

Principles of Robot Autonomy I

Course overview, mobile robot kinematics



Stanford
University



Team

Instructor



Prof. Jeannette Bohg

Course Assistants



Zhengguan(Gary) Dai



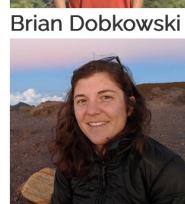
Brian Dobkowski



Mason Murray-Cooper



Hao Li



Stephanie Newdick



Alvin Sun

Collaborators

Daniel Watzenig

Labs



Autonomous Systems Lab



Center for Automotive

Research at Stanford



From automation...



9/26/22

AA 274A | Lecture 1

3

...to autonomy

Waymo Self-Driving Car



Intuitive DaVinci Surgical Robot



Apollo Robot at MPI for Intelligent Systems



Boston Dynamics – Spot Mini



Astrobee - NASA



Zipline

9/26/22

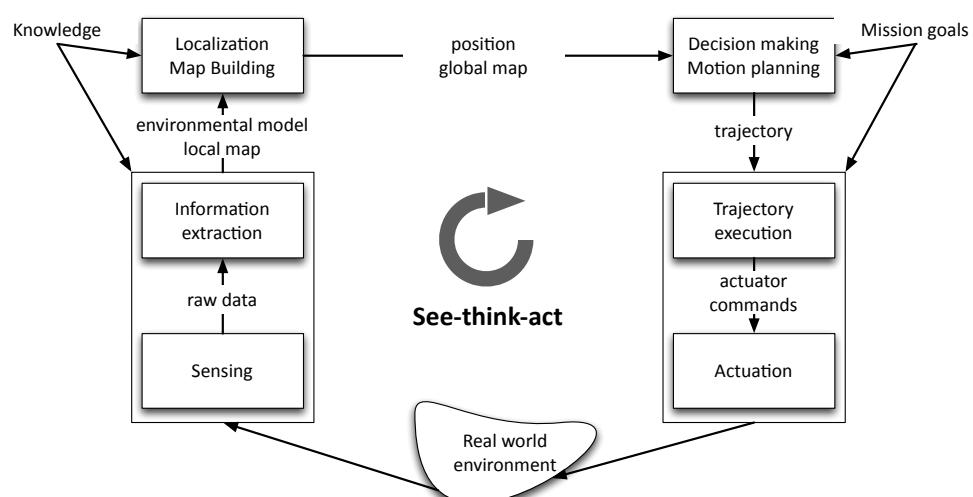
AA 274A | Lecture 1

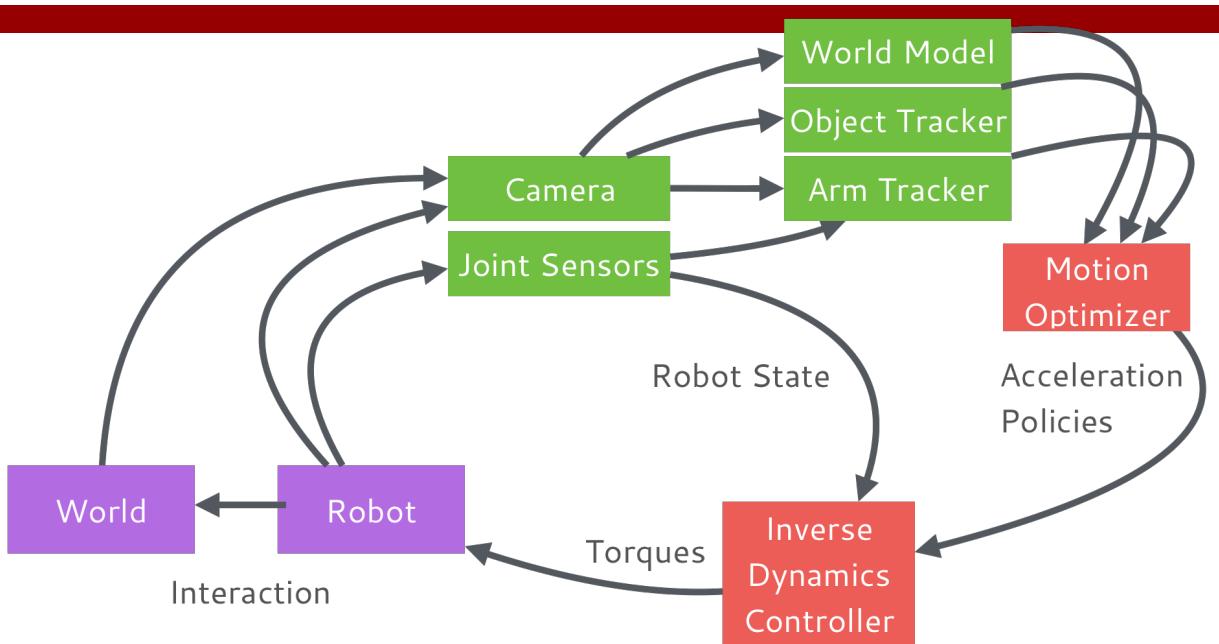
4

Course goals

- To learn the *theoretical, algorithmic, and implementation* aspects of main techniques for robot autonomy. Specifically, the student will
 1. Gain a fundamental knowledge of the “autonomy stack”
 2. Be able to apply such knowledge in applications / research by using ROS
 3. Devise novel methods and algorithms for robot autonomy

The see-think-act cycle





Kappler et al. *Real-Time Perception meets Reactive Motion Generation*. RA-L + ICRA'18. Finalist 2018 Amazon Best Systems Paper
9/26/22 AA 274B | Lecture 5

Course structure

- Four modules, roughly of equal length
 1. motion control and planning
 2. robotic perception
 3. localization and SLAM
 4. state machines and system architecture
- Extensive use of the Robot Operating System (ROS)
- Requirements
 - CS 106A or equivalent
 - CME 100 or equivalent (for calculus, linear algebra)
 - CME 106 or equivalent (for probability theory)
 - See also the [pre-knowledge quiz](#) on the course website

Schedule

Week	Topic
1	Course overview, mobile robot kinematics Introduction to the Robot Operating System (ROS) <i>Thursday: HW1 out</i>
2	Trajectory optimization Trajectory tracking & closed loop control
3	Motion planning I: graph search methods Motion planning II: sampling-based methods <i>Tuesday: HW1 due, HW2 out</i>
4	Robotic sensors & introduction to computer vision Camera models & camera calibration
