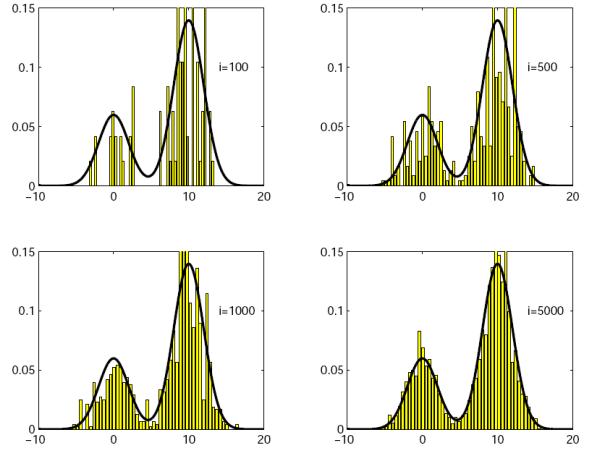
$$\langle f(x) \rangle_p = \int_x f(x)p(x)dx \approx \sum_{i=1}^N w^i f(x^i)$$

See *An Introduction to MCMC for Machine Learning* [www.cs.ubc.ca/~nando/papers/mlintro.pdf](http://www.cs.ubc.ca/~nando/papers/mlintro.pdf)

3:39

## Particle Approximation of a Distribution

Histogram of a particle representation:



3:40

## Motivation for particle distributions

- Numeric representation of "difficult" distributions
  - Very general and versatile
  - But often needs many samples
- Distributions over games (action sequences), sample based planning, MCTS
- State estimation, particle filters
- etc

3:41

## Some more continuous distributions\*

Gaussian	$\mathcal{N}(x   a, A)$	=
	$\frac{1}{\sqrt{2\pi}  A ^{1/2}} e^{-\frac{1}{2}(x-a)^\top A^{-1} (x-a)}$	
Dirac or $\delta$	$\delta(x) = \frac{\partial}{\partial x} H(x)$	
Student's t (=Gaussian for $\nu \rightarrow \infty$ , otherwise heavy tails)	$p(x; \nu) \propto [1 + \frac{x^2}{\nu}]^{-\frac{\nu+1}{2}}$	
Exponential (distribution over single event time)	$p(x; \lambda) = [x \geq 0] \lambda e^{-\lambda x}$	
Laplace ("double exponential")	$p(x; \mu, b) = \frac{1}{2b} e^{- x-\mu /b}$	
Chi-squared	$p(x; k) \propto [x \geq 0] x^{k/2-1} e^{-x/2}$	
Gamma	$p(x; k, \theta) \propto [x \geq 0] x^{k-1} e^{-x/\theta}$	

3:42

## 4 Mobile Robotics

### Orga

- **Mailinglist:**

<https://mailman.informatik.uni-stuttgart.de/mailman/listinfo/lehre15-robotik>

- **Tutorial:**

because of the “Ringvorlesung” the second tutorial will start at 18:15 after the “Ringvorlesung”.

- **Code:**

We're going to release our code tomorrow. Install it and run it.

Use Ubuntu 14.04. For instruction see:

<https://ipvs.informatik.uni-stuttgart.de/mlr/teaching/robotics/>

4:1



<http://www.darpa.mil/grandchallenge05/>

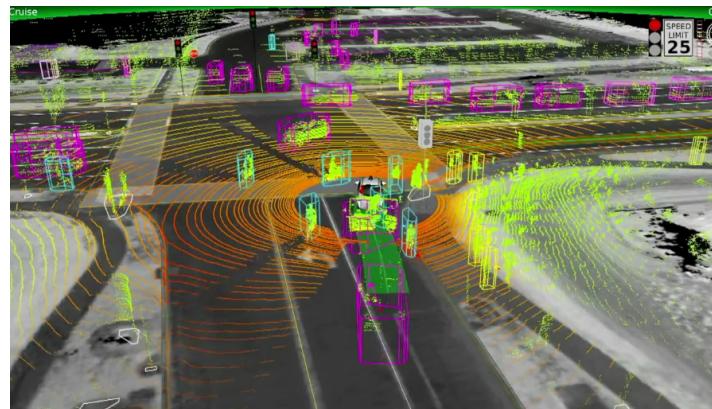
DARPA Grand Challenge 2005

4:2



<http://www.darpa.mil/grandchallenge/index.asp>

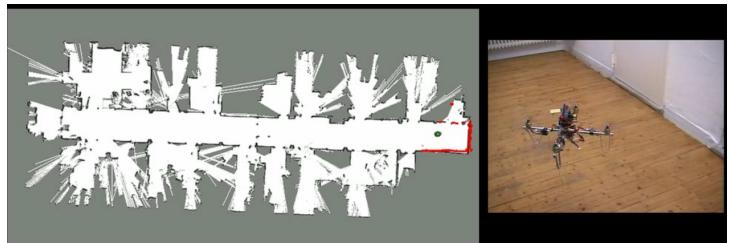
DARPA Grand Urban Challenge 2007



<https://www.youtube.com/watch?v=tiwVMrTLUWg>

Google Autonomous Car

4:3



<http://www.slawomir.de/publications/grzonka09icra/grzonka09icra.pdf>

Quadcopter Indoor Localization

4:4



<http://stair.stanford.edu/multimedia.php>

STAIR: STanford Artificial Intelligence Robot

4:5

### Mobile Robotics

## Mobile Robotics

Bayes' filter for localization (and mapping)

## Recursive state estimation

### Probabilistic state estimation

#### Bayes' rule

4:7

- Given the local sensor readings  $y_t$ , the current state  $x_t$  (location, position) is *uncertain*.



- which hallway?
- which door exactly?
- which heading direction?

## Outline

### PART I:

A core challenge in mobile robotics is **state estimation**  
 → Bayesian filtering & smoothing  
 histogram filter, particle filter, Kalman filter

### PART II:

Another challenge is to **build a map** while exploring  
 → SLAM (simultaneous localization and mapping)

How to represent our knowledge? → **Probabilities**

TODO switch to probability slides

4:8

## State Estimation Problem

- What is the probability of being in front of room 154, given we see what is shown in the image?
- What is the probability given that we were just in front of room 156?
- What is the probability given that we were in front of room 156 and moved 15 meters?



4:12

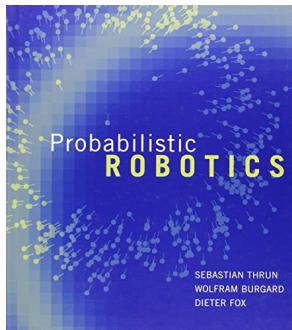
## Recall Bayes' theorem

$$P(X|Y) = \frac{P(Y|X) P(X)}{P(Y)}$$

$$\text{posterior} = \frac{\text{likelihood} \cdot \text{prior}}{\text{(normalization)}}$$

4:13

- How can we apply this to the



Sebastian Thrun, Wolfram Burgard, Dieter Fox: Probabilistic Robotics  
[probabilistic-robotics.org](http://probabilistic-robotics.org)  
 SEBASTIAN THRUN  
 WOLFRAM BURGARD  
 DIETER FOX

4:9

## 4.1 PART I: State Estimation

4:10

**Goal:** Given a map, sensor measurements and control signals estimate our localization.

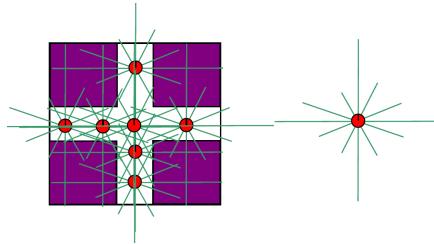
- Our sensory data does not provide sufficient information to determine our location.

State Estimation Problem?



Using Bayes Rule:

$$P(\text{location} | \text{sensor}) = \frac{P(\text{sensor} | \text{location}) P(\text{location})}{P(\text{sensor})}$$



4:14

## Bayes Filter

$x_t$  = state (location) at time  $t$

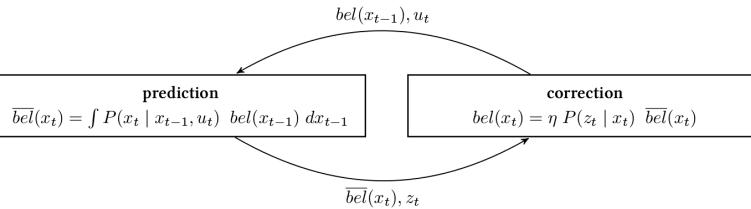
$y_t$  = sensor readings at time  $t$

$u_{t-1}$  = control command (action, steering, velocity) at time  $t-1$

- Given the history  $y_{0:t}$  of observations and  $u_{0:t-1}$  control, we want to compute the probability distribution over the state at time  $t$

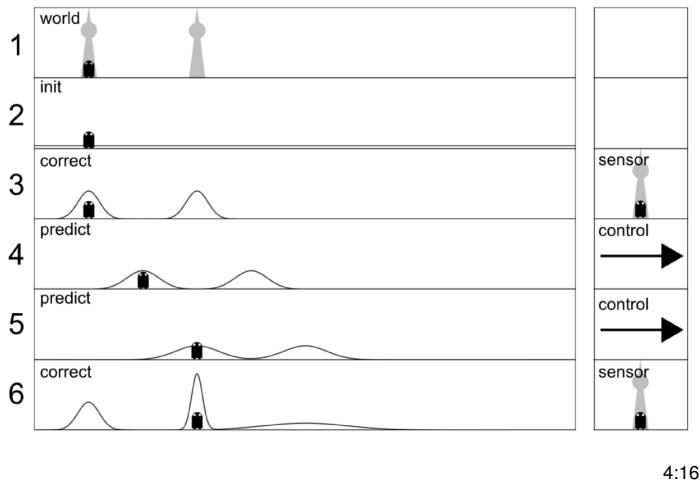
$$p_t(x_t) := P(x_t | y_{0:t}, u_{0:t-1})$$

- Generally:



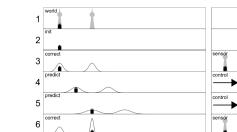
4:15

## Bayes Filter



4:16

## Bayes Filter



- We have a belief  $p_{t-1}(x_{t-1})$  of our previous position

- We use the motion model to predict the current position

$$\hat{p}_t(x_t) \propto \int_{x_{t-1}} P(x_t | u_{t-1}, x_{t-1}) p_{t-1}(x_{t-1}) dx_{t-1}$$

- We integrate this with the current observation to get a better belief

$$p_t(x_t) \propto P(y_t | x_t) \hat{p}_t(x_t)$$

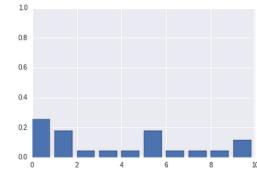
$$p_t(x_t) \propto \underbrace{P(y_t | x_t)}_{\text{new information}} \underbrace{\int_{x_{t-1}} P(x_t | u_{t-1}, x_{t-1}) \underbrace{p_{t-1}(x_{t-1})}_{\text{old estimate}} dx_{t-1}}_{\text{predictive estimate } \hat{p}_t(x_t)}$$

4:17

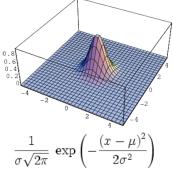
## Bayesian Filters

How to represent the belief  $p_t(x_t)$ :

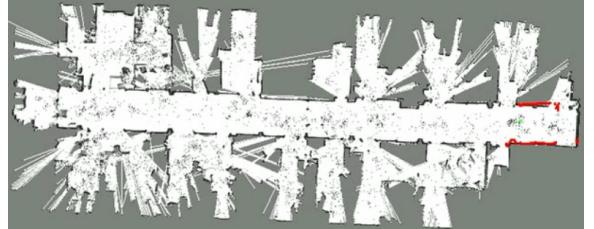
### Histogram



### Gaussian



### Particles



4:18

## Histogram Filter

Intuition first: histogram filter as simple bayesian filter

4:19

## Histogram Filter

```

1 def histogram_filter({p_k,t-1}, u_t, z_t):
2     for all k:
3         # prediction
4         p_k,t = sum_i P(X = x_k | u_t, X_{t-1} = x_i) p_{i,t-1}
4                                     state transition probability
5         # correction
6         p_k,t = p_k,t * P(z_t | x_k) / p_k,t
6                                     measurement probability
7     return {p_k,t}

```

4:20

## Orga

- Slides

Earlier?

- Homework:

Any problems?

- Software:

- Not released yet, sorry!
- Future homework

## Intermission

Prediction/correction filter

→ motion model

→ observation model

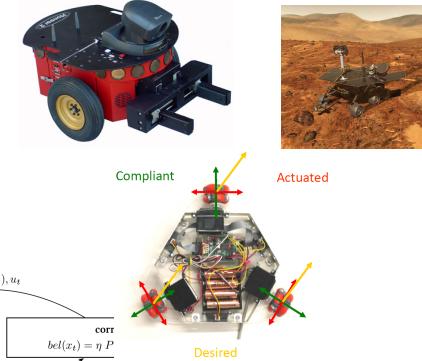
4:24

- Notation

- Questions?

4:21

## Types of Robot Mobility



### Recap:

- Probability basics, Bayes' theorem and Bayes' filter
- Histogram filter: discrete Bayes' filter

### Today:

- Derive Bayes' filter
- Intermission: motion and observation models
- Particle filter
- Kalman filter family

4:22

## Bayes Filter

$$\begin{aligned}
 p_t(x_t) &:= P(x_t | y_{0:t}, u_{0:t-1}) \\
 &\propto P(y_t | x_t, y_{0:t-1}, u_{0:t-1}) P(x_t | y_{0:t-1}, u_{0:t-1}) \\
 &\propto P(y_t | x_t) \\
 &\propto P(y_t | x_t) \\
 &\propto P(y_t | x_t) \\
 &\propto \underbrace{P(y_t | x_t)}_{\text{new information}} \underbrace{\int_{x_{t-1}} P(x_t, x_{t-1} | y_{0:t-1}, u_{0:t-1}) dx_{t-1}}_{\text{predictive estimate}}
 \end{aligned}$$

Derive Bayes' filter

Bayes rule:  $P(X|Y, Z) = c P(Y|X, Z) P(X|Z)$

Markov property: independence of past observations and controls

by definition of the marginal

by definition of a conditional

given  $x_{t-1}$ ,  $x_t$  depends only on the controls  $u_{t-1}$  (Markov Property)

→ Bayes' filter

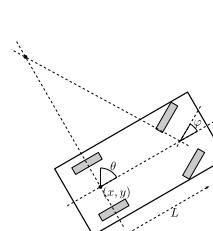
## Types of Robot Mobility: Non-holonomic Systems

- We define a **nonholonomic system** as one with **differential constraints**:

$$\dim(u_t) < \dim(x_t)$$

⇒ Not all degrees of freedom are directly controllable

- Example: car



$$\begin{aligned}
 \text{State } q &= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} & \text{Controls } u &= \begin{pmatrix} v \\ \varphi \end{pmatrix}
 \end{aligned}$$

**System equation**

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v \cos \theta \\ v \sin \theta \\ (v/L) \tan \varphi \end{pmatrix}$$

(More on non-holonomic systems in the "path planning" lecture)

4:26

## Motion Models

- Each type of robot mobility corresponds to a

system equation  $x_{t+1} = x_t + \tau f(x_t, u_t)$

4:23

or, if the dynamics are stochastic,

$$P(x_{t+1} | u_t, x_t) = \mathcal{N}(x_{t+1} | x_t + \tau f(x_t, u_t), \Sigma)$$

- I.e.:

- describe how the system will move over time
- describe the noise and uncertainty of the movement

4:27

- Different representations:

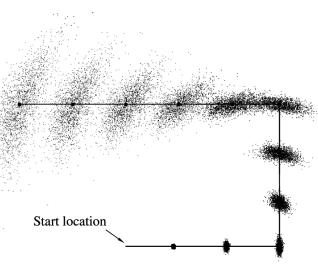
- cartesian coordinates:  $z_t := (z_x, z_y)$
- range and bearing:  $z_t := (z_{\text{dist}}, z_{\alpha})$
- polar coordinates:  $z_t := (z_{\text{pitch}}, z_{\text{yaw}})$



4:30

## Example Transition Model

- The predictive distributions  $\hat{p}_t(x_t)$  without integrating observations (removing the  $P(y_t|x_t)$  part from the Bayesian filter)



(from *Robust Monte Carlo localization for mobile robots* Sebastian Thrun, Dieter Fox, Wolfram Burgard, Frank Dellaert)

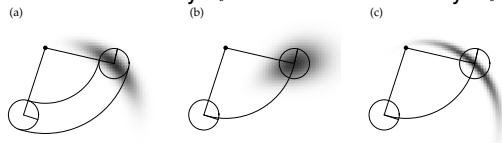
Also called “Dead Reckoning” (Odometry)

4:28

## Motion Model and Noise Parameterization

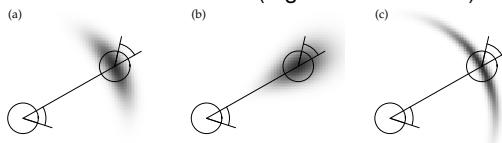
- **Velocity motion model:**  $u_t = (v_t, w_t)$

with transitional velocity  $v_t$  and rotational velocity  $w_t$



- **Odometry motion model:**  $u_t = (\bar{x}_t, \bar{x}_{t-1})$

with  $\bar{x}$  the internal measured (e.g. wheel encoder) state



- Many more depending on your robot...

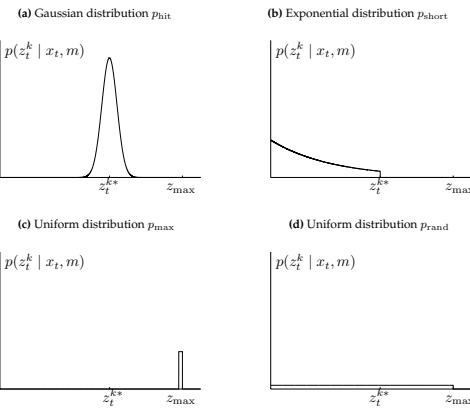
4:29

## Observation Model

- Different sensors

- require different observation models
- have different noise characteristics

## Example: Range Finder

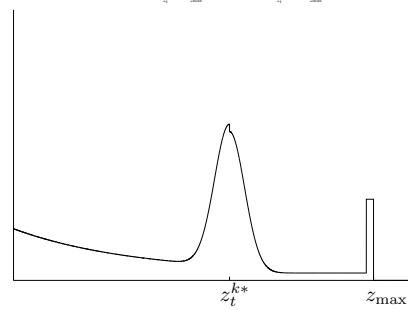
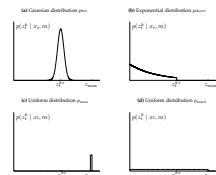


a) correct   b) unexpected object   c) failures   d) random

(taken from *Probabilistic Robotics*. Sebastian Thrun, Dieter Fox, Wolfram Burgard)

4:31

## Example: Range Finder



(taken from *Probabilistic Robotics*. Sebastian Thrun, Dieter Fox, Wolfram Burgard)

4:32

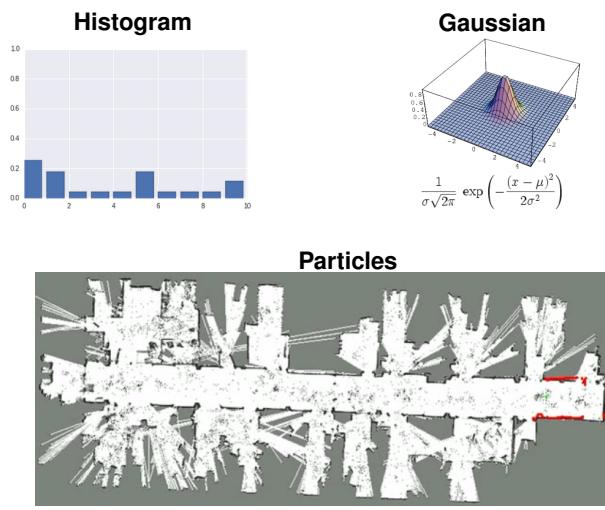
## End of Intermission

→ back to Bayes' filter and localization

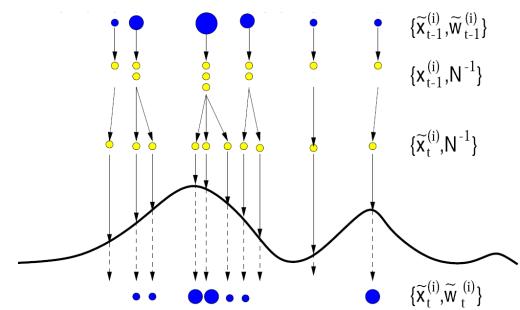
4:33

## Bayesian Filters

How to represent the belief  $p_t(x_t)$ :



4:34



1. Start with  $N$  particles  $\{(x_{t-1}^i, w_{t-1}^i)\}_{i=1}^N$
2. Resample particles to get  $N$  weight-1-particles:  $\{\hat{x}_t^i\}_{i=1}^N$
3. Use motion model to get new "predictive" particles  $\{x_t^i\}_{i=1}^N$  each  $x_t^i \sim P(x_t | u_{t-1}, \hat{x}_{t-1}^i)$
4. Use observation model to assign new weights  $w_t^i \propto P(y_t | x_t^i)$

4:36

## Practical Considerations

- Given a complex multi-modal distribution: *where are we?*
  - particle with the highest weight?
  - which *cluster*?
  - kernel density estimation: cluster with highest density?
- Particle deprivation:  $\{x_t^i\}_{i=1}^N$  have very low observation likelihood  $P(y_t | x_t^i)$ : "particle die over time"
  - random samples
  - *sensor reset*: samples from  $P(y_t | x_t)$
- How many particles?  $\rightarrow$  adaptive techniques
- Resampling is important: many variations exist!
  - Liu & Chen (1998): Sequential Monte Carlo Methods for Dynamic Systems.*
  - Douc, Cappé & Moulines: Comparison of Resampling Schemes for Particle Filtering.*

4:37

## Particle Filter: Summary

- conceptually simple
- approximates the true distribution  $p(x)$
- non-parametric
- can deal with non-linear transformations
- "Particle Filter" in different communities
  - *Monte Carlo Localization* in the mobile robotics community
  - *Condensation Algorithm* in the vision community

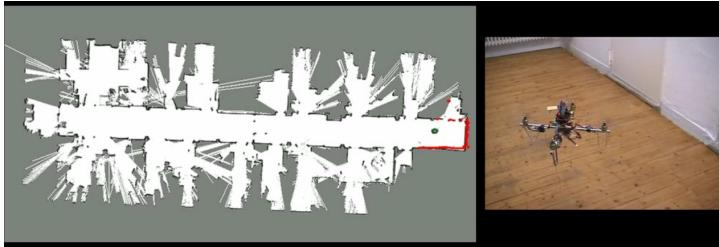
4:35

## Particle Filter := Bayesian Filtering with Particles

$$(\text{Bayes Filter: } p_t(x_t) \propto P(y_t | x_t) \int_{x_{t-1}} P(x_t | u_{t-1}, x_{t-1}) p_{t-1}(x_{t-1}) dx_{t-1})$$

4:38

## Example: Quadcopter Localization



<http://www.slawomir.de/publications/grzonka09icra/grzonka09icra.pdf>

### Quadcopter Indoor Localization

4:39

## Kalman filter := Bayesian Filtering with Gaussians

Exact filter with gaussian state, linear Gaussian observations and linear Gaussian transitions.

- parametric
- uni-modal

4:40

## Kalman filter := Bayesian Filtering with Gaussians

Bayes Filter:  $p_t(x_t) \propto P(y_t | x_t) \int_{x_{t-1}} P(x_t | u_{t-1}, x_{t-1}) p_{t-1}(x_{t-1}) dx_{t-1}$

- Can be computed analytically for linear-Gaussian observations and transitions:

$$P(y_t | x_t) = \mathcal{N}(y_t | Cx_t + c, W)$$

$$P(x_t | u_{t-1}, x_{t-1}) = \mathcal{N}(x_t | A(u_{t-1}) x_{t-1} + a(u_{t-1}), Q)$$

Definition:

$$\mathcal{N}(x | a, A) = \frac{1}{(2\pi A)^{1/2}} \exp\left\{-\frac{1}{2}(x - a)^\top A^{-1} (x - a)\right\}$$

Product:

$$\mathcal{N}(x | a, A) \mathcal{N}(x | b, B) = \mathcal{N}(x | B(A+B)^{-1}a + A(A+B)^{-1}b, A(A+B)^{-1}B) \mathcal{N}(a | b, A+B)$$

"Propagation":

$$\int_y \mathcal{N}(x | a + Fy, A) \mathcal{N}(y | b, B) dy = \mathcal{N}(x | a + Fb, A + FBF^\top)$$

Transformation:

$$\mathcal{N}(Fx + f | a, A) = \frac{1}{|F|} \mathcal{N}(x | F^{-1}(a - f), F^{-1}AF^{-\top})$$

(more identities: see "Gaussian identities" <http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gaussians.pdf>)

4:41

## Kalman filter derivation

- state:  $p_t(x_t) = \mathcal{N}(x_t | s_t, S_t)$
- motion. model:  $P(x_t | u_{t-1}, x_{t-1}) = \mathcal{N}(x_t | Ax_{t-1} + a, Q)$
- obs. model:  $P(y_t | x_t) = \mathcal{N}(y_t | Cx_t + c, W)$

$$\begin{aligned} p_t(x_t) &\propto P(y_t | x_t) \int_{x_{t-1}} P(x_t | u_{t-1}, x_{t-1}) p_{t-1}(x_{t-1}) dx_{t-1} \\ &= \mathcal{N}(y_t | Cx_t + c, W) \int_{x_{t-1}} \mathcal{N}(x_t | Ax_{t-1} + a, Q) \mathcal{N}(x_{t-1} | s_{t-1}, S_{t-1}) dx_{t-1} \\ &= \mathcal{N}(y_t | Cx_t + c, W) \mathcal{N}(x_t | \underbrace{As_{t-1} + a}_{=: \hat{s}_t}, \underbrace{Q + AS_{t-1}A^\top}_{=: \hat{S}_t}) \\ &= \mathcal{N}(Cx_t + c | y_t, W) \mathcal{N}(x_t | \hat{s}_t, \hat{S}_t) \\ &= \mathcal{N}[x_t | C^\top W^{-1}(y_t - c), C^\top W^{-1}C] \mathcal{N}(x_t | \hat{s}_t, \hat{S}_t) \\ &= \mathcal{N}(x_t | s_t, S_t) \cdot \langle \text{terms indep. of } x_t \rangle \end{aligned}$$

$$S_t = (C^\top W^{-1} C + \hat{S}_t^{-1})^{-1} = \hat{S}_t - \underbrace{\hat{S}_t C^\top (W + C \hat{S}_t C^\top)^{-1} C \hat{S}_t}_{\text{"Kalman gain" } K}$$

$$s_t = S_t [C^\top W^{-1} (y_t - c) + \hat{S}_t^{-1} \hat{s}_t] = \hat{s}_t + K(y_t - C\hat{s}_t - c)$$

insert definitions

"propagate"

switch  $x$  and mean

multiply Gaussians

reorder

new covariance  $S_t$ ; uses the general Woodbury identity

new mean  $s_t$ ; uses  $S_t C^\top W^{-1} = K$  and  $S_t \hat{S}_t^{-1} = I - KC$

4:42

## Orga

- Mailinglist:

<https://mailman.informatik.uni-stuttgart.de/mailman/listinfo/lehre15-robotik>

- Check your mails:

You got spam?!

- Software:

DEMO

Still problems? We'll solve them after the lecture. Please stay.

- Clarifications:

– Why particle filters?

– Observation model and likelihood example

– Normalization

4:43

## KF Alternative Formulation

$\mu \in \mathbb{R}^n$  mean,  $\Sigma \in \mathbb{R}^{n \times n}$  covariance matrix,  $u \in \mathbb{R}^m$  control,  $z \in \mathbb{R}^k$  observation,  $A \in \mathbb{R}^{n \times n}$  and  $B \in \mathbb{R}^{n \times m}$  motion transition matrix,  $C \in \mathbb{R}^{k \times n}$  observation matrix,  $R \in \mathbb{R}^{n \times n}$  and  $Q \in \mathbb{R}^{k \times k}$  additive motion and observation noise

KF update( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$ ):

predict:

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^\top + R$$

correct:

$$K_t = \bar{\Sigma}_t C_t^\top (C_t \bar{\Sigma}_t C_t^\top + Q_t)^{-1} \quad \text{Kalman gain}$$

$$v_t = z_t - C_t \bar{\mu}_t \quad \text{innovation}$$

$$\mu_t = \bar{\mu}_t + K_t v_t$$

$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$$

4:44

# Example

## What is $A, B, C$ ?

4:45

### KF Extensions

- KF assumes linear dynamics, but (almost) nothing has linear dynamics
  - approximate non-linear function: *Extended Kalman Filter*
  - sample locally and reapproximate as Gaussian: *Unscented Transform*
- KF assumes uni-modal Gaussian belief, but the world is more complex (the belief is rarely unimodal)
  - use *Gaussian Mixture*

4:46

### Extended Kalman filter (EKF)

Bayes Filter:  $p_t(x_t) \propto P(y_t | x_t) \int_{x_{t-1}} P(x_t | u_{t-1}, x_{t-1}) p_{t-1}(x_{t-1}) dx_{t-1}$

- If  $P(y_t | x_t)$  or  $P(x_t | u_{t-1}, x_{t-1})$  are not linear:
 
$$P(y_t | x_t) = \mathcal{N}(y_t | g(\mathbf{x}_t), W)$$

$$P(x_t | u_{t-1}, x_{t-1}) = \mathcal{N}(x_t | f(\mathbf{x}_{t-1}, u_{t-1}), Q)$$
  - approximate  $f$  and  $g$  as locally linear → Jacobians  $F$  and  $G$
  - in practice EKFs are more commonly used than vanilla KFs

4:47

### EKF Code

$\mu \in \mathbb{R}^n$  mean,  $\Sigma \in \mathbb{R}^{n \times n}$  covariance matrix,  $u \in \mathbb{R}^m$  control,  $z \in \mathbb{R}^k$  observation,  $g$  non linear state transition function,  $G$  Jacobi matrix of  $g$ ,  $h$  non linear observation function,  $H$  Jacobi matrix of  $h$ ,  $R \in \mathbb{R}^{n \times n}$  and  $Q \in \mathbb{R}^{k \times k}$  additive motion and observation noise

KF update( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$ ):

predict:

$$\bar{\mu}_t = f(u_t, \mu_{t-1})$$

$$\bar{\Sigma}_t = F_t \Sigma_{t-1} F_t^T + R_t$$

correct:

$$K_t = \bar{\Sigma}_t G_t^T (G_t \bar{\Sigma}_t G_t^T + Q_t)^{-1} \quad \text{Kalman gain}$$

$$v_t = z_t - g(\bar{\mu}_t) \quad \text{innovation}$$

$$\mu_t = \bar{\mu}_t + K_t v_t$$

$$\Sigma_t = (I - K_t G_t) \bar{\Sigma}_t$$

4:48

### Unscented Transform

If  $P(y_t | x_t)$  or  $P(x_t | u_{t-1}, x_{t-1})$  are not linear:

$$P(y_t | x_t) = \mathcal{N}(y_t | g(\mathbf{x}_t), W)$$

$$P(x_t | u_{t-1}, x_{t-1}) = \mathcal{N}(x_t | f(\mathbf{x}_{t-1}, u_{t-1}), Q)$$

- *Unscented Transform (UT)* is a method to calculate statistics of RV
- reapproximate the posterior as Gaussian using UT

#### How:

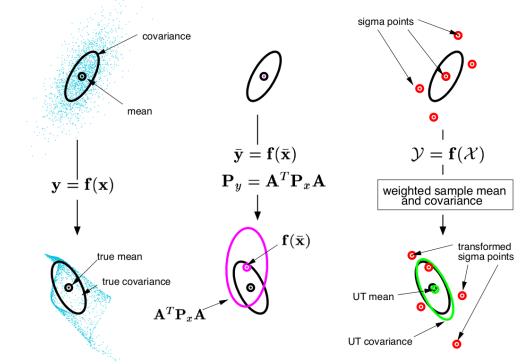
- deterministically sample from the prior,
- apply non-linear function on sample points,
- from this re-approximate the posterior as Gaussian

(no need to calculate the derivative of  $f$  and  $g$ )

4:49

### Unscented Transform

Actual (sampling)      Linearized (EKF)      UT



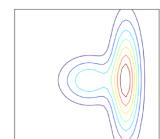
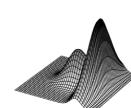
Eric A. Wan and Rudolph van der Merwe: *The Unscented Kalman Filter for Non-linear Estimation*

4:50

### Multi-Hypotheses KF

- Belief is rarely uni-modal → use a mixture of Gaussians

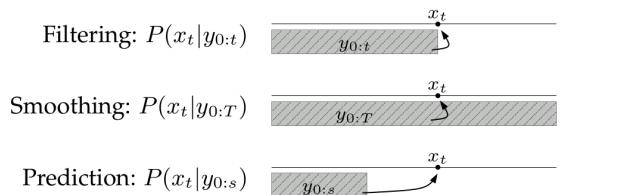
$$p_t = \sum_{i=1}^N w_i \mathcal{N}(x_i, S_i) \quad \text{with } \sum w_i = 1$$



- Conceptually similar to particle filters
- One component is the result of unique sequence of data association decisions

4:51

## Bayes smoothing



4:52

## Bayes smoothing

- Let  $\mathcal{P} = y_{0:t}$  past observations,  $\mathcal{F} = y_{t+1:T}$  future observations

$$\begin{aligned} P(x_t | \mathcal{P}, \mathcal{F}, u_{0:T}) &\propto P(\mathcal{F} | x_t, \mathcal{P}, u_{0:T}) P(x_t | \mathcal{P}, u_{0:T}) \\ &= \underbrace{P(\mathcal{F} | x_t, u_{t:T})}_{=: \beta_t(x_t)} \underbrace{P(x_t | \mathcal{P}, u_{0:t-1})}_{=: p(x_t)} \end{aligned}$$

*Bayesian smoothing fuses a forward filter  $p_t(x_t)$  with a backward "filter"  $\beta_t(x_t)$*

- Backward recursion (derivation analogous to the Bayesian filter)

$$\begin{aligned} \beta_t(x_t) &:= P(y_{t+1:T} | x_t, u_{t:T}) \\ &= \int_{x_{t+1}} \beta_{t+1}(x_{t+1}) P(y_{t+1} | x_{t+1}) P(x_{t+1} | x_t, u_t) dx_{t+1} \end{aligned}$$

4:53

## 4.2 PART II: Simultaneous Localization and Mapping (SLAM)

4:54

### Localization and Mapping

- The Bayesian filter requires an observation model  $P(y_t | x_t)$
- A **map** is something that provides the observation model:  
*A map tells us for each  $x_t$  what the sensor readings  $y_t$  might look like*

4:55

## Types of maps

### Grid map



### Landmark map



### Laser scan map



### Victoria Park data set

M. Montemerlo, S. Thrun, D. Koller, & B. Wegbreit (2003): *FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges*. IJCAI, 1151–1156.

4:56

## Simultaneous Localization and Mapping Problem

- Notation:

$x_t$  = state (location) at time  $t$

$y_t$  = sensor readings at time  $t$

$u_{t-1}$  = control command (action, steering, velocity) at time  $t-1$

$m$  = the map; formally: *a map is the parameters that define  $P(y_t | x_t, m)$*

- Given the history  $y_{0:t}$  and  $u_{0:t-1}$ , we want to compute the belief over the pose **AND THE MAP**  $m$  at time  $t$

$$p_t(x_t, m) := P(x_t, m | y_{0:t}, u_{0:t-1})$$

- We assume to know the:

– transition model  $P(x_t | u_{t-1}, x_{t-1})$

– observation model  $P(y_t | x_t, m)$  (*defined by the map*)

4:57

## SLAM: classical "chicken or egg problem"

- If we knew the state trajectory  $x_{0:t}$  we could efficiently compute the belief over the map

$$P(m | x_{0:t}, y_{0:t}, u_{0:t-1})$$

- If we knew the map we could use a Bayes filter to compute the belief over the state

$$P(x_t | m, y_{0:t}, u_{0:t-1})$$

- SLAM requires to tie state estimation and map building together:

- 1) Joint inference on  $x_t$  and  $m$   
(→ Kalman-SLAM)

- 2) Tie a state hypothesis (=particle) to a map hypothesis  
(→ particle SLAM)
- 3) Frame everything as a graph optimization problem  
(→ graph SLAM)

4:58

- Lacks advantages of Particle Filter  
(multiple hypothesis, more robust to non-linearities)

4:61

## Joint Bayesian Filter over $x$ and $m$

- A (formally) straight-forward approach is the joint Bayesian filter

$$p_t(x_t, m) \propto P(y_t | x_t, m) \int_{x_{t-1}} P(x_t | u_{t-1}, x_{t-1}) p_{t-1}(x_{t-1}, m) dx_{t-1}$$

compare to:

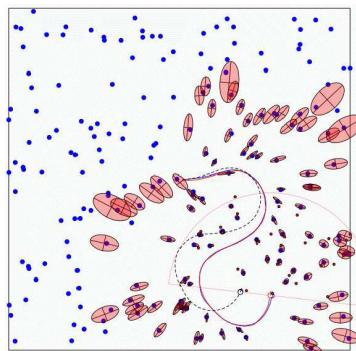
$$p_t(x_t) \propto P(y_t | x_t) \int_{x_{t-1}} P(x_t | u_{t-1}, x_{t-1}) p_{t-1}(x_{t-1}) dx_{t-1}$$

But: How represent a belief over high-dimensional  $x_t, m$ ?

4:59

## Map uncertainty

- In the case the map  $m = (\theta_1, \dots, \theta_N)$  is a set of  $N$  landmarks,  
 $\theta_j \in \mathbb{R}^2$



- Use Gaussians to represent the uncertainty of landmark positions

4:60

## SLAM with particles

**Core idea: Each particle carries its own map belief**

- Use a conditional representation “ $p_t(x_t, m) = p_t(x_t) p_t(m | x_t)$ ”  
(This notation is flaky... the below is more precise)
- As for the Localization Problem use particles to represent the *pose belief*  $p_t(x_t)$   
Note: Each particle  $i$  actually “has a history  $x_{0:t}^i$ ” – a whole trajectory!
- For each particle separately, estimate the *map belief*  $p_t^i(m)$  conditioned on the particle history  $x_{0:t}^i$ .  
The conditional beliefs  $p_t^i(m)$  may be factorized over grid points or landmarks of the map

K. Murphy (1999): *Bayesian map learning in dynamic environments*.

[http://www.cs.ubc.ca/~murphyk/Papers/map\\_nips99.pdf](http://www.cs.ubc.ca/~murphyk/Papers/map_nips99.pdf)

4:62

## Map estimation for a *given* particle history

- Given  $x_{0:t}$  (e.g. a trajectory of a particle), what is the posterior over the map  $m$ ?

→ simplified Bayes Filter:

$$p_t(m) := P(m | x_{0:t}, y_{0:t}) \propto P(y_t | m, x_t) p_{t-1}(m)$$

(no transition model: assumption that map is constant)

- In the case of landmarks (FastSLAM):  
 $m = (\theta_1, \dots, \theta_N)$   
 $\theta_j$  = position of the  $j$ th landmark,  $j \in \{1, \dots, N\}$   
 $n_t$  = which landmark we observe at time  $t$ ,  $n_t \in \{1, \dots, N\}$

We can use a separate (Gaussian) Bayes Filter for each  $\theta_j$  conditioned on  $x_{0:t}$ , each  $\theta_j$  is independent from each  $\theta_k$ :

$$P(\theta_{1:N} | x_{0:t}, y_{0:n}, n_{0:t}) = \prod_j P(\theta_j | x_{0:t}, y_{0:n}, n_{0:t})$$

- Drawbacks:

- Scaling (full covariance matrix is  $O(N^2)$ )
- Sometimes non-robust (uni-modal, “data association problem”)

4:63

## Particle likelihood in SLAM

- Particle likelihood for Localization Problem:

$$w_t^i = P(y_t | x_t^i)$$

(determines the new importance weight  $w_t^i$ )

- In SLAM the map is uncertain → each particle is weighted with the *expected* likelihood:

$$w_t^i = \int P(y_t | x_t^i, m) p_{t-1}(m) dm$$

- In case of landmarks (FastSLAM):

$$w_t^i = \int P(y_t | x_t^i, \theta_{nt}, n_t) p_{t-1}(\theta_{nt}) d\theta_{nt}$$

- Data association problem (actually we don't know  $n_t$ ):

For each particle separately choose  $n_t^i = \text{argmax}_{n_t} w_t^i(n_t)$

4:64

## Particle-based SLAM summary

- We have a set of  $N$  particles  $\{(x^i, w^i)\}_{i=1}^N$  to represent the pose belief  $p_t(x_t)$
- For each particle we have a separate map belief  $p_t^i(m)$ ; in the case of landmarks, this factorizes in  $N$  separate 2D-Gaussians

- Iterate

1. Resample particles to get  $N$  weight-1-particles:  $\{\hat{x}_{t-1}^i\}_{i=1}^N$
2. Use motion model to get new “predictive” particles  $\{x_t^i\}_{i=1}^N$
3. Update the map belief  $p_m^i(m) \propto P(y_t | m, x_t) p_{t-1}^i(m)$  for each particle
4. Compute new importance weights  $w_t^i \propto \int P(y_t | x_t^i, m) p_{t-1}(m) dm$  using the observation model and the map belief

4:65

## Demo: Visual SLAM

- Map building from a freely moving camera



4:66

## Demo: Visual SLAM

- Map building from a freely moving camera

– SLAM has become a big topic in the vision community..

– features are typically landmarks  $\theta_{1:N}$  with SURF/SIFT features

– PTAM (Parallel Tracking and Mapping) parallelizes computations...

[PTAM1](#) [PTAM2](#) [DTAM](#) [KinectFusion](#) [LSD-SLAM](#)

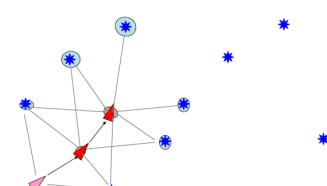
Klein & Murray: *Parallel Tracking and Mapping for Small AR Workspaces* <http://www.robots.ox.ac.uk/~gk/PTAM/>

Newcombe, Lovegrove & Davison: *DTAM: Dense Tracking and Mapping in Real-Time* ICCV 2011. <http://www.doc.ic.ac.uk/~rnewcomb/>

Engel, Schöps & Cremers: LSD-SLAM: Large-Scale Direct Monocular SLAM, ECCV 2014. [http://vision.in.tum.de/\\_media/spezial/bib/engel14eccv.pdf](http://vision.in.tum.de/_media/spezial/bib/engel14eccv.pdf)

4:67

## Alternative SLAM approach: Graph-based



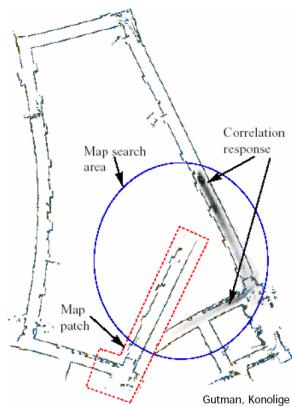
- Represent the previous trajectory as a graph
  - nodes = estimated positions & observations
  - edges = transition & step estimation based on scan matching
- Loop Closing: check if some nodes might coincide → new edges
- Classical Optimization:

The whole graph defines an optimization problem: Find poses that minimize sum of edge & node errors

4:68

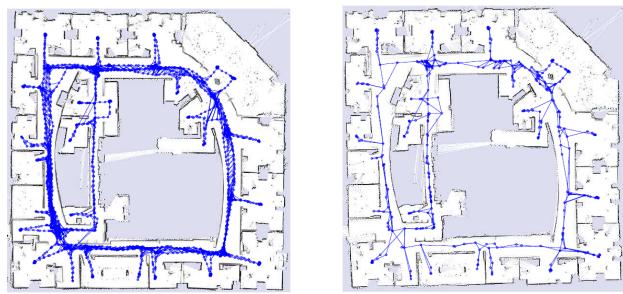
## Loop Closing Problem

(Doesn't explicitly exist in Particle Filter methods: If particles cover the belief, then “data association” solves the “loop closing problem”)



4:69

## Graph-based SLAM



*Life-long Map Learning for Graph-based SLAM Approaches in Static Environments* Kretzschmar, Grisetti, Stachniss

4:70

## SLAM code

- Graph-based and grid map methods:

<http://openslam.org/>

- Visual SLAM

see previous links

e.g. <http://ewokrampage.wordpress.com/>

4:71

## 5 Kinematics

### Course overview

- Mobile robots

*goal: localize and map yourself*

State estimation, Bayes filter, odometry, particle filter, Kalman filter, Bayes smoothing, SLAM, joint Bayes filter, EKF SLAM, particle SLAM, graph-based SLAM

- Kinematics & Dynamics

*goal: orchestrate joint movements for desired movement in task spaces*

Kinematic map, Jacobian, optimality principle of inverse kinematics, singularities, configuration/operational/null space, multiple simultaneous tasks, special task variables, trajectory interpolation, motion profiles; 1D point mass, damping & oscillation, PID, general dynamic systems, Newton-Euler, joint space control, reference trajectory following, optimal operational space control

- Planning & optimization

*goal: planning around obstacles, optimizing trajectories*

Path finding vs. trajectory optimization, local vs. global, Dijkstra, Probabilistic Roadmaps, Rapidly Exploring Random Trees, differential constraints, metrics; trajectory optimization, general cost function, task variables, transition costs, gradient methods, 2nd order methods, Dynamic Programming

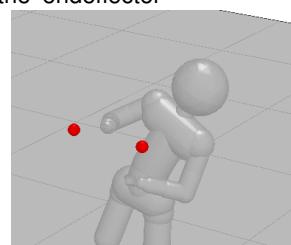
- Control Theory

*theory on designing optimal controllers*

Topics in control theory, optimal control, HJB equation, infinite horizon case, Linear-Quadratic optimal control, Riccati equations (differential, algebraic, discrete-time), controllability, stability, eigenvalue analysis, Lyapunov function

### Basic motion generation problem

- Move all joints in a coordinated way so that the endeffector



makes a desired movement

01-kinematics: ./x.exe -mode 2/3/4

### Outline

- Kinematics:  $\phi: q \mapsto y$

- Inverse Kinematics:  $y^* \mapsto q^* = \operatorname{argmin}_q \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2$
- Basic motion heuristics: Motion profiles

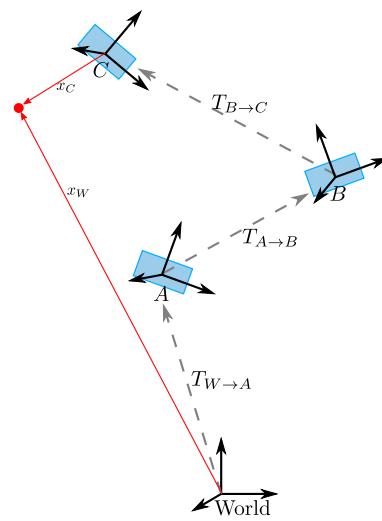
- Additional things to know
  - Many simultaneous task variables
  - Singularities, null space

5:3

### 5.1 Kinematics

5:4

#### Composition of transforms

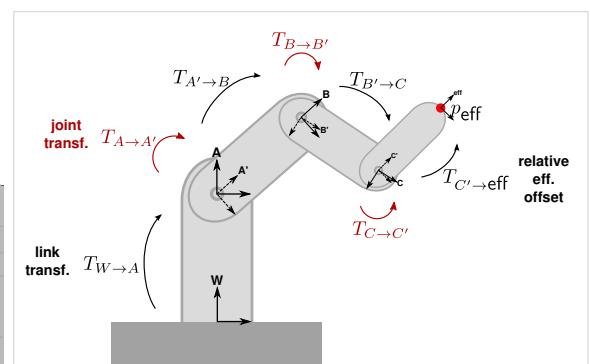


$$T_{W \rightarrow C} = T_{W \rightarrow A} T_{A \rightarrow B} T_{B \rightarrow C}$$

$$x^W = T_{W \rightarrow A} T_{A \rightarrow B} T_{B \rightarrow C} x^C$$

5:5

#### Kinematic structure



- A *kinematic structure* is a graph (usually tree or chain) of rigid **links** and **joints**

$$T_{W \rightarrow \text{eff}}(q) = T_{W \rightarrow A} T_{A \rightarrow A'}(q) T_{A' \rightarrow B} T_{B \rightarrow B'}(q) T_{B' \rightarrow C} T_{C \rightarrow C'}(q) T_{C' \rightarrow \text{eff}}$$

5:6

## Joint types

- Joint transformations:  $T_{A \rightarrow A'}(q)$  depends on  $q \in \mathbb{R}^n$

revolute joint: joint angle  $q \in \mathbb{R}$  determines rotation about  $x$ -axis:

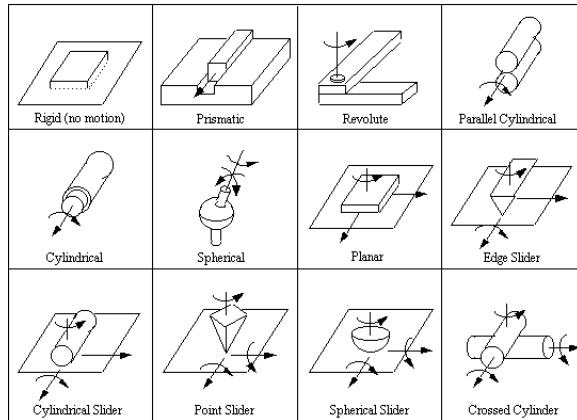
$$T_{A \rightarrow A'}(q) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(q) & -\sin(q) & 0 \\ 0 & \sin(q) & \cos(q) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

prismatic joint: offset  $q \in \mathbb{R}$  determines translation along  $x$ -axis:

$$T_{A \rightarrow A'}(q) = \begin{pmatrix} 1 & 0 & 0 & q \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

others: screw (1dof), cylindrical (2dof), spherical (3dof), universal (2dof)

5:7



5:8

## Kinematic Map

- For any joint angle vector  $q \in \mathbb{R}^n$  we can compute  $T_{W \rightarrow \text{eff}}(q)$  by *forward chaining* of transformations

$T_{W \rightarrow \text{eff}}(q)$  gives us the *pose* of the endeffector in the world frame

- In general, a kinematic map is *any* (differentiable) mapping

$$\phi : q \mapsto y$$

that maps to *some arbitrary feature*  $y \in \mathbb{R}^d$  of the joint vector  $q \in \mathbb{R}^n$

## Kinematic Map

- The three most important examples for a *kinematic map*  $\phi$  are
  - A position  $v$  on the endeffector transformed to world coordinates:

$$\phi_{\text{eff},v}^{\text{pos}}(q) = T_{W \rightarrow \text{eff}}(q) v \in \mathbb{R}^3$$

- A direction  $v \in \mathbb{R}^3$  attached to the endeffector in world coordinates:

$$\phi_{\text{eff},v}^{\text{vec}}(q) = R_{W \rightarrow \text{eff}}(q) v \in \mathbb{R}^3$$

Where  $R_{A \rightarrow B}$  is the rotation in  $T_{A \rightarrow B}$ .

- The (quaternion) orientation  $u \in \mathbb{R}^4$  of the endeffector:

$$\phi_{\text{eff}}^{\text{quat}}(q) = R_{W \rightarrow \text{eff}}(q) \in \mathbb{R}^4$$

- See the technical reference later for more kinematic maps, especially *relative* position, direction and quaternion maps.

5:10

## Jacobian

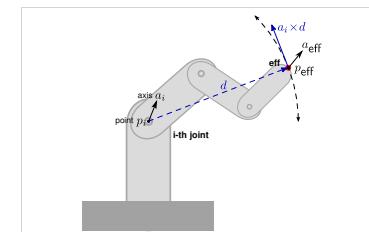
- When we change the joint angles,  $\delta q$ , how does the effector position change,  $\delta y$ ?
- Given the kinematic map  $y = \phi(q)$  and its Jacobian  $J(q) = \frac{\partial \phi}{\partial q}(q)$ , we have:

$$\delta y = J(q) \delta q$$

$$J(q) = \frac{\partial}{\partial q} \phi(q) = \begin{pmatrix} \frac{\partial \phi_1(q)}{\partial q_1} & \frac{\partial \phi_1(q)}{\partial q_2} & \dots & \frac{\partial \phi_1(q)}{\partial q_n} \\ \frac{\partial \phi_2(q)}{\partial q_1} & \frac{\partial \phi_2(q)}{\partial q_2} & \dots & \frac{\partial \phi_2(q)}{\partial q_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi_d(q)}{\partial q_1} & \frac{\partial \phi_d(q)}{\partial q_2} & \dots & \frac{\partial \phi_d(q)}{\partial q_n} \end{pmatrix} \in \mathbb{R}^{d \times n}$$

5:11

## Jacobian for a rotational degree of freedom

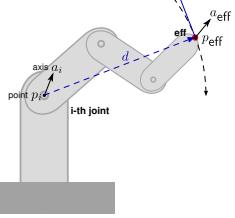


- Assume the  $i$ -th joint is located at  $p_i = t_{W \rightarrow i}(q)$  and has rotation axis  $a_i = R_{W \rightarrow i}(q) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$
- We consider an infinitesimal variation  $\delta q_i \in \mathbb{R}$  of the  $i$ th joint and see how an endeffector position  $p_{\text{eff}} = \phi_{\text{eff},v}^{\text{pos}}(q)$  and attached vector  $a_{\text{eff}} = \phi_{\text{eff},v}^{\text{vec}}(q)$  change.

5:12

5:9

## Jacobian for a rotational degree of freedom



Consider a variation  $\delta q_i$   
→ the whole sub-tree rotates

$$\begin{aligned}\delta p_{\text{eff}} &= [a_i \times (p_{\text{eff}} - p_i)] \delta q_i \\ \delta a_{\text{eff}} &= [a_i \times a_{\text{eff}}] \delta q_i\end{aligned}$$

⇒ Position Jacobian:

$$J_{\text{eff},v}^{\text{pos}}(q) = \begin{pmatrix} [p_{\text{eff}} - p_1] \\ [a_1 \times (p_{\text{eff}} - p_2)] \\ [a_2 \times (p_{\text{eff}} - p_3)] \\ \dots \\ [a_n \times (p_{\text{eff}} - p_n)] \end{pmatrix} \in \mathbb{R}^{3n \times n} \quad J_{\text{eff},v}^{\text{vec}}(q) = \begin{pmatrix} a_{\text{eff}} \\ [a_1 \times a_{\text{eff}}] \\ [a_2 \times a_{\text{eff}}] \\ \dots \\ [a_n \times a_{\text{eff}}] \end{pmatrix} \in \mathbb{R}^{3n \times n}$$

5:13

## Jacobian for general degrees of freedom

- Every degree of freedom  $q_i$  generates (infinitesimally, at a given  $q$ )
  - a rotation around axis  $a_i$  at point  $p_i$
  - and/or a translation along the axis  $b_i$

For instance:

- the DOF of a hinge joint just creates a rotation around  $a_i$  at  $p_i$
- the DOF of a prismatic joint creates a translation along  $b_i$
- the DOF of a rolling cylinder creates rotation and translation
- the first DOF of a cylindrical joint generates a translation, its second DOF a translation

- We can compute all Jacobians from knowing  $a_i$ ,  $p_i$  and  $b_i$  for all DOFs (in the current configuration  $q \in \mathbb{R}^n$ )

5:14

## 5.2 Inverse Kinematics

5:15

### Inverse Kinematics problem

- Generally, the aim is to find a robot configuration  $q$  such that  $\phi(q) = y^*$
- Iff  $\phi$  is invertible

$$q^* = \phi^{-1}(y^*)$$

- But in general,  $\phi$  will not be invertible:

1) The pre-image  $\phi^{-1}(y^*)$  may be empty: No configuration can generate the desired  $y^*$

2) The pre-image  $\phi^{-1}(y^*)$  may be large: many configurations can generate the desired  $y^*$

5:16

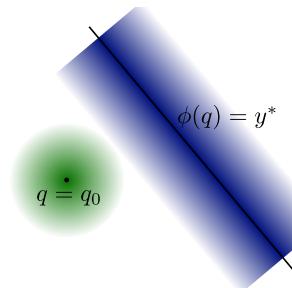
## Inverse Kinematics as optimization problem

- We formalize the inverse kinematics problem as an optimization problem

$$q^* = \underset{q}{\operatorname{argmin}} \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2$$

- The 1st term ensures that we find a configuration even if  $y^*$  is not exactly reachable

The 2nd term disambiguates the configurations if there are many  $\phi^{-1}(y^*)$



5:17

## Inverse Kinematics as optimization problem

$$q^* = \underset{q}{\operatorname{argmin}} \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2$$

- The formulation of IK as an optimization problem is very powerful and has many nice properties
- We will be able to take the limit  $C \rightarrow \infty$ , enforcing exact  $\phi(q) = y^*$  if possible
- Non-zero  $C^{-1}$  and  $W$  corresponds to a regularization that ensures numeric stability
- Classical concepts can be derived as special cases:
  - Null-space motion
  - regularization; singularity robustness
  - multiple tasks
  - hierarchical tasks

5:18

## Solving Inverse Kinematics

- The obvious choice of optimization method for this problem is Gauss-Newton, using the Jacobian of  $\phi$
- We first describe just one step of this, which leads to the classical equations for inverse kinematics using the local Jacobian...

5:19

## Solution using the local linearization

- When using the local linearization of  $\phi$  at  $q_0$ ,

$$\phi(q) \approx y_0 + J(q - q_0), \quad y_0 = \phi(q_0)$$

- We can derive the optimum as

$$\begin{aligned} f(q) &= \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2 \\ &= \|y_0 - y^* + J(q - q_0)\|_C^2 + \|q - q_0\|_W^2 \\ \frac{\partial}{\partial q} f(q) &= 0^\top = 2(y_0 - y^* + J(q - q_0))^\top C J + 2(q - q_0)^\top W \\ J^\top C (y^* - y_0) &= (J^\top C J + W)(q - q_0) \end{aligned}$$

$$q^* = q_0 + J^\sharp(y^* - y_0)$$

with  $J^\sharp = (J^\top C J + W)^{-1} J^\top C = W^{-1} J^\top (J W^{-1} J^\top + C^{-1})^{-1}$  (*Woodbury identity*)

- For  $C \rightarrow \infty$  and  $W = \mathbf{I}$ ,  $J^\sharp = J^\top (J J^\top)^{-1}$  is called *pseudo-inverse*
- $W$  generalizes the metric in  $q$ -space
- $C$  regularizes this pseudo-inverse (see later section on singularities)

5:20

$$= q' + J^\sharp(y^* - y') + (I - J^\sharp J) h, \quad h = q_0 - q' \quad (3)$$

- What if we want to find the *exact* (local) optimum? E.g. what if we want to compute a big step (where  $q^*$  will be remote from  $q$ ) and we cannot rely only on the local linearization approximation?