5	Image processing, feature detection & description Information extraction & classic visual recognition <i>Tuesday: HW2 due, HW3 out</i>
6	Intro to localization & filtering theory Parameteric filtering (KF, EKF, UKF)
7	Intro to localization & filtering theory Parameteric filtering (KF, EKF, UKF)
8	Object detection / tracking, EKF localization Simultaneous localization and mapping (SLAM)
N/A	<i>Thanksgiving Break</i>
9	Multi-sensor perception & sensor fusion I (by Daniel Wattenig) Multi-sensor perception & sensor fusion II (by Daniel Wattenig) <i>Tuesday: HW4 due</i>
10	Stereo vision State machines <i>Tuesday: Final project check-in due</i>
11	Final Project Presentation and Demo <i>12/15 3:30 - 6:30 PM</i>

In-Person attendance is not required!

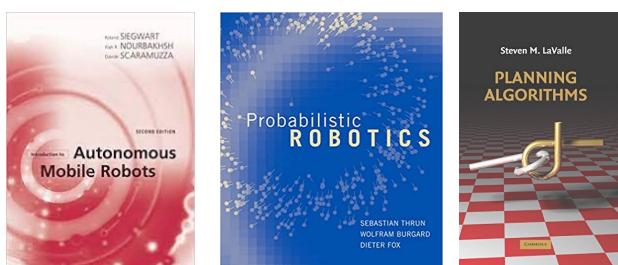
9/26/22

AA 274A | Lecture 1

9

Logistics - Lectures

- Tuesdays and Thursdays, 10:30am – 11:50 (Gates B1)
- Recordings will be made available to all students on Canvas.
- Course Materials in addition to Course Notes:



1 Mobile Robot Kinematics

Mobile Robot Kinematics

Motion, planning, and control are fundamental components of robotic autonomy¹. For example, in order for an autonomous car to accomplish an objective (e.g. move from point A to B) it first needs to plan a trajectory and determine what control inputs (e.g. throttle and steering) will enable it to follow the trajectory. Both of these components require an understanding of the physical behavior of the robot in order to develop reasonable/actuable plans and controls. In the context of motion planning and control, a robot's physical behavior

¹ R. Siegwart, J. R. Nourbakhsh, and D. Scaramuzza. *Introduction to Autonomous Mobile Robots*. MIT Press, 2011

9/26/22

AA 274A | Lecture 1

10

Logistics – Homework Assignments

- 4 assignments
- First Assignment out on Thursday
- ~2 weeks to submit on Gradescope
- Assignments are due Tuesdays which is when a new assignment will be released
- Budget of 6 late days, max 3 days per assignment
- Cooperation and discussion is encouraged, but solutions must be prepared individually. Add names of classmates who you collaborated with. Copying from other students or other sources is considered a case of academic dishonesty.
- Need to be typeset in Latex!