- Iterate equation (1) (optionally with a step size  $< 1$  to ensure convergence) by setting the point  $y'$  of linearization to the current  $q^*$
- This is equivalent to the Gauss-Newton algorithm

5:23

## Where are we?

- We've derived a basic motion generation principle in robotics from
  - an understanding of robot geometry & kinematics
  - a basic notion of optimality
- In the remainder:
  - Discussion of classical concepts
  - Heuristic motion profiles for simple trajectory generation
  - Extension to multiple task variables

5:24

## Discussion of classical concepts

- Singularity and singularity-robustness
- Nullspace, task/operational space, joint space
- “inverse kinematics”  $\leftrightarrow$  “motion rate control”

5:25

## Singularity

- In general: A matrix  $J$  **singular**  $\iff \text{rank}(J) < d$ 
  - rows of  $J$  are linearly dependent
  - dimension of image is  $< d$
  - $\delta y = J \delta q \Rightarrow$  dimensions of  $\delta y$  limited
  - Intuition: arm fully stretched
- Implications:
  - $\det(J J^\top) = 0$
  - $\rightarrow$  pseudo-inverse  $J^\top (J J^\top)^{-1}$  is ill-defined!
  - $\rightarrow$  inverse kinematics  $\delta q = J^\top (J J^\top)^{-1} \delta y$  computes “infinite” steps!

- **Singularity robust pseudo inverse**  $J^\top (J J^\top + \epsilon \mathbf{I})^{-1}$

The term  $\epsilon \mathbf{I}$  is called **regularization**

- Recall our general solution (for  $W = \mathbf{I}$ )

$$J^\sharp = J^\top (J J^\top + C^{-1})^{-1}$$

is already singularity robust

## “Small step” application

- This approximate solution to IK makes sense
  - if the local linearization of  $\phi$  at  $q_0$  is “good”
  - if  $q_0$  and  $q^*$  are close
- This equation is therefore typically used to iteratively compute small steps in configuration space

$$q_{t+1} = q_t + J^\sharp(y_{t+1}^* - \phi(q_t))$$

where the target  $y_{t+1}^*$  moves smoothly with  $t$

5:21

## Example: Iterating IK to follow a trajectory

- Assume initial posture  $q_0$ . We want to reach a desired endeff position  $y^*$  in  $T$  steps:

**Input:** initial state  $q_0$ , desired  $y^*$ , methods  $\phi^{\text{pos}}$  and  $J^{\text{pos}}$

**Output:** trajectory  $q_{0:T}$

```

1: Set  $y_0 = \phi^{\text{pos}}(q_0)$  // starting endeff position
2: for  $t = 1 : T$  do
3:    $y \leftarrow \phi^{\text{pos}}(q_{t-1})$  // current endeff position
4:    $J \leftarrow J^{\text{pos}}(q_{t-1})$  // current endeff Jacobian
5:    $\hat{y} \leftarrow y_0 + (t/T)(y^* - y_0)$  // interpolated endeff target
6:    $q_t = q_{t-1} + J^\sharp(\hat{y} - y)$  // new joint positions
7:   Command  $q_t$  to all robot motors and compute all
    $T_{W \rightarrow i}(q_t)$ 
8: end for

```

01-kinematics: ./x.exe -mode 2/3

- Why does this not follow the interpolated trajectory  $\hat{y}_{0:T}$  exactly?
  - What happens if  $T = 1$  and  $y^*$  is far?

5:22

## Two additional notes

- What if we linearize at some arbitrary  $q'$  instead of  $q_0$ ?

$$\phi(q) \approx y' + J(q - q'), \quad y' = \phi(q')$$

$$q^* = \underset{q}{\operatorname{argmin}} \|\phi(q) - y^*\|_C^2 + \|q - q' + (q' - q_0)\|_W^2$$

5:26

## Null/task/operational/joint/configuration spaces

- The space of all  $q \in \mathbb{R}^n$  is called **joint/configuration space**

The space of all  $y \in \mathbb{R}^d$  is called **task/operational space**

Usually  $d < n$ , which is called **redundancy**

- For a desired endeffector state  $y^*$  there exists a whole manifold (assuming  $\phi$  is smooth) of joint configurations  $q$ :

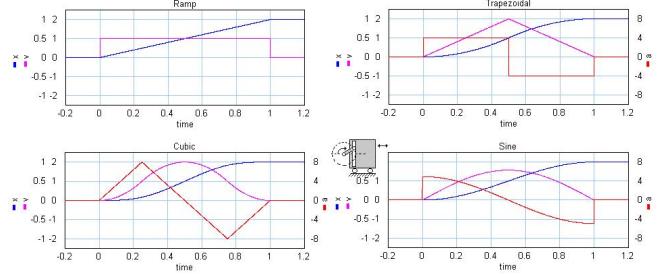
$$\text{nullspace}(y^*) = \{q \mid \phi(q) = y^*\}$$

- We have

$$\begin{aligned}\delta q &= \underset{q}{\operatorname{argmin}} \|q - a\|_W^2 + \|Jq - \delta y\|_C^2 \\ &= J^\# \delta y + (\mathbf{I} - J^\# J)a, \quad J^\# = W^{-1} J^\top (JW^{-1} J^\top + C^{-1})^{-1}\end{aligned}$$

In the limit  $C \rightarrow \infty$  it is guaranteed that  $J\delta q = \delta y$  (we are exactly on the manifold). The term  $a$  introduces additional “nullspace motion”.

5:27



The sine profile  $x_t = x_0 + \frac{1}{2}[1 - \cos(\pi t/T)](x_T - x_0)$  is a compromise for low max-acceleration and max-velocity

Taken from [http://www.20sim.com/webhelp/toolboxes/mechatronics\\_toolbox/motion\\_profile\\_wizard/motionprofiles.htm](http://www.20sim.com/webhelp/toolboxes/mechatronics_toolbox/motion_profile_wizard/motionprofiles.htm)

5:30

## Motion profiles

- Generally, let's define a motion profile as a mapping

$$\text{MP} : [0, 1] \mapsto [0, 1]$$

with  $\text{MP}(0) = 0$  and  $\text{MP}(1) = 1$  such that the interpolation is given as

$$x_t = x_0 + \text{MP}(t/T) (x_T - x_0)$$

## Inverse Kinematics and Motion Rate Control

Some clarification of concepts:

- The notion “kinematics” describes the mapping  $\phi : q \mapsto y$ , which usually is a many-to-one function.
- The notion “inverse kinematics” in the strict sense describes some mapping  $g : y \mapsto q$  such that  $\phi(g(y)) = y$ , which usually is non-unique or ill-defined.
- In practice, one often refers to  $\delta q = J^\# \delta y$  as **inverse kinematics**.
- When iterating  $\delta q = J^\# \delta y$  in a control cycle with time step  $\tau$  (typically  $\tau \approx 1 - 10$  msec), then  $\dot{y} = \delta y/\tau$  and  $\dot{q} = \delta q/\tau$  and  $\dot{q} = J^\# \dot{y}$ . Therefore the control cycle effectively controls the endeffector velocity—this is why it is called **motion rate control**.

5:28

## Heuristic motion profiles

5:29

## Heuristic motion profiles

- Assume initially  $x = 0, \dot{x} = 0$ . After 1 second you want  $x = 1, \dot{x} = 0$ .

How do you move from  $x = 0$  to  $x = 1$  in one second?

- For example

$$\text{MP}_{\text{ramp}}(s) = s$$

$$\text{MP}_{\text{sin}}(s) = \frac{1}{2}[1 - \cos(\pi s)]$$

5:31

## Joint space interpolation

- Optimize a desired final configuration  $q_T$ :

Given a desired final task value  $y_T$ , optimize a final joint state  $q_T$  to minimize the function

$$f(q_T) = \|q_T - q_0\|_{W/T}^2 + \|y_T - \phi(q_T)\|_C^2$$

– The metric  $\frac{1}{T}W$  is consistent with  $T$  cost terms with step metric  $W$ .  
– In this optimization,  $q_T$  will end up remote from  $q_0$ . So we need to iterate Gauss-Newton, as described on slide 5.2.

- Compute  $q_{0:T}$  as interpolation between  $q_0$  and  $q_T$ :

Given the initial configuration  $q_0$  and the final  $q_T$ , interpolate on a straight line with a motion profile. E.g.,

$$q_t = q_0 + \text{MP}(t/T) (q_T - q_0)$$

5:32

## Task space interpolation

- 1) Compute  $y_{0:T}$  as interpolation between  $y_0$  and  $y_T$ :

Given a initial task value  $y_0$  and a desired final task value  $y_T$ , interpolate on a straight line with a motion profile. E.g,

$$y_t = y_0 + \text{MP}(t/T) (y_T - y_0)$$

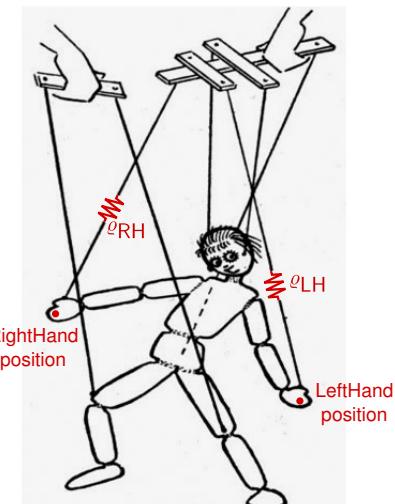
- 2) Project  $y_{0:T}$  to  $q_{0:T}$  using inverse kinematics:

Given the task trajectory  $y_{0:T}$ , compute a corresponding joint trajectory  $q_{0:T}$  using inverse kinematics

$$q_{t+1} = q_t + J^\dagger(y_{t+1} - \phi(q_t))$$

(As steps are small, we should be ok with just using this local linearization.)

5:33



5:37

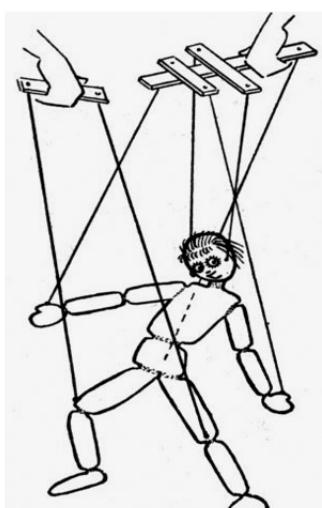
peg-in-a-hole demo

5:34

## 5.3 Multiple tasks

5:35

### Multiple tasks



5:36

### Multiple tasks

- Assume we have  $m$  simultaneous tasks; for each task  $i$  we have:

- a kinematic map  $\phi_i : \mathbb{R}^n \rightarrow \mathbb{R}^{d_i}$
- a current value  $\phi_i(q_t)$
- a desired value  $y_i^*$
- a precision  $\varrho_i$  (equiv. to a task cost metric  $C_i = \varrho_i \mathbf{I}$ )

- Each task contributes a term to the objective function

$$q^* = \underset{q}{\operatorname{argmin}} \|q - q_0\|_W^2 + \varrho_1 \|\phi_1(q) - y_1^*\|^2 + \varrho_2 \|\phi_2(q) - y_2^*\|^2 + \dots$$

which we can also write as

$$q^* = \underset{q}{\operatorname{argmin}} \|q - q_0\|_W^2 + \|\Phi(q)\|^2$$

$$\text{where } \Phi(q) := \begin{pmatrix} \sqrt{\varrho_1} (\phi_1(q) - y_1^*) \\ \sqrt{\varrho_2} (\phi_2(q) - y_2^*) \\ \vdots \end{pmatrix} \in \mathbb{R}^{\sum_i d_i}$$

5:38

### Multiple tasks

- We can “pack” together all tasks in one “big task”  $\Phi$ .

Example: We want to control the 3D position of the left hand and of the right hand. Both are “packed” to one 6-dimensional task vector which becomes zero if both tasks are fulfilled.

- The big  $\Phi$  is scaled/normalized in a way that
  - the desired value is always zero
  - the cost metric is  $\mathbf{I}$

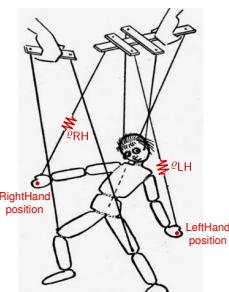
### Multiple tasks

- Using the local linearization of  $\Phi$  at  $q_0$ ,  $J = \frac{\partial \Phi(q_0)}{\partial q}$ , the optimum is

$$\begin{aligned} q^* &= \underset{q}{\operatorname{argmin}} \|q - q_0\|_W^2 + \|\Phi(q)\|^2 \\ &\approx q_0 - (J^\top J + W)^{-1} J^\top \Phi(q_0) = q_0 - J^\# \Phi(q_0) \end{aligned}$$

5:39

## Multiple tasks



- We learnt how to “puppeteer a robot”
- We can handle many task variables (but specifying their precisions  $\varrho_i$  becomes cumbersome...)
- In the remainder:
  - Classical limit of “hierarchical IK” and nullspace motion
  - What are interesting task variables?

5:40

## Hierarchical IK & nullspace motion

- In the classical view, tasks should be executed *exactly*, which means taking the limit  $\varrho_i \rightarrow \infty$  in some prespecified hierarchical order.
- We can rewrite the solution in a way that allows for such a hierarchical limit:
- One task plus “nullspace motion”:

$$\begin{aligned} f(q) &= \|q - a\|_W^2 + \varrho_1 \|J_1 q - y_1\|^2 \\ q^* &= [W + \varrho_1 J_1^\top J_1]^{-1} [W a + \varrho_1 J_1^\top y_1] \\ &= J_1^\# y_1 + (\mathbf{I} - J_1^\# J_1) a \\ J_1^\# &= (W/\varrho_1 + J_1^\top J_1)^{-1} J_1^\top = W^{-1} J_1^\top (J_1 W^{-1} J_1^\top + \mathbf{I}/\varrho_1)^{-1} \end{aligned}$$

- Two tasks plus nullspace motion:

$$\begin{aligned} f(q) &= \|q - a\|_W^2 + \varrho_1 \|J_1 q - y_1\|^2 + \varrho_2 \|J_2 q - y_2\|^2 \\ q^* &= J_1^\# y_1 + (\mathbf{I} - J_1^\# J_1) [J_2^\# y_2 + (\mathbf{I} - J_2^\# J_2) a] \\ J_2^\# &= (W/\varrho_2 + J_2^\top J_2)^{-1} J_2^\top = W^{-1} J_2^\top (J_2 W^{-1} J_2^\top + \mathbf{I}/\varrho_2)^{-1} \end{aligned}$$

- etc...

5:41

## Hierarchical IK & nullspace motion

- The previous slide did nothing but rewrite the nice solution  $q^* = -J^\# \Phi(q_0)$  (for the “big”  $\Phi$ ) in a strange hierarchical way that allows to “see” nullspace projection
- The benefit of this hierarchical way to write the solution is that one can take the hierarchical limit  $\varrho_i \rightarrow \infty$  and retrieve classical hierarchical IK
- The drawbacks are:
  - It is somewhat ugly
  - In practise, I would recommend regularization in any case (for numeric stability). Regularization corresponds to NOT taking the full limit  $\varrho_i \rightarrow \infty$ . Then the hierarchical way to write the solution is unnecessary. (However, it points to a “hierarchical regularization”, which might be numerically more robust for very small regularization?)

- The general solution allows for arbitrary blending of tasks

5:42

## Reference: interesting task variables

The following slides will define 10 different types of task variables.

This is meant as a reference and to give an idea of possibilities...

5:43

## Position

Position of some point attached to link $i$	
dimension	$d = 3$
parameters	link index $i$ , point offset $v$
kin. map	$\phi_{iv}^{\text{pos}}(q) = T_{W \rightarrow i} v$
Jacobian	$J_{iv}^{\text{pos}}(q)_{\cdot k} = [k \prec i] a_k \times (\phi_{iv}^{\text{pos}}(q) - p_k)$

Notation:

- $a_k, p_k$  are axis and position of joint  $k$
- $[k \prec i]$  indicates whether joint  $k$  is between root and link  $i$
- $J_{\cdot k}$  is the  $k$ th column of  $J$

5:44

## Vector

Vector attached to link $i$	
dimension	$d = 3$
parameters	link index $i$ , attached vector $v$
kin. map	$\phi_{iv}^{\text{vec}}(q) = R_{W \rightarrow i} v$
Jacobian	$J_{iv}^{\text{vec}}(q) = A_i \times \phi_{iv}^{\text{vec}}(q)$

Notation:

- $A_i$  is a matrix with columns  $(A_i)_{\cdot k} = [k \prec i] a_k$  containing the joint axes or zeros
- the short notation “ $A \times p$ ” means that each column in  $A$  takes the cross-product with  $p$ .

5:45

## Relative position

Position of a point on link $i$ relative to point on link $j$	
dimension	$d = 3$
parameters	link indices $i, j$ , point offset $v$ in $i$ and $w$ in $j$
kin. map	$\phi_{iv jw}^{\text{pos}}(q) = R_j^{-1}(\phi_{iv}^{\text{pos}} - \phi_{jw}^{\text{pos}})$
Jacobian	$J_{iv jw}^{\text{pos}}(q) = R_j^{-1}[J_{iv}^{\text{pos}} - J_{jw}^{\text{pos}} - A_j \times (\phi_{iv}^{\text{pos}} - \phi_{jw}^{\text{pos}})]$

Derivation:

For  $y = Rp$  the derivative w.r.t. a rotation around axis  $a$  is  $y' = Rp' + R'p = Rp' + a \times Rp$ . For  $y = R^{-1}p$  the derivative is  $y' = R^{-1}p' - R^{-1}(R')R^{-1}p = R^{-1}(p' - a \times p)$ . (For details see <http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/3d-geometry.pdf>)

5:46

## Relative vector

Vector attached to link $i$ relative to link $j$	
dimension	$d = 3$
parameters	link indices $i, j$ , attached vector $v$ in $i$
kin. map	$\phi_{iv j}^{\text{vec}}(q) = R_j^{-1} \phi_{iv}^{\text{vec}}$
Jacobian	$J_{iv j}^{\text{vec}}(q) = R_j^{-1} [J_{iv}^{\text{vec}} - A_j \times \phi_{iv}^{\text{vec}}]$

5:47

## Center of gravity

Center of gravity of the whole kinematic structure	
dimension	$d = 3$
parameters	(none)
kin. map	$\phi^{\text{cog}}(q) = \sum_i \text{mass}_i \phi_{ic_i}^{\text{pos}}$
Jacobian	$J^{\text{cog}}(q) = \sum_i \text{mass}_i J_{ic_i}^{\text{pos}}$

 $c_i$  denotes the center-of-mass of link  $i$  (in its own frame)

5:52

## Alignment

Alignment of a vector attached to link $i$ with a reference $v^*$	
dimension	$d = 1$
parameters	link index $i$ , attached vector $v$ , world reference $v^*$
kin. map	$\phi_{iv}^{\text{align}}(q) = v^{*\top} \phi_{iv}^{\text{vec}}$
Jacobian	$J_{iv}^{\text{align}}(q) = v^{*\top} J_{iv}^{\text{vec}}$

Note:  $\phi^{\text{align}} = 1 \leftrightarrow \text{align}$     $\phi^{\text{align}} = -1 \leftrightarrow \text{anti-align}$     $\phi^{\text{align}} = 0 \leftrightarrow \text{orthog.}$ 

5:48

## Relative Alignment

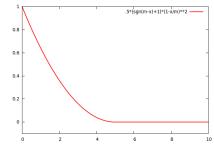
Alignment a vector attached to link $i$ with vector attached to $j$	
dimension	$d = 1$
parameters	link indices $i, j$ , attached vectors $v, w$
kin. map	$\phi_{iv jw}^{\text{align}}(q) = (\phi_{jw}^{\text{vec}})^{\top} \phi_{iv}^{\text{vec}}$
Jacobian	$J_{iv jw}^{\text{align}}(q) = (\phi_{jw}^{\text{vec}})^{\top} J_{iv}^{\text{vec}} + \phi_{iv}^{\text{vec}\top} J_{jw}^{\text{vec}}$

5:49

## Joint limits

Penetration of joint limits	
dimension	$d = 1$
parameters	joint limits $q_{\text{low}}, q_{\text{hi}}$ , margin $m$
kin. map	$\phi_{\text{limits}}(q) = \frac{1}{m} \sum_{i=1}^n [m - q_i + q_{\text{low}}]^+ + [m + q_i - q_{\text{hi}}]^+$
Jacobian	$J_{\text{limits}}(q)_{1,i} = -\frac{1}{m} [m - q_i + q_{\text{low}} > 0] + \frac{1}{m} [m + q_i - q_{\text{hi}} > 0]$

$$[x]^+ = x > 0 ? x : 0 \quad [\dots]: \text{indicator function}$$



5:50

## Collision limits

Penetration of collision limits	
dimension	$d = 1$
parameters	margin $m$
kin. map	$\phi_{\text{col}}(q) = \frac{1}{m} \sum_{k=1}^K [m -  p_k^a - p_k^b ]^+$
Jacobian	$J_{\text{col}}(q) = \frac{1}{m} \sum_{k=1}^K [m -  p_k^a - p_k^b  > 0] (-J_{p_k^a}^{\text{pos}} + J_{p_k^b}^{\text{pos}})^{\top} \frac{p_k^a - p_k^b}{ p_k^a - p_k^b }$

A collision detection engine returns a set  $\{(a, b, p^a, p^b)\}_{k=1}^K$  of potential collisions between link  $a_k$  and  $b_k$ , with nearest points  $p_k^a$  on  $a$  and  $p_k^b$  on  $b$ .

5:51

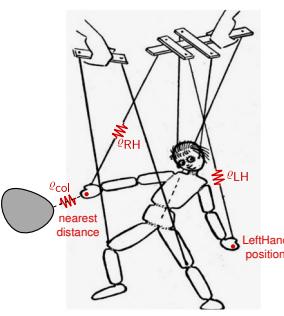
## Homing

The joint angles themselves	
dimension	$d = n$
parameters	(none)
kin. map	$\phi_{\text{qitself}}(q) = q$
Jacobian	$J_{\text{qitself}}(q) = \mathbf{I}_n$

Example: Set the target  $y^* = 0$  and the precision  $\varrho$  very low → this task describes posture comfortness in terms of deviation from the joints' zero position. In the classical view, it induces "nullspace motion".

5:53

## Task variables – conclusions



- There is much space for creativity in defining task variables! Many are extensions of  $\phi^{\text{pos}}$  and  $\phi^{\text{vec}}$  and the Jacobians combine the basic Jacobians.
- What the *right* task variables are to design/describe motion is a very hard problem! In what task space do humans control their motion? Possible to learn from data ("task space retrieval") or perhaps via Reinforcement Learning.
- In practice: Robot motion design (including grasping) may require cumbersome hand-tuning of such task variables.

5:54

## 6 Dynamics

### Kinematic

instantly change joint velocities  $\dot{q}$ :

$$\delta q_t \stackrel{!}{=} J^\dagger (y^* - \phi(q_t))$$

accounts for kinematic coupling of joints but **ignores inertia, forces, torques**

gears, **stiff**, all of industrial robots



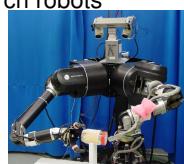
### Dynamic

instantly change joint torques  $u$ :

$$u \stackrel{!}{=} ?$$

accounts for dynamic coupling of joints and full Newtonian physics

future robots, **compliant**, few research robots



6:1

### When velocities cannot be changed/set arbitrarily

- Examples:

- An air plane flying: You cannot command it to hold still in the air, or to move straight up.
- A car: you cannot command it to move side-wards.
- Your arm: you cannot command it to throw a ball with arbitrary speed (force limits).
- A *torque controlled* robot: You cannot command it to instantly change velocity (infinite acceleration/torque).

- What all examples have in common:

- One can set **controls**  $u_t$  (air plane's control stick, car's steering wheel, your muscles activations, torque/voltage/current send to a robot's motors)
- But these controls only indirectly influence the **dynamics of state**

$$x_{t+1} = f(x_t, u_t)$$

6:2

### Dynamics

- The dynamics of a system describes how the controls  $u_t$  influence the change-of-state of the system

$$x_{t+1} = f(x_t, u_t)$$

- The notation  $x_t$  refers to the *dynamic state* of the system: e.g., joint positions *and velocities*  $x_t = (q_t, \dot{q}_t)$ .
- $f$  is an arbitrary function, often smooth

6:3

### Outline

- We start by discussing a **1D point mass** for 3 reasons:
  - The most basic force-controlled system with inertia
  - We can introduce and understand **PID control**
  - The behavior of a point mass under PID control is a *reference* that we can also follow with arbitrary dynamic robots (if the dynamics are known)

- We discuss computing the dynamics of general robotic systems
  - Euler-Lagrange equations
  - Euler-Newton method
- We derive the dynamic equivalent of inverse kinematics:
  - operational space control

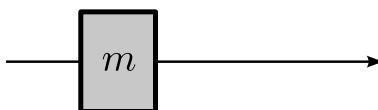
6:4

### 6.1 PID and a 1D point mass

6:5

#### The dynamics of a 1D point mass

- Start with simplest possible example: 1D point mass (no gravity, no friction, just a single mass)



- The state  $x(t) = (q(t), \dot{q}(t))$  is described by:
  - position  $q(t) \in \mathbb{R}$
  - velocity  $\dot{q}(t) \in \mathbb{R}$
- The controls  $u(t)$  is the force we apply on the mass point

- The system dynamics is:

$$\ddot{q}(t) = u(t)/m$$

6:6

#### 1D point mass – proportional feedback

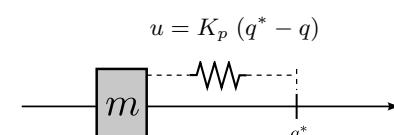
- Assume current position is  $q$ .

The goal is to move it to the position  $q^*$ .

What can we do?

- **Idea 1:**

*"Always pull the mass towards the goal  $q^*$ :"*



6:7

## 1D point mass – proportional feedback

- What's the effect of this control law?

$$m \ddot{q} = u = K_p (q^* - q)$$

$q = q(t)$  is a function of time, this is a second order differential equation

- Solution: assume  $q(t) = a + be^{\omega t}$

(a "non-imaginary" alternative would be  $q(t) = a + b e^{-\lambda t} \cos(\omega t)$ )

$$\begin{aligned} m b \omega^2 e^{\omega t} &= K_p q^* - K_p a - K_p b e^{\omega t} - K_d b \omega e^{\omega t} \\ (m b \omega^2 + K_p b) e^{\omega t} &= K_p (q^* - a) \\ \Rightarrow (m b \omega^2 + K_p b) &= 0 \wedge (q^* - a) = 0 \\ \Rightarrow \omega &= i \sqrt{K_p/m} \\ q(t) &= q^* + b e^{i \sqrt{K_p/m} t} \end{aligned}$$

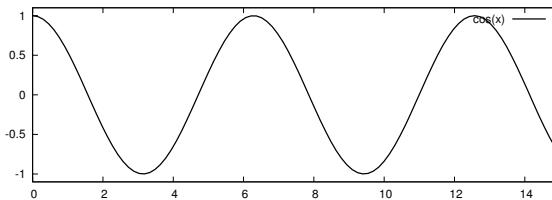
This is an oscillation around  $q^*$  with amplitude  $b = q(0) - q^*$  and frequency  $\sqrt{K_p/m}$

6:8

## 1D point mass – proportional feedback

$$\begin{aligned} m \ddot{q} &= u = K_p (q^* - q) \\ q(t) &= q^* + b e^{i \sqrt{K_p/m} t} \end{aligned}$$

Oscillation around  $q^*$  with amplitude  $b = q(0) - q^*$  and frequency  $\sqrt{K_p/m}$



6:9

## 1D point mass – derivative feedback

- Idea 2

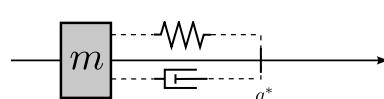
"Pull less, when we're heading the right direction already:"

"Damp the system:"

$$u = K_p(q^* - q) + K_d(\dot{q}^* - \dot{q})$$

$\dot{q}^*$  is a desired goal velocity

For simplicity we set  $\dot{q}^* = 0$  in the following.



6:10

## 1D point mass – derivative feedback

- What's the effect of this control law?

$$m \ddot{q} = u = K_p(q^* - q) + K_d(0 - \dot{q})$$

- Solution: again assume  $q(t) = a + be^{\omega t}$

$$\begin{aligned} m b \omega^2 e^{\omega t} &= K_p q^* - K_p a - K_p b e^{\omega t} - K_d b \omega e^{\omega t} \\ (m b \omega^2 + K_d b \omega + K_p b) e^{\omega t} &= K_p (q^* - a) \\ \Rightarrow (m \omega^2 + K_d \omega + K_p) &= 0 \wedge (q^* - a) = 0 \\ \Rightarrow \omega &= \frac{-K_d \pm \sqrt{K_d^2 - 4mK_p}}{2m} \\ q(t) &= q^* + b e^{\omega t} \end{aligned}$$

The term  $-\frac{K_d}{2m}$  in  $\omega$  is real  $\leftrightarrow$  exponential decay (damping)

6:11

## 1D point mass – derivative feedback

$$q(t) = q^* + b e^{\omega t}, \quad \omega = \frac{-K_d \pm \sqrt{K_d^2 - 4mK_p}}{2m}$$

- Effect of the second term  $\sqrt{K_d^2 - 4mK_p}/2m$  in  $\omega$ :

$$\begin{aligned} K_d^2 < 4mK_p &\Rightarrow \omega \text{ has imaginary part} \\ &\text{oscillating with frequency } \sqrt{K_p/m - K_d^2/4m^2} \\ q(t) &= q^* + b e^{-K_d/2m t} e^{i \sqrt{K_p/m - K_d^2/4m^2} t} \\ K_d^2 > 4mK_p &\Rightarrow \omega \text{ real} \\ &\text{strongly damped} \\ K_d^2 = 4mK_p &\Rightarrow \text{second term zero} \\ &\text{only exponential decay} \end{aligned}$$

6:12

## 1D point mass – derivative feedback

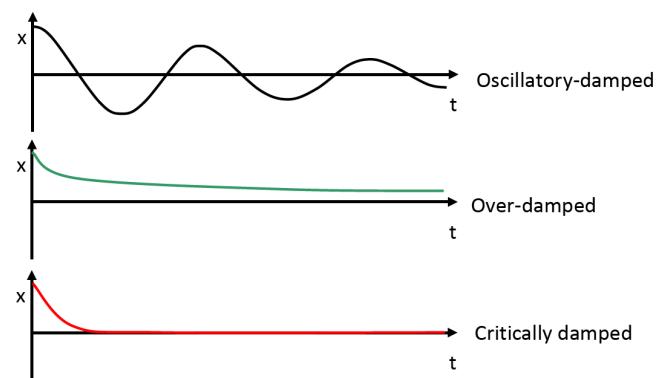


illustration from O. Brock's lecture

6:13

## 1D point mass – derivative feedback

Alternative parameterization:

Instead of the gains  $K_p$  and  $K_d$  it is sometimes more intuitive to set the

- wave length  $\lambda = \frac{1}{\omega_0} = \frac{1}{\sqrt{K_p/m}}$ ,  $K_p = m/\lambda^2$ ,  $\omega_0 = T/(2\pi)$

- damping ratio  $\xi = \frac{K_d}{\sqrt{4mK_p}} = \frac{\lambda K_d}{2m}$ ,  $K_d = 2m\xi/\lambda$

$\xi > 1$ : over-damped

$\xi = 1$ : critically damped

$\xi < 1$ : oscillatory-damped

$$q(t) = q^* + be^{-\xi t/\lambda} e^{i\sqrt{1-\xi^2} t/\lambda}$$

6:14

## 1D point mass – integral feedback

- Idea 3

"Pull if the position error accumulated large in the past:"

$$u = K_p(q^* - q) + K_d(\dot{q}^* - \dot{q}) + K_i \int_{s=0}^t (q^*(s) - q(s)) ds$$

- This is not a linear ODE w.r.t.  $x = (q, \dot{q})$ .

However, when we extend the state to  $x = (q, \dot{q}, e)$  we have the ODE

$$\dot{q} = \dot{q}$$

$$\ddot{q} = u/m = K_p/m(q^* - q) + K_d/m(\dot{q}^* - \dot{q}) + K_i/m e$$

$$\dot{e} = q^* - q$$

(no explicit discussion here)

6:15

## 1D point mass – PID control

$$u = K_p(q^* - q) + K_d(\dot{q}^* - \dot{q}) + K_i \int_{s=0}^t (q^* - q(s)) ds$$

- PID control

– Proportional Control ("Position Control")

$$u \propto K_p(q^* - q)$$

– Derivative Control ("Damping")

$$u \propto K_d(\dot{q}^* - \dot{q}) \quad (\dot{x}^* = 0 \rightarrow \text{damping})$$

– Integral Control ("Steady State Error")

$$u \propto K_i \int_{s=0}^t (q^*(s) - q(s)) ds$$

6:16

## Controlling a 1D point mass – lessons learnt

- Proportional and derivative feedback (PD control) are like adding a spring and damper to the point mass

- PD control is a *linear control law*

$$(q, \dot{q}) \mapsto u = K_p(q^* - q) + K_d(\dot{q}^* - \dot{q})$$

(linear in the *dynamic system state*  $x = (q, \dot{q})$ )

- With such linear control laws we can design approach trajectories (by tuning the gains)
  - but no optimality principle behind such motions

6:17

## 6.2 Dynamics of mechanical systems

6:18

### Two ways to derive dynamics equations for mechanical systems

- The Euler-Lagrange equation,  $L = L(t, q(t), \dot{q}(t))$ ,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = u$$

Used when you want to derive analytic equations of motion ("on paper")

- The Newton-Euler recursion (and related algorithms)

$$f_i = m\ddot{v}_i, \quad u_i = I_i\dot{w} + w \times Iw$$

Algorithms that "propagate" forces through a kinematic tree and numerically compute the *inverse dynamics*  $u = \text{NE}(q, \dot{q}, \ddot{q})$  or *forward dynamics*  $\ddot{q} = f(q, \dot{q}, u)$ .

6:19

### The Euler-Lagrange equation

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = u$$

- $L(q, \dot{q})$  is called **Lagrangian** and defined as

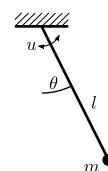
$$L = T - U$$

where  $T$ =kinetic energy and  $U$ =potential energy.

- $q$  is called generalized coordinate – any coordinates such that  $(q, \dot{q})$  describes the state of the system. Joint angles in our case.
- $u$  are external forces

6:20

### Example: A pendulum



- Generalized coordinates: angle  $q = (\theta)$
- Kinematics:
  - velocity of the mass:  $v = (l\dot{\theta} \cos \theta, 0, l\dot{\theta} \sin \theta)$
  - angular velocity of the mass:  $w = (0, -\dot{\theta}, 0)$
- Energies:

$$T = \frac{1}{2}mv^2 + \frac{1}{2}w^\top Iw = \frac{1}{2}(ml^2 + I_2)\dot{\theta}^2, \quad U = -mgl \cos \theta$$

- Euler-Lagrange equation:

$$\begin{aligned} u &= \frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} \\ &= \frac{d}{dt} (ml^2 + I_2) \dot{\theta} + mgl \sin \theta = (ml^2 + I_2) \ddot{\theta} + mgl \sin \theta \end{aligned}$$

6:21

## The Euler-Lagrange equation

- How is this typically done?
- **First**, describe the *kinematics and Jacobians* for every link  $i$ :

$$(q, \dot{q}) \mapsto \{T_{W \rightarrow i}(q), v_i, w_i\}$$

Recall  $T_{W \rightarrow i}(q) = T_{W \rightarrow A} T_{A \rightarrow A'}(q) T_{A' \rightarrow B} T_{B \rightarrow B'}(q) \dots$

Further, we know that a link's velocity  $v_i = J_i \dot{q}$  can be described via its position Jacobian.

Similarly we can describe the link's angular velocity  $w_i = J_i^\omega \dot{q}$  as linear in  $\dot{q}$ .

- **Second**, formulate the kinetic energy

$$T = \sum_i \frac{1}{2} m_i v_i^2 + \frac{1}{2} w_i^\top I_i w_i = \sum_i \frac{1}{2} \dot{q}^\top M_i \dot{q}, \quad M_i = \begin{pmatrix} J_i \\ J_i^\omega \end{pmatrix}^\top \begin{pmatrix} m_i \mathbf{I}_3 & 0 \\ 0 & I_i \end{pmatrix} \begin{pmatrix} J_i \\ J_i^\omega \end{pmatrix}$$

where  $I_i = R_i \bar{I}_i R_i^\top$  and  $\bar{I}_i$  the inertia tensor in link coordinates

- **Third**, formulate the potential energies (typically independent of  $\dot{q}$ )

$$U = g m_i \text{height}(i)$$

- **Fourth**, compute the partial derivatives analytically to get something like

$$\underbrace{u}_{\text{control}} = \frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = \underbrace{M}_{\text{inertia}} \ddot{q} + \underbrace{\dot{M} \dot{q}}_{\text{Coriolis}} - \underbrace{\frac{\partial T}{\partial q}}_{\text{gravity}} + \underbrace{\frac{\partial U}{\partial q}}_{\text{gravity}}$$

which relates accelerations  $\ddot{q}$  to the forces

6:22

## Newton-Euler recursion

- An algorithm that computes the *inverse dynamics*

$$u = \text{NE}(q, \dot{q}, \ddot{q}^*)$$

by recursively computing force balance at each joint:

- **Newton's equation** expresses the force acting at the center of mass for an accelerated body:

$$f_i = m \dot{v}_i$$

- **Euler's equation** expresses the torque (=control) acting on a rigid body given an angular velocity and angular acceleration:

$$u_i = I_i \dot{w} + w \times I w$$

- **Forward recursion:** ( $\approx$  kinematics)

Compute (angular) velocities  $(v_i, w_i)$  and accelerations  $(\dot{v}_i, \dot{w}_i)$  for every link (via forward propagation; see geometry notes for details)

- **Backward recursion:**

For the leaf links, we now know the desired accelerations  $\ddot{q}^*$  and can compute the necessary joint torques. Recurse backward.

6:23

## Numeric algorithms for forward and inverse dynamics

- **Newton-Euler recursion:** very fast ( $O(n)$ ) method to compute inverse dynamics

$$u = \text{NE}(q, \dot{q}, \ddot{q}^*)$$

Note that we can use this algorithm to also compute

- gravity terms:  $u = \text{NE}(q, 0, 0) = G(q)$
- Coriolis terms:  $u = \text{NE}(q, \dot{q}, 0) = C(q, \dot{q}) \dot{q} + G(q)$
- column of Intertia matrix:  $u = \text{NE}(q, 0, e_i) = M(q) e_i$

- **Articulated-Body-Dynamics:** fast method ( $O(n)$ ) to compute forward dynamics  $\ddot{q} = f(q, \dot{q}, u)$

6:24

## Some last practical comments

- Use *energy conservation* to measure dynamic of physical simulation
- Physical simulation engines (developed for games):
  - ODE (Open Dynamics Engine)
  - Bullet (originally focussed on collision only)
  - Physx (Nvidia)

Differences of these engines to Lagrange, NE or ABD:

- Game engine can model much more: Contacts, tissues, particles, fog, etc
- (The way they model contacts looks ok but is somewhat fictional)
- On kinematic trees, NE or ABD are much more precise than game engines
- Game engines do not provide *inverse dynamics*,  $u = \text{NE}(q, \dot{q}, \ddot{q}^*)$

- Proper modelling of contacts is really really hard

6:25

## 6.3 Controlling a dynamic robot

- We previously learnt the effect of PID control on a 1D point mass
- Robots are not a 1D point mass
  - Neither is each joint a 1D point mass
  - Applying separate PD control in each joint neglects force coupling  
(Poor solution: Apply very high gains separately in each joint  $\leftrightarrow$  make joints stiff, as with gears.)

- However, knowing the robot dynamics we can transfer our understanding of PID control of a point mass to general systems

6:27

## General robot dynamics

- Let  $(q, \dot{q})$  be the dynamic state and  $u \in \mathbb{R}^n$  the controls (typically joint torques in each motor) of a robot
- Robot dynamics can generally be written as:

$$M(q) \ddot{q} + C(q, \dot{q}) \dot{q} + G(q) = u$$

$M(q) \in \mathbb{R}^{n \times n}$	is positive definite inertia matrix (can be inverted → forward simulation of dynamics)
$C(q, \dot{q}) \in \mathbb{R}^{n \times n}$	are the centripetal and coriolis forces
$G(q) \in \mathbb{R}^n$	are the gravitational forces
$u$	are the joint torques

(cf. to the Euler-Lagrange equation on slide 22)

- We often write more compactly:

$$M(q) \ddot{q} + F(q, \dot{q}) = u$$

6:28

## Controlling a general robot

- From now on we just assume that we have algorithms to efficiently compute  $M(q)$  and  $F(q, \dot{q})$  for any  $(q, \dot{q})$
- Inverse dynamics:** If we know the desired  $\ddot{q}^*$  for each joint,

$$u = M(q) \ddot{q}^* + F(q, \dot{q})$$

gives the necessary torques

- Forward dynamics:** If we know which torques  $u$  we apply, use

$$\ddot{q}^* = M(q)^{-1}(u - F(q, \dot{q}))$$

to simulate the dynamics of the system (e.g., using Runge-Kutta)

6:29

## Following a reference trajectory in joint space

- Where could we get the desired  $\ddot{q}^*$  from?

Assume we have a nice smooth **reference trajectory**  $q_{0:T}^{\text{ref}}$  (generated with some motion profile or alike), we can at each  $t$  read off the desired acceleration as

$$\ddot{q}_t^{\text{ref}} := \frac{1}{\tau}[(q_{t+1} - q_t)/\tau - (q_t - q_{t-1})/\tau] = (q_{t-1} + q_{t+1} - 2q_t)/\tau^2$$

However, tiny errors in acceleration will accumulate greatly over time! This is Instable!!

- Choose a desired acceleration  $\ddot{q}_t^*$  that implies a *PD-like behavior around the reference trajectory!*

$$\ddot{q}_t^* = \ddot{q}_t^{\text{ref}} + K_p(q_t^{\text{ref}} - q_t) + K_d(\dot{q}_t^{\text{ref}} - \dot{q}_t)$$

This is a standard and very convenient heuristic to track a reference trajectory when the robot dynamics are known: *All joints will exactly behave like a 1D point particle around the reference trajectory!*

6:30

## Following a reference trajectory in task space

- Recall the inverse kinematics problem:
  - We know the desired step  $\delta y^*$  (or velocity  $\dot{y}^*$ ) of the *end effector*.
  - Which step  $\delta q$  (or velocities  $\dot{q}$ ) should we make in the joints?
- Equivalent dynamic problem:
  - We know how the desired acceleration  $\ddot{y}^*$  of the *end effector*.
  - What controls  $u$  should we apply?

6:31

## Operational space control

- Inverse kinematics:

$$q^* = \underset{q}{\operatorname{argmin}} \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2$$

- Operational space control (one might call it “Inverse task space dynamics”):

$$u^* = \underset{u}{\operatorname{argmin}} \|\ddot{\phi}(q) - \ddot{y}^*\|_C^2 + \|u\|_H^2$$

6:32

## Operational space control

- We can derive the optimum perfectly analogous to inverse kinematics

We identify the minimum of a locally squared potential, using the local linearization (and approx.  $\ddot{J} = 0$ )

$$\ddot{\phi}(q) = \frac{d}{dt} \dot{\phi}(q) \approx \frac{d}{dt} (J\dot{q} + \dot{J}q) \approx J\ddot{q} + 2J\dot{q} = JM^{-1}(u - F) + 2J\dot{q}$$

We get

$$u^* = T^\# (\ddot{y}^* - 2J\dot{q} + TF)$$

$$\text{with } T = JM^{-1}, \quad T^\# = (T^\top CT + H)^{-1}T^\top C$$

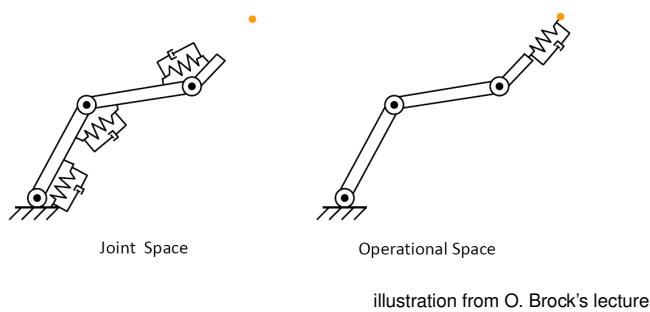
$$(C \rightarrow \infty \Rightarrow T^\# = H^{-1}T^\top(TH^{-1}T^\top)^{-1})$$

6:33

## Controlling a robot – operational space approach

- Where could we get the desired  $\ddot{y}^*$  from?
  - Reference trajectory**  $y_{0:T}^{\text{ref}}$  in operational space
  - PD-like behavior** in each operational space:

$$\ddot{y}_t^* = \ddot{y}_t^{\text{ref}} + K_p(y_t^{\text{ref}} - y_t) + K_d(\dot{y}_t^{\text{ref}} - \dot{y}_t)$$



- Operational space control: *Let the system behave as if we could directly “apply a 1D point mass behavior” to the endeffector*

6:34

## Multiple tasks

- Recall trick last time: we defined a “big kinematic map”  $\Phi(q)$  such that

$$q^* = \underset{q}{\operatorname{argmin}} \|q - q_0\|_W^2 + \|\Phi(q)\|^2$$

- Works analogously in the dynamic case:

$$u^* = \underset{u}{\operatorname{argmin}} \|u\|_H^2 + \|\Phi(q)\|^2$$

6:35

## What have we learned? What not?

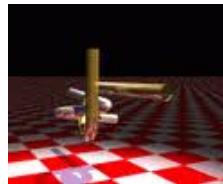
- More theory
  - Contacts → Inequality constraints on the dynamics
  - Switching dynamics (e.g. for walking)
  - Controlling contact forces
- Hardware limits
  - I think: the current success stories on highly dynamic robots are all anchored in novel hardware approaches

6:36

## 7 Path Planning

Path finding vs. trajectory optimization, local vs. global, Dijkstra, Probabilistic Roadmaps, Rapidly Exploring Random Trees, non-holonomic systems, car system equation, path-finding for non-holonomic systems, control-based sampling, Dubins curves

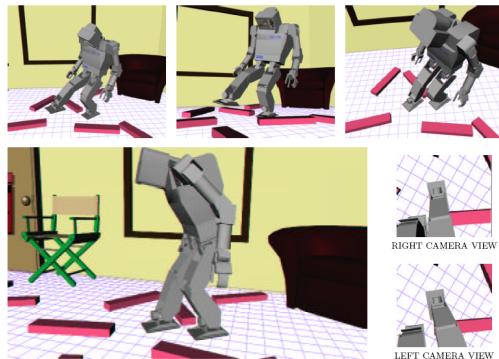
### Path finding examples



Alpha-Puzzle, solved with James Kuffner's RRTs

7:1

### Path finding examples



J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep Planning Among Obstacles for Biped Robots. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2001.

7:2

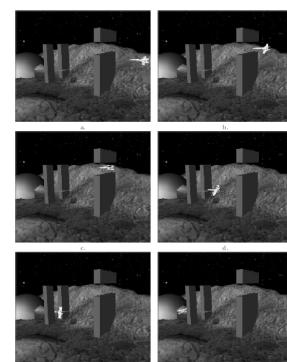
### Path finding examples



T. Bretl. Motion Planning of Multi-Limbed Robots Subject to Equilibrium Constraints: The Free-Climbing Robot Problem. International Journal of Robotics Research, 25(4):317-342, Apr 2006.

7:3

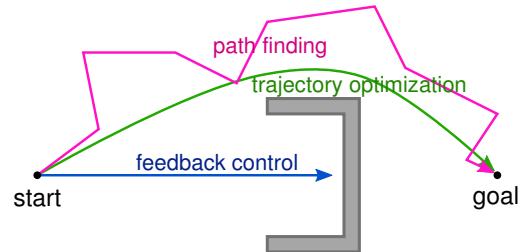
### Path finding examples



S. M. LaValle and J. J. Kuffner. Randomized Kinodynamic Planning. International Journal of Robotics Research, 20(5):378–400, May 2001.

7:4

### Feedback control, path finding, trajectory optim.



- Feedback Control: E.g.,  $q_{t+1} = q_t + J^\#(y^* - \phi(q_t))$
- Trajectory Optimization:  $\operatorname{argmin}_{q_{0:T}} f(q_{0:T})$
- Path Finding: Find some  $q_{0:T}$  with only valid configurations

7:5

### Outline

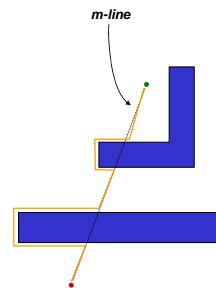
- **Really briefly:** Heuristics & Discretization (slides from Howie Choset's CMU lectures)
  - Bugs algorithm
  - Potentials to guide feedback control
  - Dijkstra
- **Sample-based Path Finding**
  - Probabilistic Roadmaps
  - Rapidly Exploring Random Trees
- **Non-holonomic systems**

7:6

### 7.1 Background

7:7

## A better bug?



### "Bug 2" Algorithm

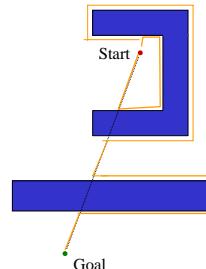
- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.
- 3) Leave the obstacle and continue toward the goal



8/61

7:8

## A better bug?



### "Bug 2" Algorithm

- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again ***closer to the goal***.
- 3) Leave the obstacle and continue toward the goal

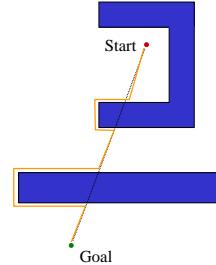


11/61

Better or worse than Bug1?

7:11

## A better bug?



### "Bug 2" Algorithm

- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.
- 3) Leave the obstacle and continue toward the goal



9/61

7:9

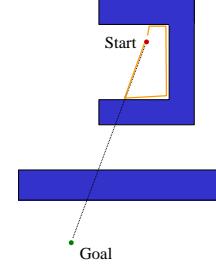
## BUG algorithms – conclusions

- Other variants: TangentBug, VisBug, RoverBug, WedgeBug, ...
- only 2D! (TangentBug has extension to 3D)
- Guaranteed convergence
- Still active research:  
K. Taylor and S.M. LaValle: *I-Bug: An Intensity-Based Bug Algorithm*

⇒ Useful for minimalistic, robust 2D goal reaching  
– not useful for finding paths in joint space

7:12

## A better bug?



### "Bug 2" Algorithm

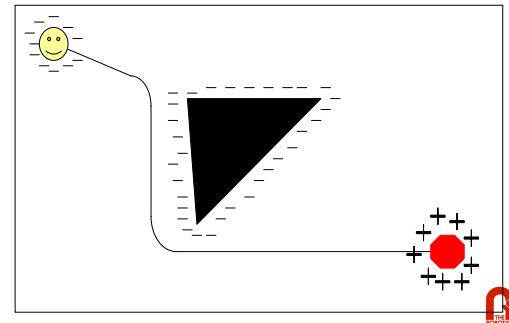
- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.
- 3) Leave the obstacle and continue toward the goal



10/61

7:10

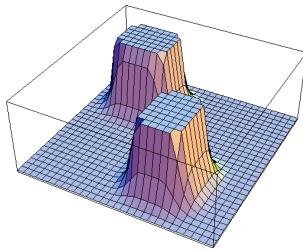
## Start-Goal Algorithm: Potential Functions



13/61

7:13

## Repulsive Potential



14/61

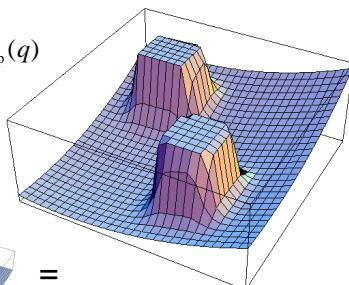
7:14

⇒ Does not solve locality problem of feedback control.

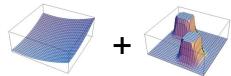
7:17

## Total Potential Function

$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$



$$F(q) = -\nabla U(q)$$



15/61

7:15

## The Wavefront in Action (Part 2)

- Now repeat with the modified cells
  - This will be repeated until no 0's are adjacent to cells with values  $\geq 2$ 
    - 0's will only remain when regions are unreachable

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0
3	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			



18/61

7:18

## The Wavefront in Action (Part 1)

- Starting with the goal, set all adjacent cells with "0" to the current cell + 1
  - 4-Point Connectivity or 8-Point Connectivity?
  - Your Choice – We'll use 8-Point Connectivity in our example

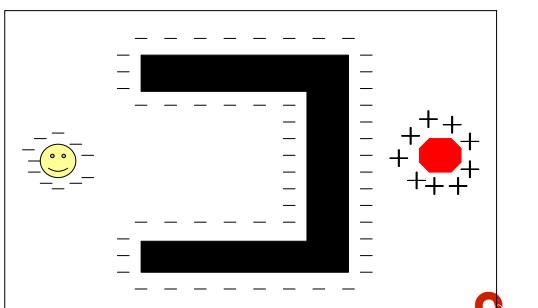
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			



19/61

7:19

## Local Minimum Problem with the Charge Analogy



16/61

7:16

## Potential fields – conclusions

- Very simple, therefore popular
- In our framework: Combining a goal (endeffector) task variable, with a constraint (collision avoidance) task variable; then using inv. kinematics is exactly the same as "Potential Fields"

## The Wavefront in Action (Part 2)

- Now repeat with the modified cells
  - This will be repeated until no 0's are adjacent to cells with values  $\geq 2$ 
    - 0's will only remain when regions are unreachable

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			



20/61

7:20

## The Wavefront in Action (Part 3)

- Repeat again...

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0
3	0	0	0	0	0	1	1	1	1	1	1	1	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



21/61

7:21

## The Wavefront in Action (Done)

- You're done
  - Remember, 0's should only remain if unreachable regions exist

7	18	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	1	1	1	1	1	1	1	1	6	6	6	6
3	17	16	15	14	1	1	1	1	1	1	1	1	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



24/61

7:24

## The Wavefront in Action (Part 4)

- And again...

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	1	1	1	1	1	1	6	6	6	6
3	0	0	0	0	0	1	1	1	1	1	1	1	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	6	5	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	6	5	4	3
0	0	0	0	0	0	0	0	0	0	0	0	0	6	5	4	3
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



22/61

7:22

## The Wavefront, Now What?

- To find the shortest path, according to your metric, simply always move toward a cell with a lower number
  - The numbers generated by the Wavefront planner are roughly proportional to their distance from the goal

Two possible shortest paths shown

7	18	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
6	17	17	16	15	14	13	12	11	10	9	8	7	7	7	7	7
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	1	1	1	1	1	1	1	1	6	6	6	6
3	17	16	15	14	1	1	1	1	1	1	1	1	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	11	10	9	8	7	6	5	4	3	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



25/61

7:25

## Dijkstra Algorithm

- Is efficient in **discrete domains**
  - Given start and goal node in an arbitrary graph
  - Incrementally label nodes with their distance-from-start
- Produces optimal (shortest) paths
- Applying this to continuous domains requires discretization
  - Grid-like discretization in high-dimensions is daunting! (*curse of dimensionality*)
  - What are other ways to “discretize” space more efficiently?

7:26

## The Wavefront in Action (Part 5)

- And again until...

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	7	7	7	7	7	7	7
4	0	0	0	0	0	1	1	1	1	1	1	1	6	6	6	6
3	0	0	0	0	0	1	1	1	1	1	1	1	5	5	5	5
2	0	0	0	0	0	0	0	0	0	7	6	5	4	4	4	4
1	0	0	0	0	0	0	0	0	0	7	6	5	4	3	3	3
0	0	0	0	0	0	0	0	0	0	7	6	5	4	3	2	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



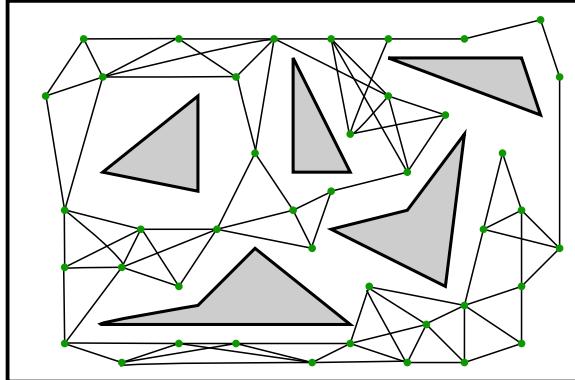
23/61

7:23

## 7.2 Sample-based Path Finding

7:27

## Probabilistic Road Maps



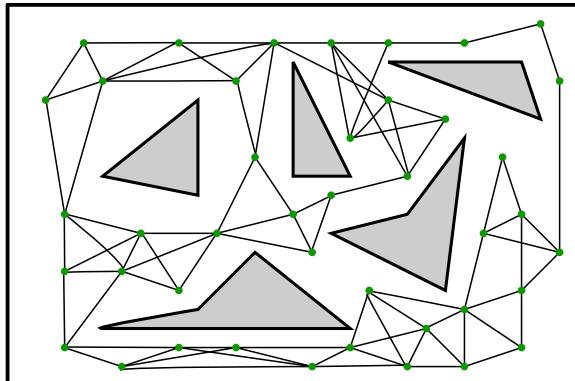
[Kavraki, Svestka, Latombe, Overmars, 95]

$q \in \mathbb{R}^n$  describes configuration

$Q_{\text{free}}$  is the set of configurations without collision

7:28

## Probabilistic Road Maps



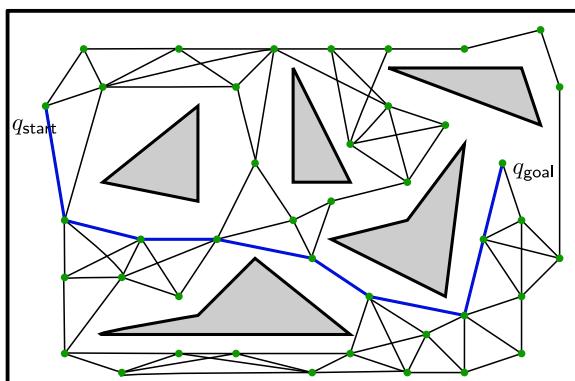
[Kavraki, Svestka, Latombe, Overmars, 95]

Probabilistic Road Map

- generates a graph  $G = (V, E)$  of configurations
  - such that configurations along each edges are  $\in Q_{\text{free}}$

7:29

## Probabilistic Road Maps



Given the graph, use (e.g.) Dijkstra to find path from  $q_{\text{start}}$  to  $q_{\text{goal}}$ .