9/26/22

AA 274A | Lecture 1

11

Logistics – Sections

- 2-hour, once-a-week sessions starting Week 2
- Hands-on exercises that complement the lecture material, build familiarity with ROS, develop skills necessary for the final project

Monday: 5:30 – 7:30pm (virtual) **alvinsun**
Tuesday: 10:00am – 12:00pm (in-person) **li2053**
Tuesday: 4:30 – 6:30pm (in-person) **garydai**
Wednesday: 10:00am – 12:00pm (in-person) **masonmc**
Wednesday: 12:30 – 2:30pm (in-person) **snewdick**
Wednesday: 6:00 – 8:00pm (in-person) **bdkwoks**

Thursday: 9:30 – 11:30am (virtual) **li2053**
Thursday: 12:00 – 2:00pm (in-person) **alvinsun**
Thursday: 4:30 – 6:30pm (virtual) **garydai**
Friday: 9:30 – 11:30am (in-person) **snewdick**
Friday: 12:00 – 2:00pm (in-person) **bdkwoks**
Friday: 2:30 – 4:30pm (in-person) **masonmc**

- Section sign-up sheet coming soon!

9/26/22

AA 274A | Lecture 1

12



Logistics - Grades

- (20%) final project.
- (60%) homework.
- (20%) sections.
- (extra 5%) participation on Ed Discussion

4 or 3 units?

- AA174A: 4 units
- AA 274A/CS 237A/EE 260A: 3 or 4. Taking this class for 4 units entails completing an additional homework problem per problem set and also writing a one-page review of a paper at the end of the quarter.

Logistics

- Office hours:
 - Prof. Jeannette Bohg: Friday, 1-2pm (Gates 244 and Zoom)
 - CAs: Mondays 1 – 3pm (in-person), `garydai`, ...
Tuesdays 2pm – 4pm (virtual) `masonmc`, `snewdick`,
Thursdays 6pm – 8pm (virtual) `alvinsun`, `bdbkows`.
Friday 10am – 12pm (virtual) `alvinsun`, `li2053`.
- Course websites:
 - For course content: <http://asl.stanford.edu/aa274a/>
 - For course-related questions: <https://edstem.org/us/courses/28635>
 - For homework submissions: <https://www.gradescope.com/courses/439779>
 - For announcements and lecture videos: <https://canvas.stanford.edu/courses/159179>
 - To contact the AA274 staff, use the email: cs237a-aut2223-staff@lists.stanford.edu
- Syllabus has all the info!

Mobile robot kinematics

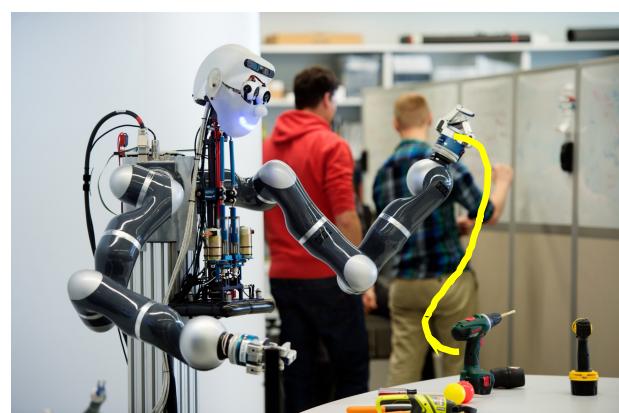
- Aim

- Understand motion constraints
- Learn about basic motion models for wheeled vehicles
- Gain insights for motion control

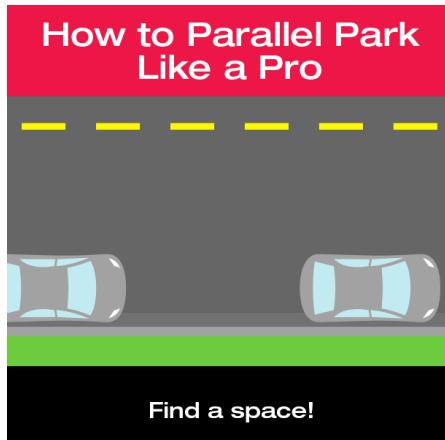
- Readings

- R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. Introduction to Autonomous Mobile Robots. MIT Press, 2nd Edition, 2011. Sections 3.1-3.3.
- B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo. Robotics: Modelling, Planning, and Control. Springer, 2008 (chapter 11).