7:30

## Probabilistic Road Maps – generation

**Input:** number  $n$  of samples, number  $k$  number of nearest neighbors

**Output:** PRM  $G = (V, E)$

```

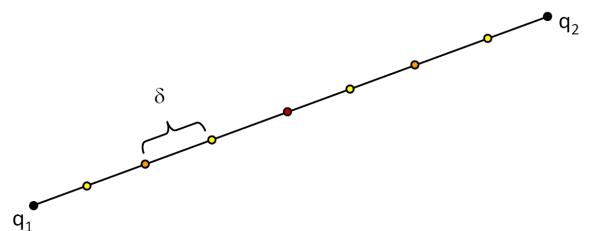
1: initialize  $V = \emptyset, E = \emptyset$ 
2: while  $|V| < n$  do           // find  $n$  collision free points  $q_i$ 
3:    $q \leftarrow$  random sample from  $Q$ 
4:   if  $q \in Q_{\text{free}}$  then  $V \leftarrow V \cup \{q\}$ 
5: end while
6: for all  $q \in V$  do // check if near points can be connected
7:    $N_q \leftarrow k$  nearest neighbors of  $q$  in  $V$ 
8:   for all  $q' \in N_q$  do
9:     if  $\text{path}(q, q') \in Q_{\text{free}}$  then  $E \leftarrow E \cup \{(q, q')\}$ 
10:    end for
11: end for

```

where  $\text{path}(q, q')$  is a local planner (easiest: straight line)

7:31

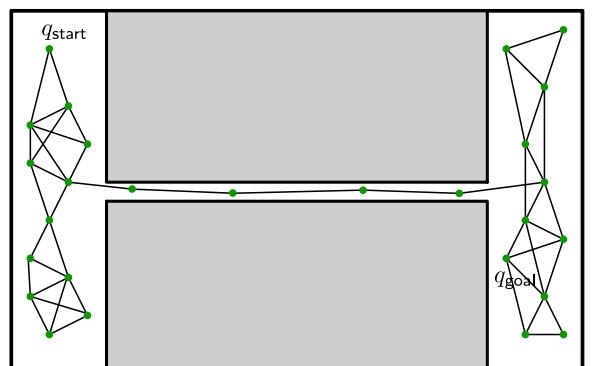
## Local Planner



tests collisions up to a specified resolution  $\delta$

7:32

## Problem: Narrow Passages



The smaller the gap (clearance  $\varrho$ ) the more unlikely to sample such points.

7:33

## PRM theory

(for uniform sampling in  $d$ -dim space)

- Let  $a, b \in Q_{\text{free}}$  and  $\gamma$  a path in  $Q_{\text{free}}$  connecting  $a$  and  $b$ .

Then the probability that PRM found the path after  $n$  samples is

$$P(\text{PRM-success} | n) \geq 1 - \frac{2|\gamma|}{\varrho} e^{-\sigma \varrho^{d_n}}$$

$$\sigma = \frac{|B_1|}{2^d |Q_{\text{free}}|}$$

$\varrho$  = clearance of  $\gamma$  (distance to obstacles)

(roughly: the exponential term are “volume ratios”)

(Ensures that RRTs can be extended to handling differential constraints.)

7:37

- This result is called *probabilistic complete* (one can achieve any probability with high enough  $n$ )
- For a given success probability,  $n$  needs to be exponential in  $d$

7:34

## Other PRM sampling strategies

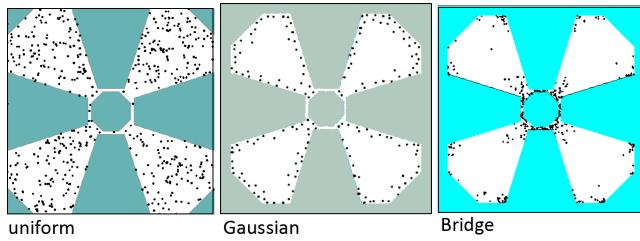


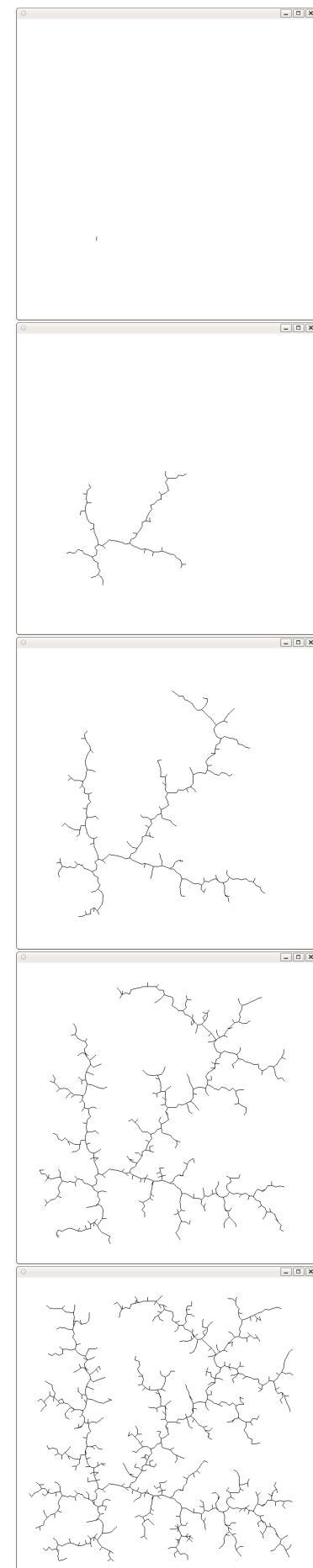
illustration from O. Brock's lecture

Gaussian:  $q_1 \sim \mathcal{U}; q_2 \sim \mathcal{N}(q_1, \sigma)$ ; if  $q_1 \in Q_{\text{free}}$  and  $q_2 \notin Q_{\text{free}}$ , add  $q_1$  (or vice versa).

Bridge:  $q_1 \sim \mathcal{U}; q_2 \sim \mathcal{N}(q_1, \sigma); q_3 = (q_1 + q_2)/2$ ; if  $q_1, q_2 \notin Q_{\text{free}}$  and  $q_3 \in Q_{\text{free}}$ , add  $q_3$ .

- Sampling strategy can be made more intelligent: “utility-based sampling”
- Connection sampling  
(once earlier sampling has produced connected components)

7:35



## Probabilistic Roadmaps – conclusions

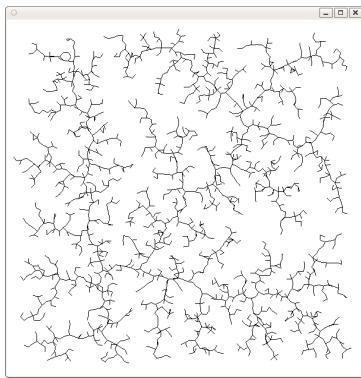
- Pros:
  - Algorithmically very simple
  - Highly explorative
  - Allows probabilistic performance guarantees
  - Good to answer many queries in an *unchanged* environment
- Cons:
  - Precomputation of exhaustive roadmap takes a long time (but not necessary for “Lazy PRMs”)

7:36

## Rapidly Exploring Random Trees

2 motivations:

- Single Query path finding: Focus computational efforts on paths for specific  $(q_{\text{start}}, q_{\text{goal}})$
- Use actually controllable DoFs to incrementally explore the search space: *control-based* path finding.



$n = 1$   $n = 100$   $n = 300$   $n = 600$   $n = 1000$   $n = 2000$

7:38



## Rapidly Exploring Random Trees

Simplest RRT with straight line local planner and step size  $\alpha$

**Input:**  $q_{\text{start}}$ , number  $n$  of nodes, stepsize  $\alpha$

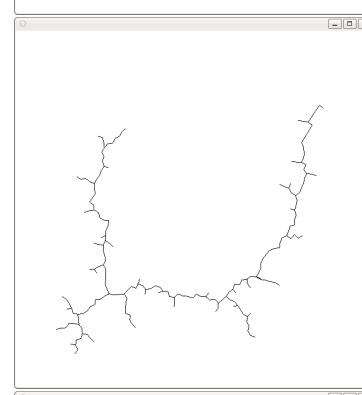
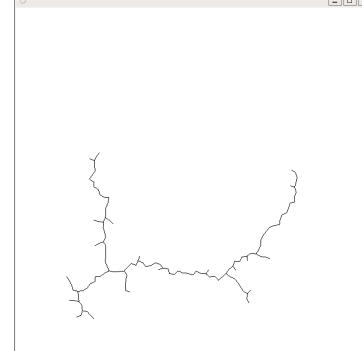
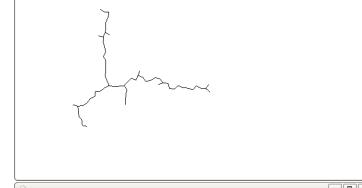
**Output:** tree  $T = (V, E)$

```

1: initialize  $V = \{q_{\text{start}}\}$ ,  $E = \emptyset$ 
2: for  $i = 0 : n$  do
3:    $q_{\text{target}} \leftarrow$  random sample from  $Q$ 
4:    $q_{\text{near}} \leftarrow$  nearest neighbor of  $q_{\text{target}}$  in  $V$ 
5:    $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{\|q_{\text{target}} - q_{\text{near}}\|} (q_{\text{target}} - q_{\text{near}})$ 
6:   if  $q_{\text{new}} \in Q_{\text{free}}$  then  $V \leftarrow V \cup \{q_{\text{new}}\}$ ,  $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$ 
7: end for

```

7:39



## Rapidly Exploring Random Trees

RRT growing directedly towards the goal

**Input:**  $q_{\text{start}}$ ,  $q_{\text{goal}}$ , number  $n$  of nodes, stepsize  $\alpha$ ,  $\beta$

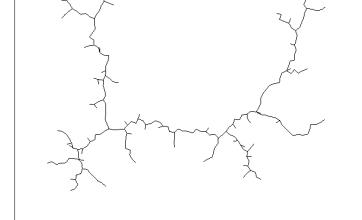
**Output:** tree  $T = (V, E)$

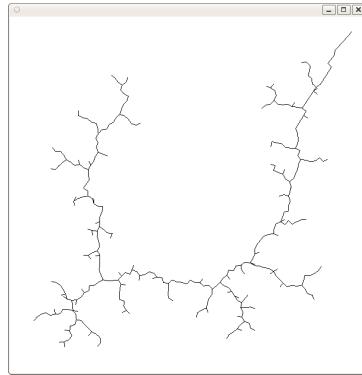
```

1: initialize  $V = \{q_{\text{start}}\}$ ,  $E = \emptyset$ 
2: for  $i = 0 : n$  do
3:   if  $\text{rand}(0, 1) < \beta$  then  $q_{\text{target}} \leftarrow q_{\text{goal}}$ 
4:   else  $q_{\text{target}} \leftarrow$  random sample from  $Q$ 
5:    $q_{\text{near}} \leftarrow$  nearest neighbor of  $q_{\text{target}}$  in  $V$ 
6:    $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{\|q_{\text{target}} - q_{\text{near}}\|} (q_{\text{target}} - q_{\text{near}})$ 
7:   if  $q_{\text{new}} \in Q_{\text{free}}$  then  $V \leftarrow V \cup \{q_{\text{new}}\}$ ,  $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$ 
8: end for

```

7:40



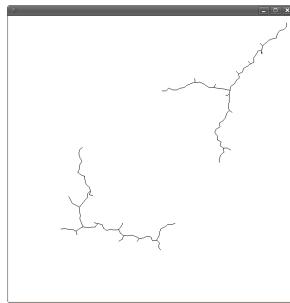


$n = 1 \ n = 100 \ n = 200 \ n = 300 \ n = 400 \ n = 500$

7:41

## Bi-directional search

- grow two trees starting from  $q_{\text{start}}$  and  $q_{\text{goal}}$



let one tree grow towards the other

(e.g., “choose  $q_{\text{new}}$  of  $T_1$  as  $q_{\text{target}}$  of  $T_2$ ”)

7:42

## Summary: RRTs

- Pros (shared with PRMs):
  - Algorithmically very simple
  - Highly explorative
  - Allows probabilistic performance guarantees
- Pros (beyond PRMs):
  - Focus computation on single query ( $q_{\text{start}}, q_{\text{goal}}$ ) problem
  - Trees from multiple queries can be merged to a roadmap
  - Can be extended to differential constraints (nonholonomic systems)
- To keep in mind (shared with PRMs):
  - The metric (for nearest neighbor selection) is sometimes critical
  - The local planner may be non-trivial

7:43

## References

Steven M. LaValle: *Planning Algorithms*, <http://planning.cs.uiuc.edu/>.

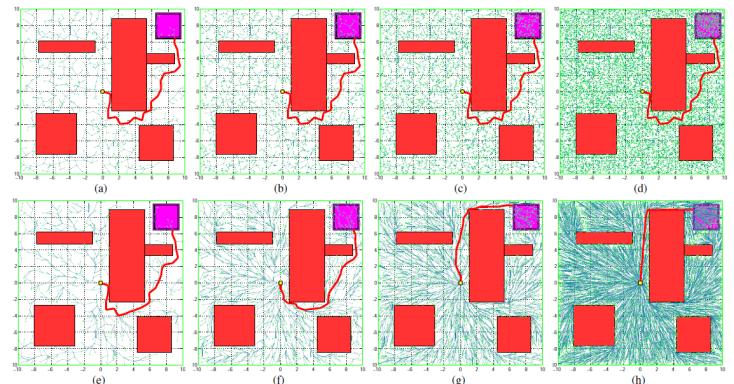
Choset et. al.: *Principles of Motion Planning*, MIT Press.

Latombe’s “motion planning” lecture, <http://robotics.stanford.edu/~latombe/cs326/2007/schedule.htm>

7:44

## RRT\*

Sertac Karaman & Emilio Frazzoli: Incremental sampling-based algorithms for optimal motion planning, arXiv 1005.0416 (2010).



Algorithm 4: ExtendRRT\*( $G, x$ )

```

1  $V' \leftarrow V; E' \leftarrow E;$ 
2  $x_{\text{nearest}} \leftarrow \text{Nearest}(G, x);$ 
3  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x);$ 
4 If  $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$  then
5    $V' \leftarrow V' \cup \{x_{\text{new}}\};$ 
6    $x_{\text{min}} \leftarrow x_{\text{nearest}};$ 
7    $X_{\text{near}} \leftarrow \text{Near}(G, x_{\text{new}}, |V|);$ 
8   for all  $x_{\text{near}} \in X_{\text{near}}$  do
9     if  $\text{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$  then
10        $c' \leftarrow \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}}));$ 
11       if  $c' < \text{Cost}(x_{\text{new}})$  then
12          $x_{\text{min}} \leftarrow x_{\text{near}},$ 
13    $E' \leftarrow E' \cup \{(x_{\text{min}}, x_{\text{new}})\};$ 
14   for all  $x_{\text{near}} \in X_{\text{near}} \setminus \{x_{\text{min}}\}$  do
15     if  $\text{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$  and
         $\text{Cost}(x_{\text{near}}) > \text{Cost}(x_{\text{new}}) + c(\text{Line}(x_{\text{new}}, x_{\text{near}}))$ 
        then
16        $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}});$ 
17        $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\};$ 
18        $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\};$ 
19
20 return  $G' = (V', E')$ 
```

7:45

## 7.3 Non-holonomic systems

7:46

### Non-holonomic systems

- We define a **nonholonomic system** as one with **differential constraints**:

$$\dim(u_t) < \dim(x_t)$$

$\Rightarrow$  Not all degrees of freedom are directly controllable

- Dynamic systems are an example!

- General definition of a differential constraint:

For any given state  $x$ , let  $U_x$  be the tangent space that is generated by controls:

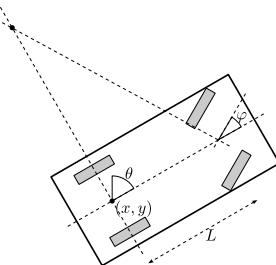
$$U_x = \{\dot{x} : \dot{x} = f(x, u), u \in U\}$$

(non-holonomic  $\iff \dim(U_x) < \dim(x)$ )

The elements of  $U_x$  are elements of  $T_x$  subject to additional differential constraints.

7:47

## Car example



$$\begin{aligned}\dot{x} &= v \cos \theta \\ \dot{y} &= v \sin \theta \\ \dot{\theta} &= (v/L) \tan \varphi \\ |\varphi| &< \Phi\end{aligned}$$

**State**  $q = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$     **Controls**  $u = \begin{pmatrix} v \\ \varphi \end{pmatrix}$

**System equation**  $\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v \cos \theta \\ v \sin \theta \\ (v/L) \tan \varphi \end{pmatrix}$

7:48

## Car example

- The car is a *non-holonomic* system: not all DoFs are controlled,  $\dim(u) < \dim(q)$

We have the *differential constraint*  $\dot{q}$ :

$$\dot{x} \sin \theta - \dot{y} \cos \theta = 0$$

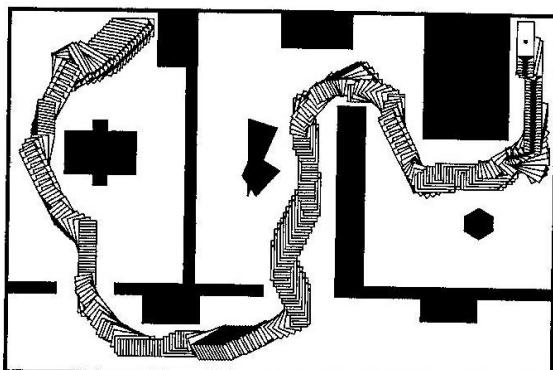
"A car cannot move directly lateral."

- Analogy to dynamic systems: Just like a car cannot instantly move sideways, a dynamic system cannot instantly change its position  $q$ : the current change in position is *constrained* by the current velocity  $\dot{q}$ .

7:49

## Path finding for a non-holonomic system

Could a car follow this trajectory?

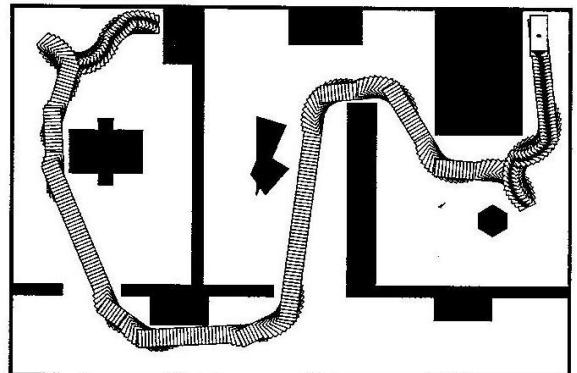


This is generated with a PRM in the state space  $q = (x, y, \theta)$  ignoring the differential constraint.

7:50

## Path finding with a non-holonomic system

This is a solution we would like to have:



The path respects **differential constraints**.

Each step in the path corresponds to setting certain controls.

7:51

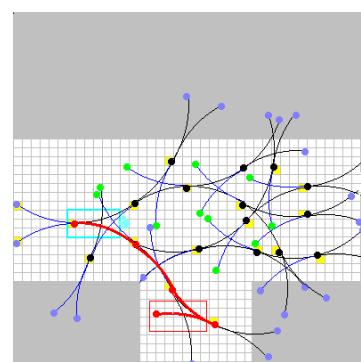
## Control-based sampling to grow a tree

- Control-based sampling: fulfills none of the nice exploration properties of RRTs, but fulfills the differential constraints:

- 1) Select a  $q \in T$  from tree of current configurations
- 2) Pick control vector  $u$  at random
- 3) Integrate equation of motion over short duration (picked at random or not)
- 4) If the motion is collision-free, add the endpoint to the tree

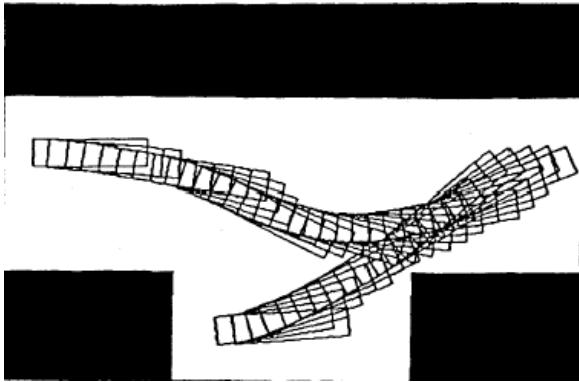
7:52

## Control-based sampling for the car



- 1) Select a  $q \in T$
- 2) Pick  $v, \phi$ , and  $\tau$
- 3) Integrate motion from  $q$
- 4) Add result if collision-free

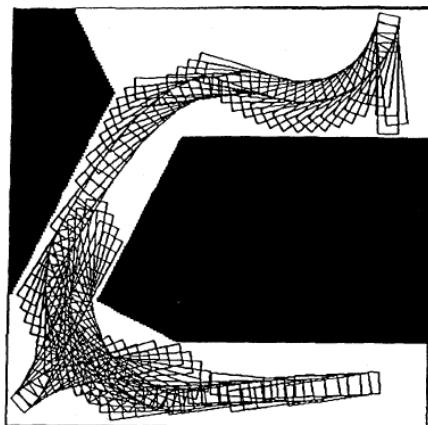
7:53



J. Barraquand and J.C. Latombe. Nonholonomic Multibody Robots: Controllability and Motion Planning in the Presence of Obstacles. Algorithmica, 10:121-155, 1993.

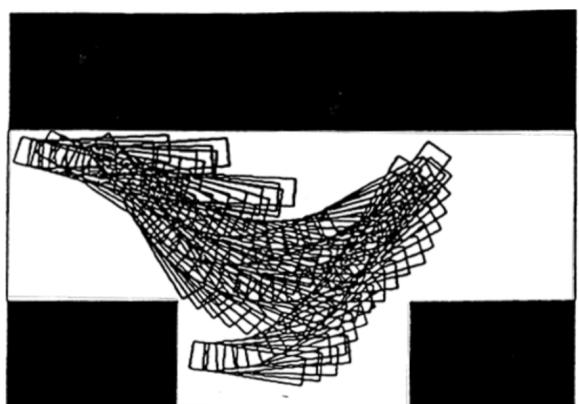
car parking

7:54



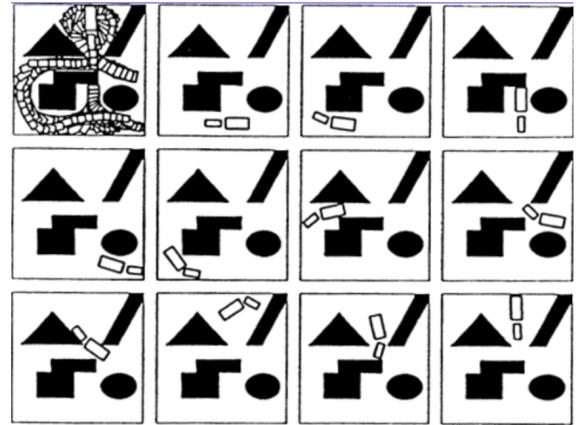
car parking

7:55



parking with only left-steering

7:56



with a trailer

7:57

## Better control-based exploration: RRTs revisited

- RRTs with differential constraints:

```

Input:  $q_{\text{start}}$ , number  $k$  of nodes, time interval  $\tau$ 
Output: tree  $T = (V, E)$ 
1: initialize  $V = \{q_{\text{start}}\}$ ,  $E = \emptyset$ 
2: for  $i = 0 : k$  do
3:    $q_{\text{target}} \leftarrow$  random sample from  $Q$ 
4:    $q_{\text{near}} \leftarrow$  nearest neighbor of  $q_{\text{target}}$  in  $V$ 
5:   use local planner to compute controls  $u$  that steer  $q_{\text{near}}$  towards  $q_{\text{target}}$ 
6:    $q_{\text{new}} \leftarrow q_{\text{near}} + \int_{t=0}^{\tau} \dot{q}(q, u) dt$ 
7:   if  $q_{\text{new}} \in Q_{\text{free}}$  then  $V \leftarrow V \cup \{q_{\text{new}}\}$ ,  $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$ 
8: end for

```

- Crucial questions:
- How measure *near* in nonholonomic systems?
- How find controls  $u$  to steer towards target?

7:58

## Configuration state metrics

Standard/Naive metrics:

- Comparing two 2D rotations/orientations  $\theta_1, \theta_2 \in SO(2)$ :
  - a) Euclidean metric between  $e^{i\theta_1}$  and  $e^{i\theta_2}$
  - b)  $d(\theta_1, \theta_2) = \min\{|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|\}$
- Comparing two configurations  $(x, y, \theta)_{1,2}$  in  $\mathbb{R}^2$ :  
Euclidian metric on  $(x, y, e^{i\theta})$
- Comparing two 3D rotations/orientations  $r_1, r_2 \in SO(3)$ :  
Represent both orientations as unit-length quaternions  $r_1, r_2 \in \mathbb{R}^4$ :  
 $d(r_1, r_2) = \min\{|r_1 - r_2|, |r_1 + r_2|\}$   
where  $|\cdot|$  is the Euclidean metric.  
(Recall that  $r_1$  and  $-r_1$  represent exactly the same rotation.)

- Ideal metric:

Optimal cost-to-go between two states  $x_1$  and  $x_2$ :

- Use optimal trajectory cost as metric
- This is as hard to compute as the original problem, of course!!  
→ Approximate, e.g., by neglecting obstacles.

7:59

## Side story: Dubins curves

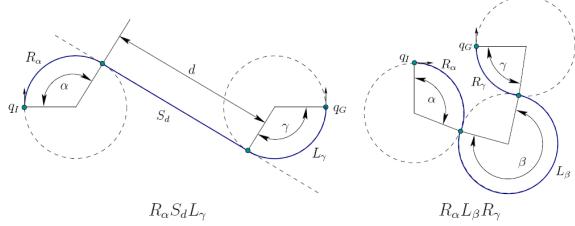
- Dubins car: constant velocity, and steer  $\varphi \in [-\Phi, \Phi]$
- Neglecting obstacles, there are only **six** types of trajectories that connect any configuration  $\in \mathbb{R}^2 \times \mathbb{S}^1$ :
 
$$\{LRL, RLR, LSL, LSR, RSL, RSR\}$$
- annotating durations of each phase:

$$\{L_\alpha R_\beta L_\gamma, R_\alpha L_\beta R_\gamma, L_\alpha S_d L_\gamma, L_\alpha S_d R_\gamma, R_\alpha S_d L_\gamma, R_\alpha S_d R_\gamma\}$$

with  $\alpha \in [0, 2\pi), \beta \in (\pi, 2\pi), d \geq 0$

7:60

## Side story: Dubins curves



→ By testing all six types of trajectories for  $(q_1, q_2)$  we can define a Dubins metric for the RRT – and use the Dubins curves as controls!

- **Reeds-Shepp curves** are an extension for cars which can drive back.  
(includes 46 types of trajectories, good metric for use in RRTs for cars)

7:61

## 8 Path Optimization – briefly

*Path finding vs. trajectory optimization, local vs. global, Dijkstra, Probabilistic Roadmaps, Rapidly Exploring Random Trees, non-holonomic systems, car system equation, path-finding for non-holonomic systems, control-based sampling, Dubins curves*

### Choice of optimizer

$$\begin{aligned} \min_{x_{0:T}} \quad & \sum_{t=0}^T f_t(x_{t-k:t})^\top f_t(x_{t-k:t}) \\ \text{s.t.} \quad & \forall_t : g_t(x_{t-k:t}) \leq 0, \quad h_t(x_{t-k:t}) = 0. \end{aligned}$$

### Outline

- These are only some very brief notes on path optimization
- The aim is to explain how to *formulate* the optimization problem. Concerning the optimization algorithm itself, refer to the *Optimization* lecture.

8:1

- Constrained optimization methods:
  - Log-barrier, squared penalties
  - **Augmented Lagrangian**
- Note: also the Lagrangian is the form of the so-called **Gauss-Newton** form. The pseudo Hessian is a banded, symmetric, positive-definite matrix.

8:5

### From inverse kinematics to path costs

- Recall our optimality principle of inverse kinematics

$$\operatorname{argmin}_q \|q - q_0\|_W^2 + \|\Phi(q)\|^2$$

- A trajectory  $q_{0:T}$  is a sequence of robot configurations  $q_t \in \mathbb{R}^n$
- Consider the cost function

$$f(q_{0:T}) = \sum_{t=0}^T \|q_{t-1} - q_t\|_W^2 + \sum_{t=0}^T \|\Phi_t(q_t)\|^2$$

(where  $(q_{-1}$  is a given prefix)

- $\|q_{t-1} - q_t\|_W^2$  represents **control costs**
- $\Phi_t(q_t)$  represents **task costs**

8:2

### General $k$ -order cost terms

[Notation:  $x_t$  instead of  $q_t$  represents joint state]

$$\begin{aligned} \min_{x_{0:T}} \quad & \sum_{t=0}^T f_t(x_{t-k:t})^\top f_t(x_{t-k:t}) \\ \text{s.t.} \quad & \forall_t : g_t(x_{t-k:t}) \leq 0, \quad h_t(x_{t-k:t}) = 0. \end{aligned}$$

8:3

### Cost terms

- The  $f_t(x_{t-k:t})$  terms can penalize various things:

$k = 1$	$f_t(x_{t-1}, x_t) = x_t - x_{t-1}$	penalize velocity
$k = 2$	$f_t(x_{t-2}, \dots, x_t) = x_t - 2x_{t-1} + x_{t-2}$	penalize acceleration
$k = 3$	$f_t(x_{t-3}, \dots, x_t) = x_t - 3x_{t-1} + 3x_{t-2} - x_{t-3}$	penalize jerk

or in some arbitrary task spaces

$k = 0$	$f_t(x_t) = \phi(x_t) - y^*$	penalize offset in some task space
$k = 1$	$f_t(x_{t-1}, x_t) = \phi x_t - \phi x_{t-1}$	
$k = 2$	$f_t(x_{t-2}, \dots, x_t) = \phi x_t - 2\phi x_{t-1} + \phi x_{t-2}$	
$k = 3$	$f_t(x_{t-3}, \dots, x_t) = \phi x_t - 3\phi x_{t-1} + 3\phi x_{t-2} - \phi x_{t-3}$	

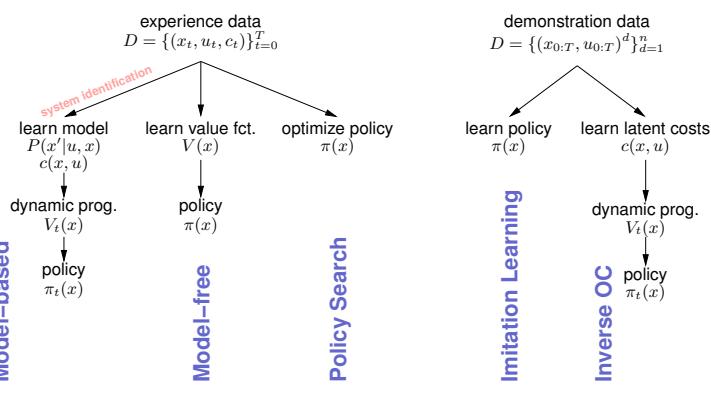
- And terms  $f_t$  can be stacked arbitrarily

8:4



## Five approaches to RL

Model-based



9:10

Or trace observed trajectories by minimizing perturbation costs  
(Atkeson & Schaal 1997)

## Imitation Learning

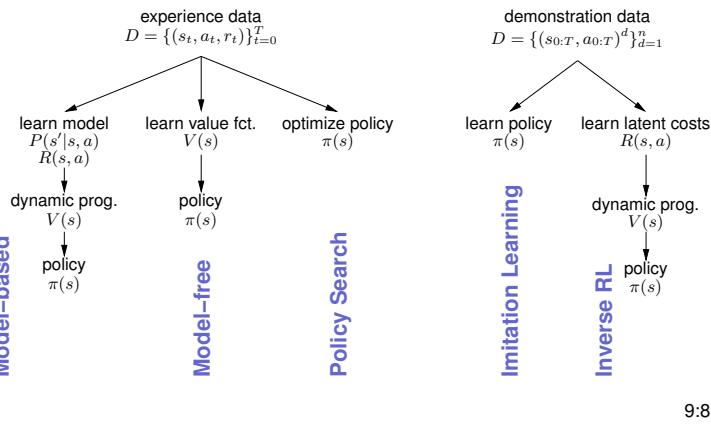


Atkeson &amp; Schaal

9:11

## Five approaches to RL

Model-based



9:8

## Imitation Learning

$$D = \{(s_0:T, a_0:T)\}_{d=1}^n \xrightarrow{\text{learn/copy}} \pi(s)$$

- Use ML to imitate demonstrated state trajectories  $x_{0:T}$

Literature:

Atkeson &amp; Schaal: Robot learning from demonstration (ICML 1997)

Schaal, Ijspeert &amp; Billard: Computational approaches to motor learning by imitation (Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences 2003)

Grimes, Chalodhorn &amp; Rao: Dynamic Imitation in a Humanoid Robot through Nonparametric Probabilistic Inference. (RSS 2006)

Rüdiger Dillmann: Teaching and learning of robot tasks via observation of human performance (Robotics and Autonomous Systems, 2004)

9:9

## Imitation Learning

- There are many ways to imitate/copy the observed policy:

Learn a density model  $P(a_t | s_t)P(s_t)$  (e.g., with mixture of Gaussians) from the observed data and use it as policy (Billard et al.)

Literature:

Pieter Abbeel &amp; Andrew Ng: Apprenticeship learning via inverse reinforcement learning (ICML 2004)

Andrew Ng &amp; Stuart Russell: Algorithms for Inverse Reinforcement Learning (ICML 2000)

Nikolay Jetchev &amp; Marc Toussaint: Task Space Retrieval Using Inverse Feedback Control (ICML 2011).

9:12

## Inverse RL (Apprenticeship Learning)

- Given: demonstrations  $D = \{x_{0:T}^d\}_{d=1}^n$
- Try to find a reward function that **discriminates demonstrations from other policies**
  - Assume the reward function is linear in some features  $R(x) = w^\top \phi(x)$
  - Iterate:
    - Given a set of candidate policies  $\{\pi_0, \pi_1, \dots\}$
    - Find weights  $w$  that maximize the value margin between teacher and all other candidates

$$\max_{w, \xi} \xi$$

$$\text{s.t. } \forall \pi_i : \underbrace{w^\top \langle \phi \rangle_D}_{\text{value of demonstrations}} \geq \underbrace{w^\top \langle \phi \rangle_{\pi_i}}_{\text{value of } \pi_i} + \xi \\ \|w\|^2 \leq 1$$

3. Compute a new candidate policy  $\pi_i$  that optimizes  $R(x) = w^\top \phi(x)$  and add to candidate list.

(Abbeel & Ng, ICML 2004)

9:13

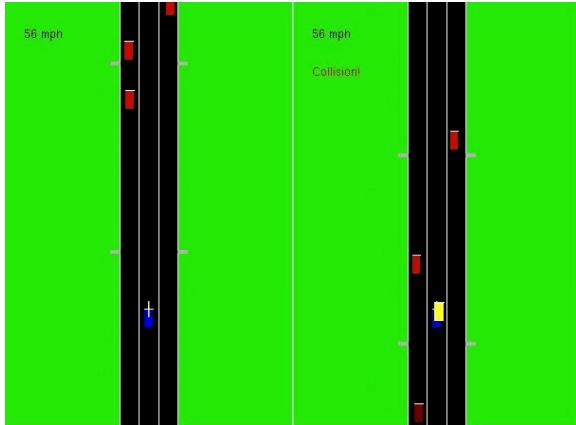
– Then the natural gradient ( $\frac{\partial V(\beta)}{\partial \beta}$  multiplied with inv. Fisher metric) updates

$$\beta^{\text{new}} = \beta + \alpha w$$

- Another is PoWER: Use the stochastic policy  $\pi(a|s)$ , let  $a_t = \phi(s_t)^\top (\beta + \epsilon_t)$  where  $\epsilon_t \sim \mathcal{N}(0, \Sigma)$  is the sampled noise, then  $\frac{\partial V(\beta)}{\partial \beta} = 0$  implies

$$\beta \leftarrow \beta + \frac{\mathbb{E}_{\xi|\beta} \{ \sum_{t=0}^H \epsilon_t Q^\pi(s_t, a_t, t) \}}{\mathbb{E}_{\xi|\beta} \{ \sum_{t=0}^H Q^\pi(s_t, a_t, t) \}}$$

9:17



9:14



Kober & Peters: *Policy Search for Motor Primitives in Robotics*, NIPS 2008.

(serious reward shaping!)

9:18

## Policy Search with Policy Gradients

9:15

### Policy gradients

- In continuous state/action case, represent the policy as linear in arbitrary state features:

$$\pi(s) = \sum_{j=1}^k \phi_j(s) \beta_j = \phi(s)^\top \beta \quad (\text{deterministic})$$

$$\pi(a|s) = \mathcal{N}(a | \phi(s)^\top \beta, \phi(s)^\top \Sigma \phi(s)) \quad (\text{stochastic})$$

with  $k$  features  $\phi_j$ .

- Given an episode  $\xi = (s_t, a_t, r_t)_{t=0}^H$ , we want to estimate

$$\frac{\partial V(\beta)}{\partial \beta}$$

9:16



Learning To Walk

Tedrake, Zhang & Seung: *Stochastic policy gradient reinforcement learning on a simple 3D biped*. IROS, 2849-2854, 2004. [http://groups.csail.mit.edu/robotics-center/public\\_papers/Tedrake04a.pdf](http://groups.csail.mit.edu/robotics-center/public_papers/Tedrake04a.pdf)

9:19

## Policy Gradients

- One approach is called REINFORCE:

$$\begin{aligned} \frac{\partial V(\beta)}{\partial \beta} &= \frac{\partial}{\partial \beta} \int P(\xi|\beta) R(\xi) d\xi = \int P(\xi|\beta) \frac{\partial}{\partial \beta} \log P(\xi|\beta) R(\xi) d\xi \\ &= \mathbb{E}_{\xi|\beta} \{ \frac{\partial}{\partial \beta} \log P(\xi|\beta) R(\xi) \} = \mathbb{E}_{\xi|\beta} \{ \sum_{t=0}^H \gamma^t \frac{\partial \log \pi(a_t|s_t)}{\partial \beta} \sum_{t'=t}^H \gamma^{t'-t} r_{t'} \} \\ &\quad \underbrace{Q^\pi(s_t, a_t, t)}_{Q^\pi(s_t, a_t, t)} \end{aligned}$$

- Another is Natural Policy Gradient

– Estimate the  $Q$ -function as linear in the basis functions  $\frac{\partial}{\partial \beta} \log \pi(a|s)$ :

$$Q(s, a) \approx \left[ \frac{\partial \log \pi(a|s)}{\partial \beta} \right]^\top w$$

## Policy Gradients – references

Peters & Schaal (2008): *Reinforcement learning of motor skills with policy gradients*, Neural Networks.

Kober & Peters: *Policy Search for Motor Primitives in Robotics*, NIPS 2008.

Vlassis, Toussaint (2009): *Learning Model-free Robot Control by a Monte Carlo EM Algorithm*. Autonomous Robots 27, 123-130.

Rawlik, Toussaint, Vijayakumar (2012): *On Stochastic Optimal Control and Reinforcement Learning by Approximate Inference*. RSS 2012. ( $\psi$ -learning)

- These methods are sometimes called **white-box optimization**:

They optimize the policy parameters  $\beta$  for the total reward  $R = \sum \gamma^t r_t$  while trying to exploit knowledge of how the process is actually parameterized

9:20

## Optimizing and Learning

### Policy Search with Black-Box Optimization [skipped]

9:21

### “Black-Box Optimization”

- The term is not really well defined
  - I use it to express that *only*  $f(x)$  can be evaluated
  - $\nabla f(x)$  or  $\nabla^2 f(x)$  are not (directly) accessible
- More common terms:
  - Global optimization
  - This usually emphasizes that methods should not get stuck in local optima
  - Very very interesting domain – close analogies to (active) Machine Learning, bandits, POMDPs, optimal decision making/planning, optimal experimental design
  - Usually mathematically well founded methods

- Stochastic search or Evolutionary Algorithms or Local Search
  - Usually these are local methods (extensions trying to be “more” global)
  - Various interesting heuristics
  - Some of them (implicitly or explicitly) locally approximating gradients or 2nd order models

9:22

Black-Box optimization is strongly related to learning:

- When we have local a gradient or Hessian, we can take that local information and run – no need to keep track of the history or learn (exception: BFGS)
- In the black-box case we have no local information directly accessible
  - one needs to account for the history in some way or another to have an idea where to continue search
- “Accounting for the history” very often means learning: Learning a local or global model of  $f$  itself, learning which steps have been successful recently (gradient estimation), or which step directions, or other heuristics

9:25

## Stochastic Search

9:26

### Stochastic Search

- The general recipe:
  - The algorithm maintains a probability distribution  $p_\theta(x)$
  - In each iteration it takes  $n$  samples  $\{x_i\}_{i=1}^n \sim p_\theta(x)$
  - Each  $x_i$  is evaluated → data  $\{(x_i, f(x_i))\}_{i=1}^n$
  - That data is used to update  $\theta$

- Stochastic Search:

---

**Input:** initial parameter  $\theta$ , function  $f(x)$ , distribution model  $p_\theta(x)$ , update heuristic  $h(\theta, D)$   
**Output:** final  $\theta$  and best point  $x$

- 1: **repeat**
- 2:     Sample  $\{x_i\}_{i=1}^n \sim p_\theta(x)$
- 3:     Evaluate samples,  $D = \{(x_i, f(x_i))\}_{i=1}^n$
- 4:     Update  $\theta \leftarrow h(\theta, D)$
- 5: **until**  $\theta$  converges

---

9:27

## Stochastic Search

- The parameter  $\theta$  is the only “knowledge/information” that is being propagated between iterations
- $\theta$  encodes what has been learned from the history
- $\theta$  defines where to search in the future

- Evolutionary Algorithms:  $\theta$  is a parent population
- Evolution Strategies:  $\theta$  defines a Gaussian with mean & variance
- Estimation of Distribution Algorithms:  $\theta$  are parameters of some distribution model, e.g. Bayesian Network
- Simulated Annealing:  $\theta$  is the “current point” and a temperature

9:28

## Black-Box Optimization

- Problem: Let  $x \in \mathbb{R}^n$ ,  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , find

$$\min_x f(x)$$

where we can only evaluate  $f(x)$  for any  $x \in \mathbb{R}^n$

- A constrained version: Let  $x \in \mathbb{R}^n$ ,  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $g : \mathbb{R}^n \rightarrow \{0, 1\}$ , find

$$\min_x f(x) \quad \text{s.t.} \quad g(x) = 1$$

where we can only evaluate  $f(x)$  and  $g(x)$  for any  $x \in \mathbb{R}^n$

I haven't seen much work on this. Would be interesting to consider this more rigorously.

9:23

## A zoo of approaches

- People with many different backgrounds drawn into this  
Ranging from heuristics and Evolutionary Algorithms to heavy mathematics
  - Evolutionary Algorithms, esp. Evolution Strategies, Covariance Matrix Adaptation, Estimation of Distribution Algorithms
  - Simulated Annealing, Hill Climbing, Downhill Simplex
  - local modelling (gradient/Hessian), global modelling

9:24

## Example: Gaussian search distribution $(\mu, \lambda)$ -ES

From 1960s/70s. Rechenberg/Schwefel

- Perhaps the simplest type of distribution model

$$\theta = (\hat{x}), \quad p_t(x) = \mathcal{N}(x|\hat{x}, \sigma^2)$$

a  $n$ -dimensional isotropic Gaussian with fixed deviation  $\sigma$

- Update heuristic:

- Given  $D = \{(x_i, f(x_i))\}_{i=1}^\lambda$ , select  $\mu$  best:  $D' = \text{bestOf}_\mu(D)$
- Compute the new mean  $\hat{x}$  from  $D'$

- This algorithm is called “Evolution Strategy  $(\mu, \lambda)$ -ES”

- The Gaussian is meant to represent a “species”
- $\lambda$  offspring are generated
- the best  $\mu$  selected

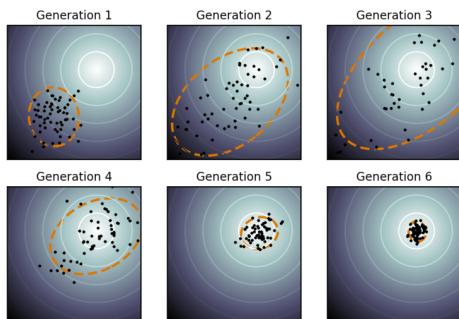
9:29

## Covariance Matrix Adaptation (CMA-ES)

- An obvious critique of the simple Evolution Strategies:

- The search distribution  $\mathcal{N}(x|\hat{x}, \sigma^2)$  is isotropic  
(no going *forward*, no preferred direction)
- The variance  $\sigma$  is fixed!

- Covariance Matrix Adaptation Evolution Strategy (CMA-ES)



9:30

## Covariance Matrix Adaptation (CMA-ES)

- In Covariance Matrix Adaptation

$$\theta = (\hat{x}, \sigma, C, p_\sigma, p_C), \quad p_\theta(x) = \mathcal{N}(x|\hat{x}, \sigma^2 C)$$

where  $C$  is the covariance matrix of the search distribution

- The  $\theta$  maintains two more pieces of information:  $p_\sigma$  and  $p_C$  capture the “path” (motion) of the mean  $\hat{x}$  in recent iterations
- Rough outline of the  $\theta$ -update:
  - Let  $D' = \text{bestOf}_\mu(D)$  be the set of selected points
  - Compute the new mean  $\hat{x}$  from  $D'$
  - Update  $p_\sigma$  and  $p_C$  proportional to  $\hat{x}_{k+1} - \hat{x}_k$

- Update  $\sigma$  depending on  $|p_\sigma|$
- Update  $C$  depending on  $p_C p_C^\top$  (rank-1-update) and  $\text{Var}(D')$

9:31

## CMA references

Hansen, N. (2006), "The CMA evolution strategy: a comparing review"

Hansen et al.: Evaluating the CMA Evolution Strategy on Multi-modal Test Functions, PPSN 2004.

Function	$f_{\text{stop}}$	init	$n$	CMA-ES	DE	RES	LOS
$f_{\text{Ackley}}(x)$	1e-3	$[-30, 30]^n$	20	<b>2667</b>	.	. 6.0e4	
			30	<b>3701</b>	12481	1.1e5 9.3e4	
			100	<b>11900</b>	36801	.	.
$f_{\text{Griewank}}(x)$	1e-3	$[-600, 600]^n$	20	<b>3111</b>	8691	.	.
			30	<b>4455</b>	11410 *	8.5e-3/2e5	.
			100	<b>12796</b>	31796	.	.
$f_{\text{Rastrigin}}(x)$	0.9	$[-5.12, 5.12]^n$	20	<b>68586</b>	<b>12971</b>	.	9.2e4
	DE:	$[-600, 600]^n$	30	<b>147416</b>	<b>20150</b> *	1.0e5 2.3e5	
			100	<b>1010989</b>	<b>73620</b>	.	.
$f_{\text{Rastrigin}}(Ax)$	0.9	$[-5.12, 5.12]^n$	30	<b>152000</b>	<b>171/1.25e6</b> *	.	.
			100	<b>1011556</b>	<b>944/1.25e6</b> *	.	.
$f_{\text{Schwefel}}(x)$	1e-3	$[-500, 500]^n$	5	<b>43810</b>	<b>2567</b> *	.	7.4e4
			10	<b>240899</b>	<b>5522</b> *	.	5.6e5

- For “large enough” populations local minima are avoided

- An interesting variant:

Igel et al.: A Computational Efficient Covariance Matrix Update and a  $(1 + 1)$ -CMA for Evolution Strategies, GECCO 2006.

9:32

## CMA conclusions

- It is a good starting point for an off-the-shelf black-box algorithm
- It includes components like estimating the local gradient ( $p_\sigma, p_C$ ), the local “Hessian” ( $\text{Var}(D')$ ), smoothing out local minima (large populations)

9:33

## Stochastic search conclusions

**Input:** initial parameter  $\theta$ , function  $f(x)$ , distribution model  $p_\theta(x)$ , update heuristic  $h(\theta, D)$

**Output:** final  $\theta$  and best point  $x$

- 1: **repeat**
- 2:     Sample  $\{x_i\}_{i=1}^n \sim p_\theta(x)$
- 3:     Evaluate samples,  $D = \{(x_i, f(x_i))\}_{i=1}^n$
- 4:     Update  $\theta \leftarrow h(\theta, D)$
- 5: **until**  $\theta$  converges

- The framework is very general
- The crucial difference between algorithms is their choice of  $p_\theta(x)$

9:34

## RL under Partial Observability

- Data:

$$D = \{(u_t, c_t, y_t)\}_{t=0}^T$$

→ state  $x_t$  not observable

- Model-based RL is daunting: Learning  $P(x'|u, x)$  and  $P(y|u, x)$  with latent  $x$  is very hard

- Model-free: The policy needs to map the **history** to a new control

$$\pi : (y_{t-h, \dots, t-1}, u_{t-h, \dots, t-1}) \mapsto u_t$$

or any **features of the history**

$$u_t = \phi(y_{t-h, \dots, t-1}, u_{t-h, \dots, t-1})^\top w$$

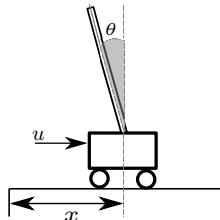
9:35

---

## 10 Control Theory

Path finding vs. trajectory optimization, local vs. global, Dijkstra, Probabilistic Roadmaps, Rapidly Exploring Random Trees, non-holonomic systems, car system equation, path-finding for non-holonomic systems, control-based sampling, Dubins curves

### Cart pole example



state  $x = (x, \dot{x}, \theta, \dot{\theta})$

$$\ddot{\theta} = \frac{g \sin(\theta) + \cos(\theta) [-c_1 u - c_2 \dot{\theta}^2 \sin(\theta)]}{\frac{4}{3}l - c_2 \cos^2(\theta)}$$

$$\ddot{x} = c_1 u + c_2 [\dot{\theta}^2 \sin(\theta) - \dot{\theta} \cos(\theta)]$$

10:1

### Control Theory

- Concerns controlled systems of the form

$$\dot{x} = f(x, u) + \text{noise}(x, u)$$

and a controller of the form

$$\pi : (x, t) \mapsto u$$

- We'll neglect stochasticity here
- When analyzing a given controller  $\pi$ , one analyzes **closed-loop system** as described by the differential equation

$$\dot{x} = f(x, \pi(x, t))$$

(E.g., analysis for convergence & stability)

10:2

### Core topics in Control Theory

- Stability\***

Analyze the stability of a closed-loop system  
→ Eigenvalue analysis or Lyapunov function method

- Controllability\***

Analyze which dimensions (DoFs) of the system can actually in principle be controlled

- Transfer function**

Analyze the closed-loop transfer function, i.e., "how frequencies are transmitted through the system". (→ Laplace transformation)

- Controller design**

Find a controller with desired stability and/or transfer function properties

- Optimal control\***

Define a cost function on the system behavior. Optimize a controller to minimize costs

10:3

### Control Theory references

- Robert F. Stengel: *Optimal control and estimation*

Online lectures: <http://www.princeton.edu/~stengel/MAE546Lectures.html> (esp. lectures 3,4 and 7-9)

- From robotics lectures:

Stefan Schaal's lecture Introduction to Robotics: <http://www-clm.usc.edu/Teaching/TeachingIntroductionToRoboticsSy>

Drew Bagnell's lecture on Adaptive Control and Reinforcement Learning <http://robotwhisperer.org/acrls11/>

Jonas Buchli's lecture on Optimal & Learning Control for Autonomous Robots <http://www.adrl.ethz.ch/doku.php/adrl:education:lecture:fs2015>

10:4

### Outline

- We'll first consider *optimal control*

Goal: understand Algebraic Riccati equation significance for local neighborhood control

- Then controllability & stability

10:5

### 10.1 Optimal control

10:6

#### Optimal control (discrete time)

Given a controlled dynamic system

$$x_{t+1} = f(x_t, u_t)$$

we define a cost function

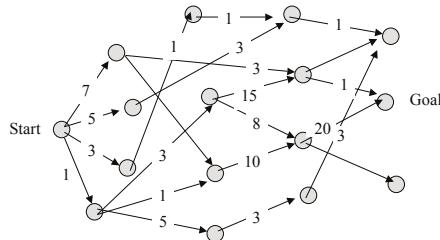
$$J^\pi = \sum_{t=0}^T c(x_t, u_t) + \phi(x_T)$$

where  $x_0$  and the controller  $\pi : (x, t) \mapsto u$  are given, which determines  $x_{1:T}$  and  $u_{0:T}$

10:7

## Dynamic Programming & Bellman principle

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.



$$V(\text{state}) = \min_{\text{edge}} [c(\text{edge}) + V(\text{next-state})]$$

10:8

## Bellman equation (discrete time)

- Define the **value function** or optimal **cost-to-go** function

$$V_t(x) = \min_{\pi} \left[ \sum_{s=t}^T c(x_s, u_s) + \phi(x_T) \right]_{x_t=x}$$

- Bellman equation

$$V_t(x) = \min_u [c(x, u) + V_{t+1}(f(x, u))]$$

The argmin gives the optimal control signal:  $\pi_t^*(x) = \arg\min_u [\dots]$

Derivation:

$$\begin{aligned} V_t(x) &= \min_{\pi} \left[ \sum_{s=t}^T c(x_s, u_s) + \phi(x_T) \right] \\ &= \min_{u_t} \left[ c(x, u_t) + \min_{\pi} \left[ \sum_{s=t+1}^T c(x_s, u_s) + \phi(x_T) \right] \right] \\ &= \min_{u_t} [c(x, u_t) + V_{t+1}(f(x, u_t))] \end{aligned}$$

10:9

## Optimal Control (continuous time)

Given a controlled dynamic system

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

we define a cost function with horizon  $T$

$$J^\pi = \int_0^T c(x(t), u(t)) dt + \phi(x(T))$$

where the start state  $x(0)$  and the controller  $\pi : (x, t) \mapsto u$  are given, which determine the closed-loop system trajectory  $x(t), u(t)$  via  $\dot{x} = f(x, \pi(x, t))$  and  $u(t) = \pi(x(t), t)$

10:10

## Hamilton-Jacobi-Bellman equation (continuous time)

- Define the **value function** or optimal **cost-to-go** function

$$V(x, t) = \min_{\pi} \left[ \int_t^T c(x(s), u(s)) ds + \phi(x(T)) \right]_{x(t)=x}$$

- Hamilton-Jacobi-Bellman equation

$$-\frac{\partial}{\partial t} V(x, t) = \min_u [c(x, u) + \frac{\partial V}{\partial x} f(x, u)]$$

The argmin gives the optimal control signal:  $\pi^*(x) = \arg\min_u [\dots]$

Derivation: Apply the discrete-time Bellman equation for  $V_t$  and  $V_{t+dt}$ :

$$\begin{aligned} V(x, t) &= \min_u \left[ \int_t^{t+dt} c(x, u) dt + V(x(t+dt), t+dt) \right] \\ &= \min_u \left[ \int_t^{t+dt} c(x, u) dt + V(x, t) + \int_t^{t+dt} \frac{dV(x, t)}{dt} dt \right] \\ &= \min_u \left[ \int_t^{t+dt} c(x, u) dt + \frac{\partial V}{\partial t} + \frac{\partial V}{\partial x} \dot{x} dt \right] \\ &= \min_u \left[ c(x, u) + \frac{\partial V}{\partial t} + \frac{\partial V}{\partial x} f(x, u) \right] \end{aligned}$$

10:11

## Infinite horizon case

$$J^\pi = \int_0^\infty c(x(t), u(t)) dt$$

- This cost function is stationary (time-invariant)!

- the optimal value function is stationary ( $V(x, t) = V(x)$ )
- the optimal control signal depends on  $x$  but not on  $t$
- the optimal controller  $\pi^*$  is stationary

- The HJB and Bellman equations remain “the same” but with the same (stationary) value function independent of  $t$ :

$$0 = \min_u [c(x, u) + \frac{\partial V}{\partial x} f(x, u)] \quad (\text{cont. time})$$

$$V(x) = \min_u [c(x, u) + V(f(x, u))] \quad (\text{discrete time})$$

The argmin gives the optimal control signal:  $\pi^*(x) = \arg\min_u [\dots]$

10:12

## Infinite horizon examples

- Cart-pole balancing:

- You always want the pole to be upright ( $\theta \approx 0$ )
  - You always want the car to be close to zero ( $x \approx 0$ )
  - You want to spare energy (apply low torques) ( $u \approx 0$ )
- You might define a cost

$$J^\pi = \int_0^\infty \|\theta\|^2 + \epsilon \|x\|^2 + \rho \|u\|^2$$

- Reference following:

- You always want to stay close to a reference trajectory  $r(t)$
- Define  $\tilde{x}(t) = x(t) - r(t)$  with dynamics  $\dot{\tilde{x}}(t) = f(\tilde{x}(t) + r(t), u) - \dot{r}(t)$
- Define a cost

$$J^\pi = \int_0^\infty \|\tilde{x}\|^2 + \rho \|u\|^2$$

- Many many problems in control can be framed this way

10:13

## Comments

- The Bellman equation is fundamental in optimal control theory, but also Reinforcement Learning
- The HJB eq. is a differential equation for  $V(x, t)$  which is in general hard to solve
- The (time-discretized) Bellman equation can be solved by Dynamic Programming starting backward:

$$V_T(x) = \phi(x), \quad V_{T-1}(x) = \min_u [c(x, u) + V_T(f(x, u))] \quad \text{etc.}$$

But it might still be hard or infeasible to represent the functions  $V_t(x)$  over continuous  $x$ !