Motion Planning and Control



Constraints in Motion Planning and Control



<https://tenor.com/view/parallel-park-parking-proper-gif-13789379>



Futurama - Put Your Head on My Shoulders [S02E10]

<https://tenor.com/view/parallel-park-parking-proper-gif-13789379>

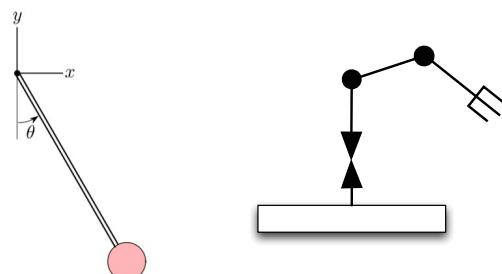
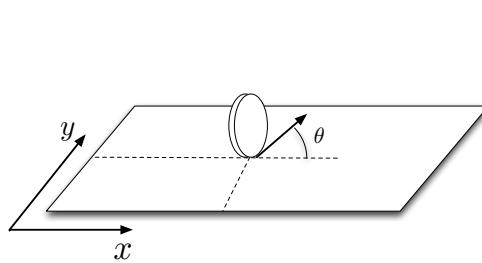
9/26/22

AA 274A | Lecture 1

19

Generalized Coordinates

- Let $\xi = [\xi_1, \dots, \xi_n]^T$ denote the configuration of a robot (e.g., $\xi = [x, y, \theta]^T$ for a wheeled mobile robot)



9/26/22

AA 274A | Lecture 1

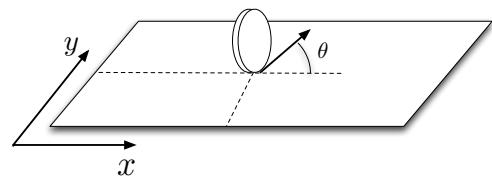
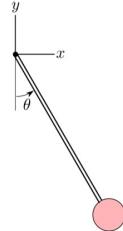
20

Kinematic constraints

$$a_i(\xi, \dot{\xi}) = 0, \quad i = 1, \dots, k < n$$

- constrain the instantaneous admissible motion of the mechanical system
- generally expressed in Pfaffian form, i.e., linear in the generalized velocities

$$a_i^T(\xi) \dot{\xi} = 0, \quad i = 1, \dots, k < n$$



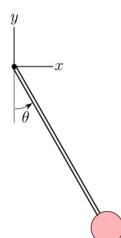
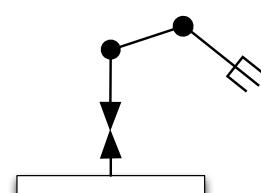
9/26/22

AA 274A | Lecture 1

21

Holonomic constraints

- $h_i(\xi) = 0$, for $i = 1, \dots, k < n$
- Reduce space of accessible configurations to an $n - k$ dimensional subset
- If all constraints are holonomic, the mechanical system is called holonomic
- Generally, the result of mechanical interconnections



9/26/22

AA 274A | Lecture 1

22

Examples of Holonomic constraints



Xiang, Qin, Mo et al., "SAPIEN: A Simulated Part-based Interactive Environment", CVPR 2020

9/26/22

AA 274A | Lecture 1

23

Kinematic constraints

$$a_i(\xi, \dot{\xi}) = 0, \quad i = 1, \dots, k < n$$

- constrain the instantaneous admissible motion of the mechanical system
- generally expressed in Pfaffian form, i.e., linear in the generalized velocities

$$a_i^T(\xi) \dot{\xi} = 0, \quad i = 1, \dots, k < n$$

- k holonomic constraints imply the existence of an equal number of kinematic constraints

$$\frac{d h_i(\xi)}{dt} = \frac{\partial h_i(\xi)}{\partial \xi} \dot{\xi} = 0, \quad i = 1, \dots, k < n$$

- However, the converse is not true in general...