- Both become significantly simpler under linear dynamics and quadratic costs:

→ Riccati equation

10:14

## Riccati differential equation = HJB eq. in LQ case

- In the Linear-Quadratic (LQ) case, the value function always is a quadratic function of  $x$ !

Let  $V(x, t) = x^T P(t) x$ , then the HJB equation becomes

$$\begin{aligned} -\frac{\partial}{\partial t} V(x, t) &= \min_u \left[ c(x, u) + \frac{\partial V}{\partial x} f(x, u) \right] \\ -x^T \dot{P}(t) x &= \min_u \left[ x^T Q x + u^T R u + 2x^T P(t)(Ax + Bu) \right] \\ 0 &= \frac{\partial}{\partial u} \left[ x^T Q x + u^T R u + 2x^T P(t)(Ax + Bu) \right] \\ &= 2u^T R + 2x^T P(t)B \\ u^* &= -R^{-1} B^T P x \end{aligned}$$

### ⇒ Riccati differential equation

$$-\dot{P} = A^T P + PA - PBR^{-1}B^T P + Q$$

10:17

## Riccati differential equation

$$-\dot{P} = A^T P + PA - PBR^{-1}B^T P + Q$$

- This is a differential equation for the matrix  $P(t)$  describing the quadratic value function. If we solve it with the finite horizon constraint  $P(T) = F$  we solved the optimal control problem
- The optimal control  $u^* = -R^{-1} B^T P x$  is called **Linear Quadratic Regulator**

Note: If the state is dynamic (e.g.,  $x = (q, \dot{q})$ ) this control is linear in the positions and linear in the velocities and is an instance of **PD control**

The matrix  $K = R^{-1} B^T P$  is therefore also called **gain matrix**. For instance, if  $x(t) = (q(t) - r(t), \dot{q}(t) - \dot{r}(t))$  for a reference  $r(t)$  and  $K = [K_p \quad K_d]$  then

$$u^* = K_p(r(t) - q(t)) + K_d(\dot{r}(t) - \dot{q}(t))$$

10:18

## Linear-Quadratic Optimal Control

### linear dynamics

$$\dot{x} = f(x, u) = Ax + Bu$$

### quadratic costs

$$c(x, u) = x^T Q x + u^T R u, \quad \phi(x_T) = x_T^T F x_T$$

- Note: Dynamics neglects constant term; costs neglect linear and constant terms. This is because
  - constant costs are irrelevant
  - linear cost terms can be made away by redefining  $x$  or  $u$
  - constant dynamic term only introduces a constant drift

10:15

## Linear-Quadratic Control as Local Approximation

- LQ control is important also to control non-LQ systems in the **neighborhood** of a desired state!

Let  $x^*$  be such a desired state (e.g., cart-pole:  $x^* = (0, 0, 0, 0)$ )
 

- linearize the dynamics around  $x^*$
- use 2nd order approximation of the costs around  $x^*$
- control the system *locally* as if it was LQ
- pray that the system will never leave this neighborhood!

10:16

## Riccati equations

- Finite horizon continuous time

### Riccati differential equation

$$-\dot{P} = A^T P + PA - PBR^{-1}B^T P + Q, \quad P(T) = F$$

- Infinite horizon continuous time

### Algebraic Riccati equation (ARE)

$$0 = A^T P + PA - PBR^{-1}B^T P + Q$$

- Finite horizon discrete time ( $J^\pi = \sum_{t=0}^T \|x_t\|_Q^2 + \|u_t\|_R^2 + \|x_T\|_F^2$ )

$$P_{t-1} = Q + A^\top [P_t - P_t B(R + B^\top P_t B)^{-1} B^\top P_t] A, \quad P_T = F$$

- Infinite horizon discrete time ( $J^\pi = \sum_{t=0}^\infty \|x_t\|_Q^2 + \|u_t\|_R^2$ )

$$P = Q + A^\top [P - PB(R + B^\top PB)^{-1} B^\top P] A$$

10:19

## Example: 1D point mass

- Dynamics:

$$\ddot{q}(t) = u(t)/m$$

$$\begin{aligned} x &= \begin{pmatrix} q \\ \dot{q} \end{pmatrix}, \quad \dot{x} = \begin{pmatrix} \dot{q} \\ \ddot{q} \end{pmatrix} = \begin{pmatrix} \dot{q} \\ u(t)/m \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 1/m \end{pmatrix} u \\ &= Ax + Bu, \quad A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 1/m \end{pmatrix} \end{aligned}$$

- Costs:

$$c(x, u) = \epsilon \|x\|^2 + \varrho \|u\|^2, \quad Q = \epsilon \mathbf{I}, \quad R = \varrho \mathbf{I}$$

- Algebraic Riccati equation:

$$\begin{aligned} P &= \begin{pmatrix} a & c \\ c & b \end{pmatrix}, \quad u^* = -R^{-1}B^\top Px = \frac{-1}{\varrho m}[cq + b\dot{q}] \\ 0 &= \bar{A}^\top P + PA - PBR^{-1}B^\top P + Q \\ &= \begin{pmatrix} c & b \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & a \\ 0 & c \end{pmatrix} - \frac{1}{\varrho m^2} \begin{pmatrix} c^2 & bc \\ bc & b^2 \end{pmatrix} + \epsilon \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned}$$

10:20

## Example: 1D point mass (cont.)

- Algebraic Riccati equation:

$$\begin{aligned} P &= \begin{pmatrix} a & c \\ c & b \end{pmatrix}, \quad u^* = -R^{-1}B^\top Px = \frac{-1}{\varrho m}[cq + b\dot{q}] \\ 0 &= \begin{pmatrix} c & b \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & a \\ 0 & c \end{pmatrix} - \frac{1}{\varrho m^2} \begin{pmatrix} c^2 & bc \\ bc & b^2 \end{pmatrix} + \epsilon \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned}$$

First solve for  $c$ , then for  $b = m\sqrt{\varrho}\sqrt{c+\epsilon}$ . Whether the damping ration  $\xi = \frac{b}{\sqrt{4mc}}$  depends on the choices of  $\varrho$  and  $\epsilon$ .

- The Algebraic Riccati equation is usually solved numerically.  
(E.g. `are`, `care` or `dare` in Octave)

10:21

## Optimal control comments

- HJB or Bellman equation are very powerful concepts
- Even if we can solve the HJB eq. and have the optimal control, we still don't know how the system really behaves?
  - Will it actually reach a “desired state”?
  - Will it be stable?
  - It is actually “controllable” at all?

- Last note on optimal control:

Formulate as a constrained optimization problem with objective function  $J^\pi$  and constraint  $\dot{x} = f(x, u)$ .  $\lambda(t)$  are the Langrange multipliers. It turns out that  $\frac{\partial}{\partial x} V(x, t) = \lambda(t)$ . (See Stengel)

10:22

## Relation to other topics

- Optimal Control:

$$\min_{\pi} J^\pi = \int_0^T c(x(t), u(t)) dt + \phi(x(T))$$

- Inverse Kinematics:

$$\min_q f(q) = \|q - q_0\|_W^2 + \|\phi(q) - y^*\|_C^2$$

- Operational space control:

$$\min_u f(u) = \|u\|_H^2 + \|\ddot{\phi}(q) - \ddot{y}^*\|_C^2$$

- Trajectory Optimization: (control hard constraints could be included)

$$\min_{q_{0:T}} f(q_{0:T}) = \sum_{t=0}^T \|\Psi_t(q_{t-k}, \dots, q_t)\|^2 + \sum_{t=0}^T \|\Phi_t(q_t)\|^2$$

- Reinforcement Learning:

- Markov Decision Processes  $\leftrightarrow$  discrete time stochastic controlled system  $P(x_{t+1} | u_t, x_t)$
- Bellman equation  $\rightarrow$  Basic RL methods (Q-learning, etc)

10:23

## 10.2 Controllability

10:24

### Controllability

- As a starting point, consider the claim:

*“Intelligence means to gain maximal controllability over all degrees of freedom over the environment.”*

Note:

- controllability (ability to control)  $\neq$  control
- What does controllability mean exactly?

- I think the general idea of *controllability* is really interesting
  - Linear control theory provides one specific definition of controllability, which we introduce next..

10:25

## Controllability

- Consider a linear controlled system

$$\dot{x} = Ax + Bu$$

How can we tell from the matrices  $A$  and  $B$  whether we can control  $x$  to eventually reach any desired state?

- Example:  $x$  is 2-dim,  $u$  is 1-dim:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u$$

Is  $x$  "controllable"?

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u$$

Is  $x$  "controllable"?

10:26

## Controllability

We consider a linear stationary (=time-invariant) controlled system

$$\dot{x} = Ax + Bu$$

- Complete controllability:** All elements of the state can be brought from arbitrary initial conditions to zero in finite time
- A system is completely controllable iff the **controllability matrix**

$$C := [B \ AB \ A^2B \ \dots \ A^{n-1}B]$$

has full rank  $\dim(x)$  (that is, all rows are linearly independent)

- Meaning of  $C$ :*

The  $i$ th row describes how the  $i$ th element  $x_i$  can be influenced by  $u$   
 "B":  $\dot{x}_i$  is directly influenced via  $B$   
 "AB":  $\ddot{x}_i$  is "indirectly" influenced via  $AB$  ( $u$  directly influences some  $\dot{x}_j$  via  $B$ ; the dynamics  $A$  then influence  $\ddot{x}_i$  depending on  $\dot{x}_j$ )  
 " $A^2B$ ":  $\dddot{x}_i$  is "double-indirectly" influenced  
 etc...

Note:  $\ddot{x} = A\dot{x} + B\dot{u} = AAx + ABu + B\ddot{u}$   
 $\ddot{x} = A^3x + A^2Bu + AB\ddot{u} + B\ddot{u}$

10:27

## Controllability

- When all rows of the controllability matrix are linearly independent  $\Rightarrow (u, \dot{u}, \ddot{u}, \dots)$  can influence all elements of  $x$  independently
- If a row is zero  $\rightarrow$  this element of  $x$  cannot be controlled at all
- If 2 rows are linearly dependent  $\rightarrow$  these two elements of  $x$  will remain coupled forever

10:28

## Controllability examples

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} u \quad C = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \text{ rows linearly dependent}$$

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u \quad C = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \text{ 2nd row zero}$$

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \quad C = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \text{ good!}$$

10:29

## Controllability

Why is it important/interesting to analyze controllability?

- The Algebraic Riccati Equation will always return an "optimal" controller – but controllability tells us whether such a controller even has a chance to control  $x$
- "Intelligence means to gain maximal controllability over all degrees of freedom over the environment."
  - real environments are non-linear
  - "to have the ability to gain controllability over the environment's DoFs"

10:30

## 10.3 Stability

10:31

### Stability

- One of the most central topics in control theory
- Instead of designing a controller by first designing a cost function and then applying Riccati, design a controller such that the desired state is provably a stable equilibrium point of the closed loop system

10:32

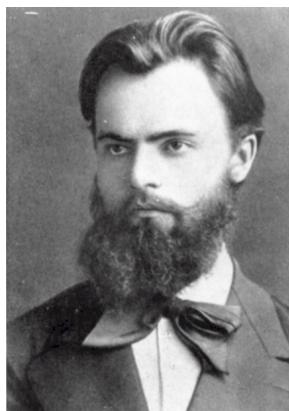
### Stability

- Stability is an analysis of the *closed loop* system. That is: for this analysis we don't need to distinguish between system and controller explicitly. Both together define the dynamics

$$\dot{x} = f(x, \pi(x, t)) =: f(x)$$

- The following will therefore discuss stability analysis of general differential equations  $\dot{x} = f(x)$
- What follows:
  - 3 basic definitions of stability
  - 2 basic methods for analysis by Lyapunov

10:33



Aleksandr Lyapunov (1857–1918)

10:34

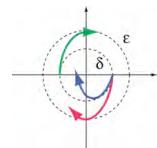
## Stability – 3 definitions

$\dot{x} = f(x)$  with equilibrium point  $x_0 = 0$

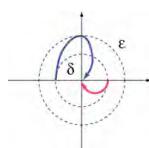
- $x_0$  is an **equilibrium point**  $\iff f(x_0) = 0$

- **Lyapunov stable or uniformly stable**  $\iff$

$$\forall \epsilon : \exists \delta \text{ s.t. } \|x(0)\| \leq \delta \Rightarrow \forall t : \|x(t)\| \leq \epsilon$$

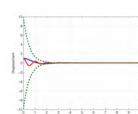


(when it starts off  $\delta$ -near to  $x_0$ , it will remain  $\epsilon$ -near forever)



- **asymptotically stable**  $\iff$

Lyapunov stable and  $\lim_{t \rightarrow \infty} x(t) = 0$



- **exponentially stable**  $\iff$

asymtotically stable and  $\exists \alpha, a \text{ s.t. } \|x(t)\| \leq ae^{-\alpha t} \|x(0)\|$

( $\rightarrow$  the “error” time integral  $\int_0^\infty \|x(t)\| dt \leq \frac{a}{\alpha} \|x(0)\|$  is bounded!)

10:35

## Linear Stability Analysis

(“Linear”  $\leftrightarrow$  “local” for a system linearized at the equilibrium point.)

- Given a linear system

$$\dot{x} = Ax$$

Let  $\lambda_i$  be the **eigenvalues** of  $A$

– The system is **asymptotically stable**  $\iff \forall i : \text{real}(\lambda_i) < 0$

– The system is **unstable**  $\iff \exists i : \text{real}(\lambda_i) > 0$

– The system is **marginally stable**  $\iff \forall i : \text{real}(\lambda_i) \leq 0$

- Meaning: An eigenvalue describes how the system behaves along one state dimension (along the eigenvector):

$$\dot{x}_i = \lambda_i x_i$$

As for the 1D point mass the solution is  $x_i(t) = ae^{\lambda_i t}$  and

– imaginary  $\lambda_i \rightarrow$  oscillation

– negative real( $\lambda_i$ )  $\rightarrow$  exponential decay  $\propto e^{-|\lambda_i| t}$

– positive real( $\lambda_i$ )  $\rightarrow$  exponential explosion  $\propto e^{|\lambda_i| t}$

10:36

## Linear Stability Analysis: Example

- Let’s take the 1D point mass  $\ddot{q} = u/m$  in *closed loop* with a PD controller  $u = -K_p q - K_d \dot{q}$

- Dynamics:

$$\begin{aligned}\dot{x} &= \begin{pmatrix} \dot{q} \\ \ddot{q} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x + 1/m \begin{pmatrix} 0 & 0 \\ -K_p & -K_d \end{pmatrix} x \\ A &= \begin{pmatrix} 0 & 1 \\ -K_p/m & -K_d/m \end{pmatrix}\end{aligned}$$

- Eigenvalues:

The equation  $\lambda \begin{pmatrix} q \\ \dot{q} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -K_p/m & -K_d/m \end{pmatrix} \begin{pmatrix} q \\ \dot{q} \end{pmatrix}$  leads to the equation  $\lambda \dot{q} = \lambda^2 q = -K_p/m q - K_d/m \lambda q$  or  $m\lambda^2 + K_d \lambda + K_p = 0$  with solution (compare slide 06:12)

$$\lambda = \frac{-K_d \pm \sqrt{K_d^2 - 4mK_p}}{2m}$$

For  $K_d^2 - 4mK_p$  negative, the  $\text{real}(\lambda) = -K_d/2m$

$\Rightarrow$  Positive derivative gain  $K_d$  makes the system stable.

10:37

## Side note: Stability for discrete time systems

- Given a discrete time linear system

$$x_{t+1} = Ax_t$$

Let  $\lambda_i$  be the **eigenvalues** of  $A$

– The system is **asymptotically stable**  $\iff \forall i : |\lambda_i| < 1$

– The system is **unstable**  $\iff \exists i : |\lambda_i| > 1$

– The system is **marginally stable**  $\iff \forall i : |\lambda_i| \leq 1$

10:38

## Linear Stability Analysis comments

- The same type of analysis can be done locally for non-linear systems, as we will do for the cart-pole in the exercises

- We can design a controller that minimizes the (negative) eigenvalues of  $A$ :

$\Leftrightarrow$  controller with fastest asymptotic convergence

This is a real alternative to optimal control!

10:39

## Lyapunov function method

- A method to analyze/prove stability for general non-linear systems is the famous “Lyapunov’s second method”

Let  $D$  be a region around the equilibrium point  $x_0$

- A Lyapunov function**  $V(x)$  for a system dynamics  $\dot{x} = f(x)$  is
  - positive,  $V(x) > 0$ , everywhere in  $D$  except...
  - at the equilibrium point where  $V(x_0) = 0$
  - always decreases,  $\dot{V}(x) = \frac{\partial V(x)}{\partial x} \dot{x} < 0$ , in  $D$  except...
  - at the equilibrium point where  $f(x) = 0$  and therefore  $\dot{V}(x) = 0$
- If there exists a  $D$  and a Lyapunov function  $\Rightarrow$  the system is *asymptotically stable*

If  $D$  is the whole state space, the system is *globally stable*

10:40

## Lyapunov function method

- The Lyapunov function method is very general.  $V(x)$  could be “anything” (energy, cost-to-go, whatever). Whenever one finds some  $V(x)$  that decreases, this proves stability
- The problem though is to think of some  $V(x)$  given a dynamics! (In that sense, the Lyapunov function method is rather a method of proof than a concrete method for stability analysis.)
- In standard cases, a good guess for the Lyapunov function is either the energy or the value function

10:41

## Lyapunov function method – Energy Example

- Let’s take the 1D point mass  $\ddot{q} = u/m$  in *closed loop* with a PD  $u = -K_p q - K_d \dot{q}$ , which has the solution (slide 06:15):

$$q(t) = b e^{-\xi/\lambda t} e^{i\omega_0 \sqrt{1-\xi^2} t}$$

- Energy of the 1D point mass:  $V(q, \dot{q}) := \frac{1}{2} m \dot{q}^2$

$$\dot{V}(x) = e^{-2\xi/\lambda t} V(x(0))$$

(using that the energy of an undamped oscillator is conserved)

- $V(x) < 0 \iff \xi > 0 \iff K_d > 0$

Same result as for the eigenvalue analysis

10:42

## Lyapunov function method – value function Example

- Consider infinite horizon linear-quadratic optimal control. The solution of the Algebraic Riccati equation gives the optimal controller.
- The value function satisfies

$$\begin{aligned} V(x) &= x^\top P x \\ \dot{V}(x) &= [Ax + Bu^*]^\top Px + x^\top P[Ax + Bu^*] \\ u^* &= -R^{-1}B^\top Px = Kx \\ \dot{V}(x) &= x^\top [(A + BK)^\top P + P(A + BK)]x \\ &= x^\top [A^\top P + PA + (BK)^\top P + P(BK)]x \\ 0 &= A^\top P + PA - PBR^{-1}B^\top P + Q \\ \dot{V}(x) &= x^\top [PBR^{-1}B^\top P - Q + (PBK)^\top + PBK]x \\ &= -x^\top [Q + K^\top RK]x \end{aligned}$$

(We could have derived this easier!  $x^\top Qx$  are the immediate state costs, and  $x^\top K^\top RKx = u^\top Ru$  are the immediate control costs—and  $\dot{V}(x) = -c(x, u^*)$ ! See slide 11 bottom.)

- That is:  $V$  is a Lyapunov function if  $Q + K^\top RK$  is positive definite!

10:43

## Observability & Adaptive Control

- When some state dimensions are not directly observable: analyzing higher order derivatives to *infer* them.  
Very closely related to controllability: Just like the controllability matrix tells whether state dimensions can (indirectly) be controlled; an observation matrix tells whether state dimensions can (indirectly) be inferred.
- Adaptive Control: When system dynamics  $\dot{x} = f(x, u, \beta)$  has unknown parameters  $\beta$ .
  - One approach is to estimate  $\beta$  from the data so far and use optimal control.
  - Another is to design a controller that has an additional internal update equation for an estimate  $\hat{\beta}$  and is provably stable. (See Schaal’s lecture, for instance.)

10:44

## 11 Exercises

### 11.1 Exercise 1

#### 11.1.1 Geometry

a) Read the notes on basic 3D geometry at <http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/3d-geometry.pdf> at least until section 2. Prepare questions for the exercises if you have any.

b) You have a book (coordinate frame  $B$ ) lying on the table (world frame  $W$ ). Initially  $B$  and  $W$  are identical. Now you move the book 1 unit to the right, then rotate it by  $45^\circ$  counter-clock-wise around its origin. Given a dot  $p$  marked on the book at position  $p^B = (1, 1)$  in the book coordinate frame, what are the coordinates  $p^W$  of that dot with respect to the world frame?

c) Given a point  $x$  with coordinates  $x^W = (0, 1)$  in world frame, what are its coordinates  $x^B$  in the book frame?

d) What is the *coordinate* transformation from world frame to book frame, and from book frame to world frame?

Please use homogeneous coordinates to derive these answers.

#### 11.1.2 Matrix equations

a) Let  $X, A$  be arbitrary matrices,  $A$  invertible. Solve for  $X$ :

$$XA + A^\top = \mathbf{I}$$

b) Let  $X, A, B$  be arbitrary matrices,  $(C - 2A^\top)$  invertible. Solve for  $X$ :

$$X^\top C = [2A(X + B)]^\top$$

c) Let  $x \in \mathbb{R}^n, y \in \mathbb{R}^d, A \in \mathbb{R}^{d \times n}$ .  $A$  obviously *not* invertible, but let  $A^\top A$  be invertible. Solve for  $x$ :

$$(Ax - y)^\top A = \mathbf{0}_n^\top$$

d) As above, additionally  $B \in \mathbb{R}^{n \times n}$ ,  $B$  positive-definite. Solve for  $x$ :

$$(Ax - y)^\top A + x^\top B = \mathbf{0}_n^\top$$

#### 11.1.3 Vector derivatives

Let  $x \in \mathbb{R}^n, y \in \mathbb{R}^d, f, g : \mathbb{R}^n \rightarrow \mathbb{R}^d, A \in \mathbb{R}^{d \times n}, C \in \mathbb{R}^{d \times d}$ . (Also provide the dimensionality of the results.)

a) What is  $\frac{\partial}{\partial x} x$  ?

b) What is  $\frac{\partial}{\partial x}[x^\top x]$  ?

c) What is  $\frac{\partial}{\partial x}[f(x)^\top f(x)]$  ?

d) What is  $\frac{\partial}{\partial x}[f(x)^\top Cg(x)]$  ?

e) Let  $B$  and  $C$  be symmetric (and pos.def.). What is the minimum of  $(Ax - y)^\top C(Ax - y) + x^\top Bx$  ?

#### 11.1.4 Optimization (optional)

Given  $x \in \mathbb{R}^n, f : \mathbb{R}^n \rightarrow \mathbb{R}$ , we want to find  $\operatorname{argmin}_x f(x)$ . (We assume  $f$  is uni-modal.)

a) What 1st-order optimization methods (querying  $f(x), \nabla f(x)$  in each iteration) do you know?

b) What 2nd-order optimization methods (querying  $f(x), \nabla f(x), \nabla^2 f(x)$  in each iteration) do you know?

c) What is backtracking line search?

## 11.2 Exercise 2

#### 11.2.1 Bayes Basics

a) Box 1 contains 8 apples and 4 oranges. Box 2 contains 10 apples and 2 oranges. Boxes are chosen with equal probability. What is the probability of choosing an apple? If an apple is chosen, what is the probability that it came from box 1?

b) The blue M&M was introduced in 1995. Before then, the color mix in a bag of plain M&Ms was: 30% Brown, 20% Yellow, 20% Red, 10% Green, 10% Orange, 10% Tan. Afterward it was: 24% Blue, 20% Green, 16% Orange, 14% Yellow, 13% Red, 13% Brown.

A friend of mine has two bags of M&Ms, and he tells me that one is from 1994 and one from 1996. He won't tell me which is which, but he gives me one M&M from each bag. One is yellow and one is green. What is the

probability that the yellow M&M came from the 1994 bag?

c) The Monty Hall Problem: I have three boxes. In one I put a prize, and two are empty. I then mix up the boxes. You want to pick the box with the prize in it. You choose one box. I then open *another* one of the two remaining boxes and show that it is empty. I then give you the chance to change your choice of boxes—should you do so?

d) Given a joint probability  $P(X, Y)$  over 2 binary random variables as the table

	$Y=0$	$Y=1$
$X=0$	.06	.24
$X=1$	.14	.56

What are  $P(X)$  and  $P(Y)$ ? Are  $X$  and  $Y$  conditionally independent?

## 11.2.2 Implementing Bayes Rule

a) Assume there is a robot that is trying to determine if a door is open. It has a perception module which is characterized by the following conditional probabilities

$$p(Z = \text{sense open} | X = \text{is open}) = .6$$

$$p(Z = \text{sense closed} | X = \text{is open}) = .4$$

$$p(Z = \text{sense open} | X = \text{is closed}) = .2$$

$$p(Z = \text{sense closed} | X = \text{is closed}) = .8$$

where  $X$  is the state of the door.

The robot has a prior belief of the state of door:  $bel(X = \text{open}) = 0.5$  and  $bel(X = \text{closed}) = .5$ . It perceives the observations “open”, “closed”, “open”, “open”, “open”. Use Bayes’ rule to update the belief. What is the belief/posterior after each update step?

b) Implement Bayes’ rule in a programming language of your choice and repeat a).

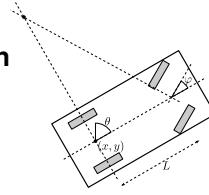
## 11.3 Exercise 3

### 11.3.1 Particle Filtering the location of a car

You are going to implement a particle filter. Start from the code in `share/teaching/RoboticsCourse/particle_`

The motion of the car is described by the following:

$$\begin{aligned} \text{State } q &= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} & \text{Con-} \\ \text{trols } u &= \begin{pmatrix} v \\ \varphi \end{pmatrix} & \text{equation} \\ \text{System} & \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v \cos \theta \\ v \sin \theta \\ (v/L) \tan \varphi \end{pmatrix} \\ \dot{x} &= v \cos \theta \\ \dot{y} &= v \sin \theta \\ \dot{\theta} &= (v/L) \tan \varphi \\ |\varphi| &< \Phi \end{aligned}$$



The `CarSimulator` simulates the described car (using Euler integration with step size 1sec). At each time step a control signal  $u = (v, \varphi)$  moves the car a bit and Gaussian noise with standard deviation  $\sigma_{\text{dynamics}} = .03$  is added to  $x$ ,  $y$  and  $\theta$ . Then, in each step, the car measures the relative positions of  $m$  landmark points, resulting in an observation  $y_t \in \mathbb{R}^{m \times 2}$ ; these observations are Gaussian-noisy with standard deviation  $\sigma_{\text{observation}} = .5$ . In the current implementation the control signal  $u_t = (.1, .2)$  is fixed (roughly driving circles).

a) Odometry (dead reckoning): First write a particle filter (with  $N = 100$  particles) that ignores the observations. For this you need to use the cars system dynamics (described above) to propagate each particle, and add some noise  $\sigma_{\text{dynamics}}$  to each particle (step 3 on slide 04:38). Draw the particles (their  $x, y$  component) into the display. Expected is that the particle cloud becomes larger and larger.

b) Next implement the likelihood weights  $w_i \propto P(y_t | x_t^i) = \mathcal{N}(y_t | y(x_t^i), \sigma) \propto e^{-\frac{1}{2}(y_t - y(x_t^i))^2 / \sigma^2}$  where  $y(x_t^i)$  is the (ideal) observation the car would have if it were in the particle position  $x_t^i$ . Since  $\sum_i w_i = 1$ , normalize the weights after this computation.

c) Test the full particle filter including the likelihood weights

(step 4) and resampling (step 2). Test using a larger ( $10\sigma_{\text{observation}}$ ) and smaller ( $\sigma_{\text{observation}}/10$ ) variance in the computation of the likelihood.

### 11.3.2 Histogram Filter

In the last exercise you implemented an binary Bayes' filter in a programming language of your choice. Now you are going to extend it to a histogram filter for localization as shown in the second lecture.

a) Represent the world as 10 bins. It is a circular world. Jumping off the edge on one side makes you reappear on the other one, e.g., going from position 0 one left makes you appear (most likely) on position 9). There are 3 landmarks: one at position 0, 1 and 5.

You can tell the robot to move one bin to the left and right and to stand still. The transition probability for the control is:

$u$	left	stand	right
left	.8,	.2	0
stand	0,	1	0
right	.1,	2	7

The observation likelihood is:

sense	no landmark	landmark
no landmark	.6	.4
landmark	.2	.8

i.e., if the robot is at a position with a landmark it perceives the landmark with a probability of 80%. Start with a uniform prior of the robot's localization.

Move the robot around and feed it observations. Be able to explain what happens.

b) (optional)

Be creative! Implement a more complex world with, e.g., 100 bins, a different motion model and Gaussian observation. You should be able to reuse the previous code.