9/26/22

AA 274A | Lecture 1

24

Nonholonomic constraints

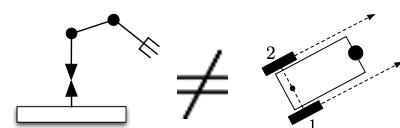
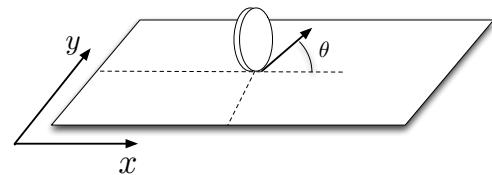
- If a kinematic constraint is not integrable in the form $h_i(\xi) = 0$, then it is said *nonholonomic* -> nonholonomic mechanical system
- Nonholonomic constraints reduce mobility in a completely different way. Consider a single Pfaffian constraint

$$a^T(\xi) \dot{\xi} = 0$$

- Holonomic
 - Can be integrated to $h(\xi) = 0$
 - Loss of accessibility, motion constrained to a level surface of dimension $n - 1$
- Nonholonomic
 - Velocities constrained to belong to a subspace of dimension $n - 1$, the null space of $a^T(\xi)$
 - No loss of accessibility

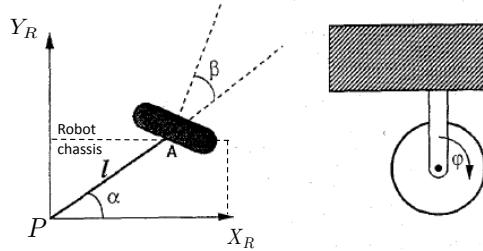
Example of nonholonomic system

- System: disk that rolls without slipping
- $\xi = [x, y, \theta]^T$
- No side slip constraint
$$[\dot{x}, \dot{y}] \cdot \begin{bmatrix} \sin \theta \\ -\cos \theta \end{bmatrix} = \dot{x} \sin \theta - \dot{y} \cos \theta = [\sin \theta, -\cos \theta, 0] \dot{\xi} = 0$$
- Facts:
 - No loss of accessibility
 - Wheeled vehicles are generally nonholonomic

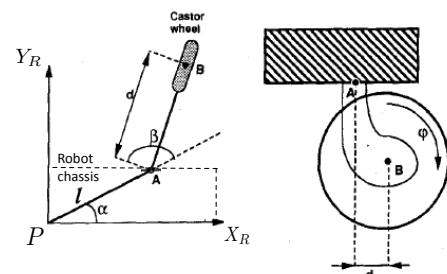


Types of wheels

- Standard wheels (four types)



Standard wheel -- fixed or steerable



Standard, off-centered wheel (caster)
-- passive or active

- Special wheels: achieve omnidirectional motion (e.g., Swedish or spherical wheels)

Kinematic models

- Assume the motion of a system is subject to k Pfaffian constraints

$$\begin{bmatrix} a_1^T(\xi) \\ \vdots \\ a_k^T(\xi) \end{bmatrix} \dot{\xi} := A^T(\xi) \dot{\xi} = 0$$

- Then, the admissible velocities at each configuration ξ belong to the $(n - k)$ -dimensional null space of matrix $A^T(\xi)$
- Denoting by $\{g_1(\xi), \dots, g_{n-k}(\xi)\}$ a basis of the null space of $A^T(\xi)$, admissible trajectories can be characterized as solutions to

$$\dot{\xi} = \sum_{j=1}^{n-k} g_j(\xi) u_j = G(\xi) u$$

Input vector

Example: unicycle

- Consider pure rolling constraint for the wheel:

$$\dot{x} \sin \theta - \dot{y} \cos \theta = [\sin \theta, -\cos \theta, 0] \dot{\xi} = a^T(\xi) \dot{\xi} = 0$$

- Consider the matrix

$$G(\xi) = [g_1(\xi), g_2(\xi)] = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix}$$

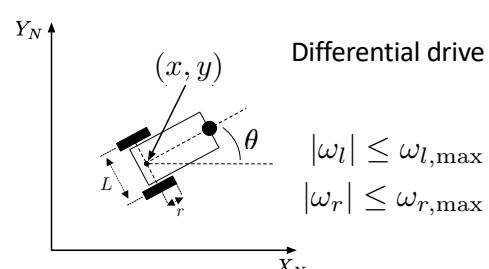
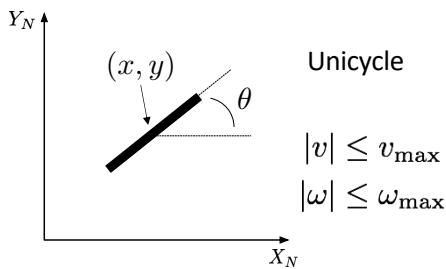
where $[g_1(\xi), g_2(\xi)]$ is a basis of the null space of $a^T(\xi)$