## 11.4 Exercise 4

Download <http://ipvs.informatik.uni-stuttgart.de/mlr/15-Robotics/e04-code.tbz2> and unpack

it into

share/teaching/RoboticsCourse/. You should have the two folders `kalman_localization` and `kalman_sla`.

### 11.4.1 Kalman Localization

We consider the same car example as for the last exercise, but track the car using a Kalman filter.

a) To apply a Kalman filter (slide 04:50) we need Gaussian models for  $P(x_t | x_{t-1}, u_{t-1})$  as well as  $P(y_t | x_t)$ . We assume that the dynamics model is given as a local Gaussian of the form

$$P(x_{t+1} | x_t, u_t) = \mathcal{N}(x_{t+1} | f(x_t, u_t), \sigma_{\text{dynamics}})$$

We need the local linearization of  $f$  w.r.t.  $u$  which is represented by the matrix  $F$ . What is  $F$ ?

b) Implement a Kalman filter to track the car. Start with the code in `RoboticsCourse/kalman_localization/`. You are given a stub of a Kalman filter: class `ExtendedKalman`. Implement the prediction step: the `predict` function, the motion model `f` and the update of the Jacobian `updateF`. Make sure you initialize all matrices properly (use `sim.dynamicsNoise` for the noise matrix `R`).

Run the Kalman filter (effectively doing dead reckoning).

c) Implement the correction step: the `correct` function, the observation model `g` (use `sim.getMeanObservation` for this), and the update of the Jacobian `updateG` (use `getObservationJacobianAtState`). Again, make sure you initialize all matrices properly (use `sim.observationNoise` for the noise matrix `Q`).

Run the Kalman filter with `predict` and `correct`.

#### Tips:

–  $\sim X$  transposes the matrix  $X$

– `inverse(X)` calculates the matrix inverse of matrix  $X$

### 11.4.2 Kalman SLAM

Slide 04:63 outlines how to use a high-dimensional Kalman filter to simultaneously estimate the robot position (localization) and the landmarks position (mapping), i.e., the one Kalman filter tracks the robots pose and the landmarks at the same time.

- a) Write a pseudo-code for Kalman SLAM. (There is much freedom in how to organize the code, choices of notation and variables, etc. Try to write it as concise as possible. What is your state? Think about the Jacobians? How do you calculate the likelihood now that you don't have a map?)
- b) (optional) Try to implement Kalman SLAM, which simultaneously tracks the car and the landmarks. See `RoboticsCourse/kalman_slam/`. You can reuse most of the previous Kalman filter implementation. You have to change `f`, `g`, `updateF`, `updateG`, and initialize some matrices differently.

#### Tips:

The routines `sim.getMeanObservationAtStateAndLandmarks` and `sim.getObservationJacobianAtStateAndLandmarks` to retrieve the mean observation and the necessary Jacobians given the current mean estimate  $\theta$  of the landmarks must be implemented by you.

## 11.5 Exercise 5

### 11.5.1 Motion profiles

Construct a motion profile that accelerates constantly in the first quarter of the trajectory, then moves with constant velocity, then decelerates constantly in the last quarter. Write down the equation  $MP(s) : [0, 1] \mapsto [0, 1]$ .

### 11.5.2 Verify some things from the lecture

- a) On slide 05:21 we derived the basic inverse kinematics law. Verify that (assuming linearity of  $\phi$ , i.e.,

$J\delta q = \delta y$ ) for  $C \rightarrow \infty$  the desired task is fulfilled exactly, i.e.,  $\phi(q^*) = y^*$ . By writing  $C \rightarrow \infty$  we mean that  $C$  is a matrix of the form  $C = \varrho C_0$ ,  $\varrho \in \mathbb{R}$ , and we take the limit  $\varrho \rightarrow \infty$ .

- b) On slide 05:28 there is a term  $(\mathbf{I} - J^\# J)$  called nullspace projection. Verify that for  $\varrho \rightarrow \infty$  (and  $C = \varrho \mathbf{I}$ ) and any choice of  $a \in \mathbb{R}^n$

$$\delta q = (\mathbf{I} - J^\# J)a \Rightarrow \delta y = 0$$

(assuming linearity of  $\phi$ , i.e.,  $J\delta q = \delta y$ ). Note: this means that any choice of  $a$ , the motion  $(\mathbf{I} - J^\# J)a$  will not change the “endeffector position”  $y$ .

### 11.5.3 IK in the simulator

Download the code here: <https://ipvs.informatik.uni-stuttgart.de/mlr/15-Robotics/e05-code.tbz2>

Unpack the code such that the folder structure is 'robotics15/share'.

There are two functions `simpleArrayOperations` and `openningSimulator` that show some basic operations of our code. In the main function you can set the 'mode' variable that selects which function to execute. You can also set the mode on the command line with '`./x.exe -mode 1`'

The goal of this task is to reach the coordinates  $y^* = (-0.2, -0.4, 1.1)$  with the right hand of the robot. Assume  $W = \mathbf{I}$  and  $\sigma = .01$ .

- a) The provided function `reach()` already generates a motion in one step using inverse kinematics  $\delta q = J^\# \delta y$  with  $J^\# = (J^\top J + \sigma^2 W)^{-1} J^\top$ . Record the task error, that is, the deviation of hand position from  $y^*$  after each step. Why is it initially large?
- b) Try to do 100 smaller steps  $\delta q = \alpha J^\# \delta y$  with  $\alpha = .1$  (each step starting with the outcome of the previous step). How does the task error evolve over time?
- c) Generate a nice trajectory composed of  $T = 100$  time steps. Interpolate the target linearly  $\hat{y} \leftarrow y_0 + (t/T)(y^* - y_0)$  in each time step.
- d) Generate a trajectory that moves the right hand in a circle centered at  $(-0.2, -0.4, 1.1)$ , aligned with the  $xz$ -plane, with radius 0.2.

## 11.6 Exercise 6

### 11.6.1 Task spaces and Jacobians (3P)

In the lecture we introduced the basic kinematic maps  $\phi_{i,v}^{\text{pos}}(q)$  and  $\phi_{i,v}^{\text{vec}}(q)$ , and their Jacobians,  $J_{i,v}^{\text{pos}}(q)$  and  $J_{i,v}^{\text{vec}}(q)$ , where  $i$  may refer to any part of the robot and  $v$  is any point or direction on this part. In the following you may assume that we have routines to compute these for any  $q$ . The problem is to express other kinematic maps and their Jacobians in terms of these knowns. In the following you're asked to define more involved kinematics maps (a.k.a. task maps)  $\phi(q)$  to solve certain motion problems. Please formulate all these maps such that the overall optimization problem is

$$f(q) = \|q - q_0\|_W^2 + \|\phi(q)\|^2$$

that is, the motion aims to minimize  $\phi(q)$  to zero (absorb the  $y^*$  in the definition of  $\phi$ .)

- a) Assume you would like to control the pointing direction of the robot's head (e.g., its eyes) to point to a desired external world point  $x^W$ . What task map can you define to achieve this? What is the Jacobian?
- b) You would like the two hands or the robot to become parallel (e.g. for clapping). What task map can you define to achieve this? What is the Jacobian?
- c) You would like to control a standard endeffector position  $p_{\text{eff}}$  to be at  $y^*$ , as usual. Can you define a 1-dimensional task map  $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$  to achieve this? What is its Jacobian?

### 11.6.2 Verify some things from the lecture (1P)

On slide 05:40 it says that

$$\begin{aligned} \underset{q}{\operatorname{argmin}} \|q - q_0\|_W^2 + \|\Phi(q)\|^2 \\ \approx q_0 - (J^\top J + W)^{-1} J^\top \Phi(q_0) = q_0 - J^\# \Phi(q_0) \end{aligned}$$

where the approximation  $\approx$  means that we use the local linearization  $\Phi(q) = \Phi(q_0) + J(q - q_0)$ . Verify this.

### 11.6.3 Multiple task variables (2P)

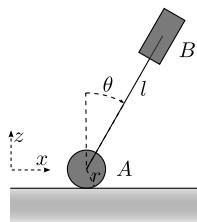
Consider again the last week's exercise where the robot moved his hand in a circle. A solution to this exercise is provided as starting point here: <https://ipvs.informatik.uni-stuttgart.de/mlr/15-Robotics/e06-code.tbz2>

[informatik.uni-stuttgart.de/mlr/15-Robotics/e06-code.tbz2](https://ipvs.informatik.uni-stuttgart.de/mlr/15-Robotics/e06-code.tbz2)

- a) We've seen that the solution does track the circle nicely, but the trajectory "gets lost" in joint space, leading to very strange postures. We can fix this by adding more tasks, esp. a task that permanently tries to (moderately) minimize the distance of the configuration  $q$  to a natural posture  $q_{\text{home}}$ . Realize this by adding a respective task.
- b) Make the robot simultaneously point upward with the left hand. Use `kinematicsVec` (and `jacobianVec`) to compute the orientation of a robot shape for this.

## 11.7 Exercise 7

### 11.7.1 Fun with Euler-Lagrange (3P)



Consider an inverted pendulum mounted on a wheel in the 2D x-z-plane; similar to a Segway. The exercise is to derive the Euler-Lagrange equation for this system.

Tips:

- Strictly follow the scheme we discussed on slide 06:23.
- Use the generalized coordinates

$$q = (x, \theta) \quad (4)$$

with  $x$  is the position of the wheel and  $\theta$  the angle of the pendulum relative to the world-vertical.

- The system can be parameterized by
  - $m_A, I_A, m_B, I_B$ : masses and inertias of bodies  $A$  (=wheel) and  $B$  (=pendulum)
  - $r$ : radius of the wheel
  - $l$ : length of the pendulum (height of its COM)
- Describe the **pose** of every body (depending on  $q$ ) by the  $(x, z, \phi)$  coordinate: its position in the x-z-plane, and its rotation  $\phi$  relative to the world-vertical. Accordingly represent the (linear and angular) velocities as a 3-vector.
- In this 3-dim space, the mass matrix of every body is

$$M_i = \begin{pmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & I_i \end{pmatrix} \quad (5)$$

### 11.7.2 PD control to hold an arm steady (3P)

Download the code framework here:

<https://ipvs.informatik.uni-stuttgart.de/mlr/15-Robotics/e07-code.tbz2>

In the main.cpp we provide a dynamics simulation of a robot arm that simulates the system for 1000 timesteps.

The task is to write a controller that holds the robot arm steady, i.e.,  $q^* = 0$  and  $\dot{q}^* = 0$ .

- Apply direct PD control (*without* using  $M$  and  $F$ ) to each joint separately and try to find parameters  $K_p$  and  $K_d$  (potentially different for each joint) to hold the arm steady.
- Try to do the same with a PID controller that also includes the integral error

$$u = K_p(q^* - q) + K_d(\dot{q}^* - \dot{q}) + K_i \int_{s=0}^t (q^* - q(s)) ds .$$

- Now use the knowledge of  $M$  and  $F$  (slide 06:29) in each time step (both matrices are already provided in the the for loop of the code). Use the PD equation to determine desired accelerations  $\ddot{q}^*$  and use inverse dynamics to determine the necessary  $u$  (slide 06:30 - 06:31).

#### Bonus:

Try different starting positions.

Try the same controllers for the arm in pegArm2.ors.

Play with `setDynamicsSimulationNoise` and check stability.

## 11.8 Exercise 8

### 11.8.1 RRTs for path finding

In the code in `teaching/RoboticsCourse/rrt` (Download the code framework here:

<https://ipvs.informatik.uni-stuttgart.de/mlr/15-Robotics/e08-code.tbz2>

- The code demonstrates an RRT exploration and displays the explored endeffector positions. What is the endeffector's exploration distribution in the limit  $n \rightarrow \infty$ ? Specify such a distribution analytically for a planar 2 link arm.
- First grow an RRT *backward* from the target configuration  $q^* = (0.945499, 0.431195, -1.97155, 0.623969, 2.22355, -0.665206, -1.48356)$ . Stop when there exists a node close (<stepSize) to the  $q = 0$  configuration. Read out the collision free path from the tree and display it. Why would it be more difficult to grow the tree *forward* from  $q = 0$  to  $q^*$ ?

c) Find a collision free path using bi-directional RRTs (that is, 2 RRTs growing together). Use  $q^*$  to root the backward tree and  $q = 0$  to root the forward tree. Stop when a newly added node is close to the other tree. Read out the collision free path from the tree and display it.

d) (Bonus) Propose a simple method to make the found path smoother (while keeping it collision free). You're free to try anything.

### 11.8.2 A distance measure in phase space

Consider the 1D point mass with mass  $m = 1$  state  $x = (q, \dot{q})$ . The 2D space of  $(q, \dot{q})$  combining position and velocity is also called phase space. In RRT's (in line 4 on slide 07:40) we need to find the nearest configuration  $q_{\text{near}}$  to a  $q_{\text{target}}$ . But what does "near" mean in phase space? This exercise explores this question.

Consider a current state  $x_0 = (0, 1)$  (at position 0 with velocity 1). Pick *any* random phase state  $x_{\text{target}} \in \mathbb{R}^2$ . How would you connect  $x_0$  with  $x_{\text{target}}$  in a way that fulfills the differential constraints of the point mass dynamics? Given this trajectory connecting  $x_0$  with  $x_{\text{target}}$ , how would you quantify/measure the distance? (If you defined the connecting trajectory appropriately, you should be able to give an analytic expression for this distance.) Given a set (tree) of states  $x_{1:n}$  and you pick the closest to  $x_{\text{target}}$ , how would you "grow" the tree from this closest point towards  $x_{\text{target}}$ ?

## 11.9 Exercise 09

### 11.9.1 The value function in Markov Decision Processes

On slide 5 of the RL lecture we defined MDPs as

$$P(s_{0:T+1}, a_{0:T}, r_{0:T}; \pi) = P(s_0) \prod_{t=0}^T P(a_t|s_t; \pi) P(r_t|s_t, a_t) P(s_{t+1}|s_t, a_t), \quad (6)$$

where  $P(a_0|s_0; \pi)$  described the agent's policy. We assume a deterministic agent and write  $a_t = \pi(s_t)$ . The value of a state  $s$  is defined as the expected discounted sum of future rewards,

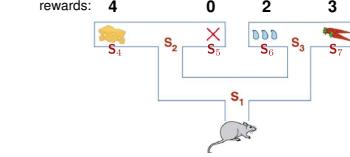
$$V^\pi(s) = \mathbb{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \dots | s_0 = s\} \quad (7)$$

given that the agent starts in state  $s$  and from there executes the policy  $\pi$ .

a) Prove

$$V^\pi(s) = \mathbb{E}\{r_0 | s, \pi(s)\} + \gamma \sum_{s'} P(s' | s, \pi(s)) V^\pi(s'). \quad (8)$$

b) Consider the following T-maze:



We distinguish 7 states  $s_1, \dots, s_7$  in the maze. The first 3 states are the T-junctions; the last 4 states receive rewards (4, 0, 2, 3). At each T-junction we have two possible actions: left, right. Everything is deterministic. Assume a discounting  $\gamma = 0.5$ .

Compute (by hand) the value function  $V^\pi$  over all states when  $\pi$  is the *random policy* (50/50 left/right at each junction).

c) Bellman's principle of optimality says that the optimal policy  $\pi^*$  has a value function  $V^{*\pi}(s) = V^*(s)$ ,

$$V^*(s) = \max_a \left[ \mathbb{E}\{r_0 | s, a\} + \gamma \sum_{s'} P(s' | s, a) V^\pi(s') \right]. \quad (9)$$

Compute (by hand) the optimal value function  $V^*$  over all states for the example above.

d) Now consider continuous state  $s$  and action  $a$ . Let the policy be stochastic and linear in features  $\phi(s) \in \mathbb{R}^k$ , that is,

$$\pi(a|s; \beta) = \mathcal{N}(a|\phi(s)^\top \beta, \phi(s)^\top \Sigma \phi(s)). \quad (10)$$

The covariance matrix  $\phi(s)^\top \Sigma \phi(s)$  describes that each action  $a_t = \phi(s_t)^\top (\beta + \epsilon_t)$  was generated by adding a noise term  $\epsilon_t \sim \mathcal{N}(0, \Sigma)$  to the parameter  $\beta$ . We always start in the same state  $\hat{s}$  and the value  $V^\pi(s_0)$  is

$$V^\pi(\hat{s}) = \mathbb{E}\{\sum_{t=0}^H H r_t | s_0 = \hat{s}\} \quad (11)$$

(no discounting, but only a finite horizon  $H$ .)

Optimality now requires that  $\frac{\partial V^\pi(\beta)}{\partial \beta} = 0$ . Assume that  $a \in \mathcal{R}$  is just 1-dimensional and  $\Sigma \in \mathbb{R}$  just a number. Try to prove (see slide 18) that we can derive

$$\beta^* = \beta^{\text{old}} + \left[ \mathbb{E}_{\xi|\beta} \left\{ \sum_{t=0}^H W(s_t) Q^{\pi(\beta)}(s_t, a_t, t) \right\} \right]^{-1} \mathbb{E}_{\xi|\beta} \left\{ \sum_{t=0}^H W(s_t) e_t Q^{\pi(\beta)}(s_t, a_t, t) \right\}$$

from  $\frac{\partial V^{\pi(\beta^*)}}{\partial \beta} = 0$ . (In the scalar case,  $W(s) = 1$ .) As a first step, derive

$$\frac{\partial}{\partial \beta} \log \pi(a|s)$$

Then insert  $a_t = \phi(s_t)^\top (\beta^{\text{old}} + \epsilon_t)$  and solve

$$\mathbb{E}_{\xi|\beta} \left\{ \sum_{t=0}^H \frac{\partial}{\partial \beta} \log \pi(a_t|s_t) Q(s_t, a_t, t) \right\} = 0$$

for  $\beta$ . This shows how you can get the optimal policy parameters  $\beta^*$  based on samples generated with  $\beta$ .

## 11.10 Exercise 11

### 11.10.1 Local linearization and Algebraic Riccati equation

Slide 10:02 describes the cart pole, which is similar to the Segway-type system of Exercise 7, but a little simpler. We'll solve the cart pole in this exercise.

The state of the cart-pole is given by  $x = (p, \dot{p}, \theta, \dot{\theta})$ , with  $p \in \mathbb{R}$  the position of the cart,  $\theta \in \mathbb{R}$  the pendulum's angular deviation from the upright position and  $\dot{p}, \dot{\theta}$  their respective temporal derivatives. The only control signal  $u \in \mathbb{R}$  is the force applied on the cart. The analytic model of the cart pole is

$$\ddot{\theta} = \frac{g \sin(\theta) + \cos(\theta) \left[ -c_1 u - c_2 \dot{\theta}^2 \sin(\theta) \right]}{\frac{4}{3}l - c_2 \cos^2(\theta)} \quad (12)$$

$$\ddot{p} = c_1 u + c_2 \left[ \dot{\theta}^2 \sin(\theta) - \ddot{\theta} \cos(\theta) \right] \quad (13)$$

with  $g = 9.8 \text{ ms}^{-2}$  the gravitational constant,  $l = 1 \text{ m}$  the pendulum length and constants  $c_1 = (M_p + M_c)^{-1}$  and  $c_2 = l M_p (M_p + M_c)^{-1}$  where  $M_p = M_c = 1 \text{ kg}$  are the pendulum and cart masses respectively.

a) Derive the local linearization of these dynamics around  $x^* = (0, 0, 0, 0)$ . The eventual dynamics should be in the form

$$\dot{x} = Ax + Bu$$

Note that

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{\partial \ddot{p}}{\partial p} & \frac{\partial \ddot{p}}{\partial \dot{p}} & \frac{\partial \ddot{p}}{\partial \theta} & \frac{\partial \ddot{p}}{\partial \dot{\theta}} \\ 0 & 0 & 0 & 1 \\ \frac{\partial \ddot{\theta}}{\partial p} & \frac{\partial \ddot{\theta}}{\partial \dot{p}} & \frac{\partial \ddot{\theta}}{\partial \theta} & \frac{\partial \ddot{\theta}}{\partial \dot{\theta}} \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ \frac{\partial \ddot{p}}{\partial u} \\ 0 \\ \frac{\partial \ddot{\theta}}{\partial u} \end{pmatrix}$$

where all partial derivatives are taken at the point  $p = \dot{p} = \theta = \dot{\theta} = 0$ .

The solution (to continue with the other parts) is

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{c_2 g}{\frac{4}{3}l - c_2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g}{\frac{4}{3}l - c_2} & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ c_1 + \frac{c_1 c_2}{\frac{4}{3}l - c_2} \\ 0 \\ \frac{-c_1}{\frac{4}{3}l - c_2} \end{pmatrix}$$

b) We assume a stationary infinite-horizon cost function of the form

$$\begin{aligned} J^\pi &= \int_0^\infty c(x(t), u(t)) dt \\ c(x, u) &= x^\top Q x + u^\top R u \\ Q &= \text{diag}(c, 0, 1, 0), \quad R = \mathbf{I}. \end{aligned}$$

That is, we penalize position offset  $c\|p\|^2$  and pole angle offset  $\|\theta\|^2$ . Choose  $c = \varrho = 1$  to start with.

Solve the Algebraic Riccati equation

$$0 = A^\top P + P^\top A - P B R^{-1} B^\top P + Q$$

by initializing  $P = Q$  and iterating using the following iteration:

$$P_{k+1} = P_k + \epsilon [A^\top P_k + P_k^\top A - P_k B R^{-1} B^\top P_k + Q]$$

Choose  $\epsilon = 1/1000$  and iterate until convergence. Output the gains  $K = -R^{-1} B^\top P$ . (Why should this iteration converge to the solution of the ARE?)

c) Solve the same Algebraic Riccati equation by calling the `are` routine of the octave control package (or a similar method in Matlab). For Octave, install the Ubuntu packages `octave3.2`, `octave-control`, and `qtoctave`, perhaps use `pkg load control` and `help are` in octave to ensure everything is installed, use `P=are(A, B*inv(R))` to solve the ARE. Output  $K = -R^{-1} B^\top P$  and compare to b).

(I found the solution  $K = (1.0000, 2.6088, 52.9484, 16.5952)$ .)

d) Implement the optimal Linear Quadratic Regulator  $u^* = Kx$  on the cart pole simulator. Download the code framework here: <https://ipvs.informatik.uni-stuttgart.de/mlr/15-Robotics/e10-code.tbz2>

Unpack the code such that the folder structure is 'robotics15/share'.

Simulate the optimal LQR and test it for various noise levels (by changing the variable `dynamicsNoise`).

## 11.11 Exercise 11

### 11.11.1 Controllability

Consider the local linearization of the cart-pole,

$$\dot{x} = Ax + Bu, \quad A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{c_2 g}{\frac{4}{3}l - c_2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g}{\frac{4}{3}l - c_2} & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ c_1 + \frac{c_1 c_2}{\frac{4}{3}l - c_2} \\ 0 \\ \frac{-c_1}{\frac{4}{3}l - c_2} \end{pmatrix}$$

Use the parameters  $c_1, c_2, l, g$  from the last exercise sheet.

Is the system controllable?

### 11.11.2 Stable control for the cart-pole (optional)

Consider a linear controller  $u = w^\top x$  with 4 parameters  $w \in \mathbb{R}^4$  for the cart-pole.

- a) What is the closed-loop linear dynamics  $\dot{x} = \hat{A}x$  of the system?
- b) Test if the controller with  $w = (1.0000, 2.6088, 52.9484, 16.5952)$  (computed using ARE) is asymptotically stable. What are the eigenvalues?
- c) Come up with a method that finds parameters  $w$  such that the closed-loop system is “maximally stable” around  $x^* = (0, 0, 0, 0)$  (e.g., asymptotically stable with fastest convergence rate).
- d) Output the optimal parameters and test them on the cart-pole simulation you developed in exercise 10.

## 12 Bullet points to help learning

*This is a summary list of core topics in the lecture and intended as a guide for preparation for the exam. Test yourself also on the bullet points in the table of contents. Going through all exercises is equally important.*

### 12.1 Geometry

- 3D geometry
  - Definition of an object pose, frame, transformations
  - Homogeneous transformations ( $4 \times 4$  matrix)
  - Composition of transformations, notation  $x^W = T_{W \rightarrow A} T_{A \rightarrow B} T_{B \rightarrow C} x^C$

### 12.2 Probability

- Properties of probability distributions
- Product Rule/Sum Rule/Bayes' Theorem
- Gaussian Distribution
  - Definition
  - Identities

### 12.3 Mobile Robotics

- Probability Basics
  - Definitions of random variable, probability distribution, joint, marginal, conditional distribution, independence
  - Bayes' Theorem
  - Continuous distributions, Gaussian, particles
- State Estimation
  - Formalization of the state estimation problem
  - The Bayes Filter as the general analytic solution
  - Gaussians and particles to approximate the Bayes filter and make it computationally feasible:
    - Details of a Particle Filter
    - Kalman filter (esp. assumptions made, not eq. or derivation)
    - Extended KF (assumptions made)

– Odometry (dead reckoning) as “Bayes filter without observations”

- SLAM
  - Definition of the SLAM problem
  - In what sense SLAM is a “chicken or egg problem”
  - Joint filtering over  $x$  and  $m$ : Extended Kalman SLAM
  - Particle-based SLAM (map belief for each particle)
  - Roughly: graph-based SLAM & loop closing

### 12.4 Kinematics

- Fwd kinematics & Jacobian
  - Fwd kinematics as composition of transformations
  - Transformations of a rotational joint
  - Kinematic maps  $\phi^{\text{pos}} : q \mapsto y$  and  $\phi^{\text{vec}}$
  - Definition of a Jacobian
  - Derivation of the position and vector Jacobians  $J^{\text{pos}}, J^{\text{vec}}$
- Inverse kinematics (IK)
  - Optimality criterion for IK
  - Using the local linearization to find the optimum
  - Pseudo code of following a task trajectory in small steps using Inverse Kinematics
  - Definition & example for a singularity
- Motion profiles & Interpolation
  - Motion profiles (esp. sine profile)
  - Joint space vs. task space interpolation of a trajectory [e.g. using a motion profile in one or the other space]
- Multiple Tasks
  - How to incorporate multiple tasks
  - What are interesting task maps; know at least about pos, vec, align, and limits
- Further
  - Be able to explain the consequences of the local linearization in IK (big steps → errors)

## 12.5 Dynamics

- 1D point mass & PID control
  - General form of a dynamic system
  - Dynamics of a 1D point mass
  - Position, derivative and integral feedback to control it to a desired state
  - Solution to the *closed-loop* PD system equations
  - Qualitative behaviors: oscillatory-damped, over-damped, critically damped
- Euler-Lagrange equation
  - Definition
  - Roughly: structure of the Euler-Lagrange equation for systems; understand at least  $T = \frac{1}{2}\dot{q}^T M \dot{q}$
  - Be able to apply on minimalistic system (e.g. pendulum, point mass)
- Robot dynamics & joint/operational space control
  - General form of the dynamics equation
  - Joint space control: given desired  $\ddot{q}^*$ , choose  $u^* = M(q) \ddot{q}^* + F(q, \dot{q})$
  - Operational space control: given desired  $\ddot{y}^*$ , choose  $u^* = T^\sharp(\ddot{y}^* - J\dot{q} + TF)$
  - Following a reference trajectory  $q_{0:T}^{\text{ref}}$  in joint space by imposing PD behavior around it

## 12.6 Path planning

- Basics
  - Path finding vs. trajectory optimization vs. feedback control
  - Roughly: BUG algorithms
  - Potential functions, and that they're nothing but IK with special task variables
  - Dijkstra Algorithm
- Probabilistic Road Maps (PRMs)
  - Definition & Generation
  - Importance of local planner
  - Roughly: knowing about probabilistic completeness
  - Roughly: alternative sampling strategies
- Rapidly Exploring Random Trees (RRTs)
  - Algorithm
  - Goal-directed & bi-directional extensions

- Non-holonomic Systems

- Definition of non-holonomicity. Be able to give example
- Path finding: control-based sampling
- RRT extension for control-based exploration
- Roughly: Intricacies with metrics for non-holonomic systems

## 12.7 Reinforcement Learning

- Markov decision process
  - Bellman's principle of optimality,
  - optimal value functions
  - optimal policy
  - Trajectory distribution
  - Policy gradient algorithms

## 12.8 Control Theory

- Optimal Control
  - Definition of the (continuous time) optimal control problem
  - Concept & definition of the value function
  - HJB equation
  - Infinite horizon → stationary solution
  - Awareness that optimal control is not the only approach; it shifts the problem of designing a controller to designing a cost function.
- Linear-Quadratic Optimal Control
  - Definition of problem (esp. assumptions made)
  - Be able to express system dynamics in (locally linearized) standard form  $\dot{x} = Ax + Bu$
  - Fact that the value function is quadratic  $V(x, t) = x^T P(t) x$
  - Riccati differential equation
  - How  $P$  gives the optimal Linear-Quadratic Regulator
  - Algebraic (infinite horizon) Riccati equation
- Controllability
  - Definition and understanding/interpretation of the controllability matrix  $C$
  - Definition of controllability
  - Be able to apply on simple examples

- Stability
  - Definitions of stability
  - Eigenvalue analysis for linear systems
  - Optimize controllers for negativity of eigenvalues
  - Definition of a Lyapunov function
  - Lyanunov's theorem:  $\exists$  Lyapunov fct.  $\rightarrow$  stability
  - Energy and value function as candidates for a Lyapunov fct.