- All admissible velocities are therefore obtained as linear combination of $g_1(\xi)$ and $g_2(\xi)$

Unicycle and differential drive models

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos \theta \\ \sin \theta \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \omega$$

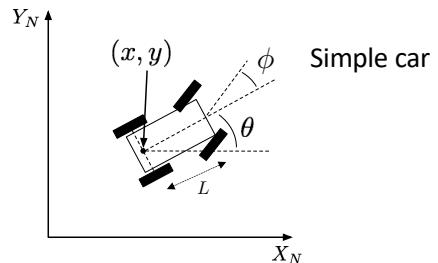
$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \frac{r}{2}(\omega_l + \omega_r) \cos \theta \\ \frac{r}{2}(\omega_l + \omega_r) \sin \theta \\ \frac{r}{L}(\omega_r - \omega_l) \end{pmatrix}$$



The kinematic model of the unicycle also applies to the differential drive vehicle, via the one-to-one input mappings: $v = \frac{r}{2}(\omega_r + \omega_l)$ $\omega = \frac{r}{L}(\omega_r - \omega_l)$

Simplified car model

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v \cos \theta \\ v \sin \theta \\ \frac{v}{L} \tan \phi \end{pmatrix}$$



$$|v| \leq v_{\max}, |\phi| \leq \phi_{\max} < \frac{\pi}{2}$$

$$v \in \{-v_{\max}, v_{\max}\}, |\phi| \leq \phi_{\max} < \frac{\pi}{2}$$

$$v = v_{\max}, |\phi| \leq \phi_{\max} < \frac{\pi}{2}$$

→ Simple car model

→ Reeds&Shepp's car

→ Dubins' car

References: (1) J.-P. Laumond. Robot Motion Planning and Control. 1998. (2) S. LaValle. Planning algorithms, 2006.

From kinematic to dynamic models

- A kinematic state space model should be interpreted only as a subsystem of a more general dynamical model
- Improvements to the previous kinematic models can be made by placing **integrators** in front of action variables
- For example, for the unicycle model, one can set the speed as the integration of an action a representing acceleration, that is

$$\dot{x} = v \cos \theta, \quad \dot{y} = v \sin \theta, \quad \dot{\theta} = \omega, \quad \dot{v} = a$$

AA 274

Principles of Robotic Autonomy

The Robot Operating System (ROS)



Writing Software for Robotics

- Robotics requires very complex software
- The software you will deal with in AA274A has **way** more moving parts than what you've dealt with in most other classes...



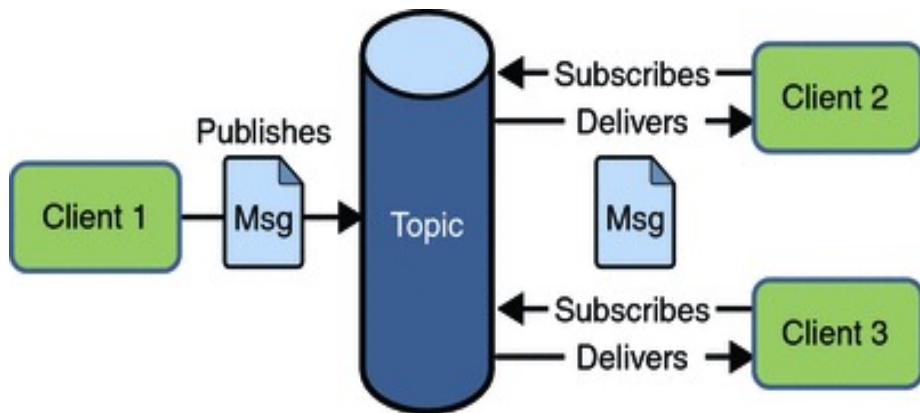
Writing Software for Robotics

- We deal with the complexity through **modularity**
- We enable modularity by following the right **design pattern**: “a general, reusable solution to a commonly occurring problem within a given context in software design” – Wikipedia

The Pub/Sub Design Pattern

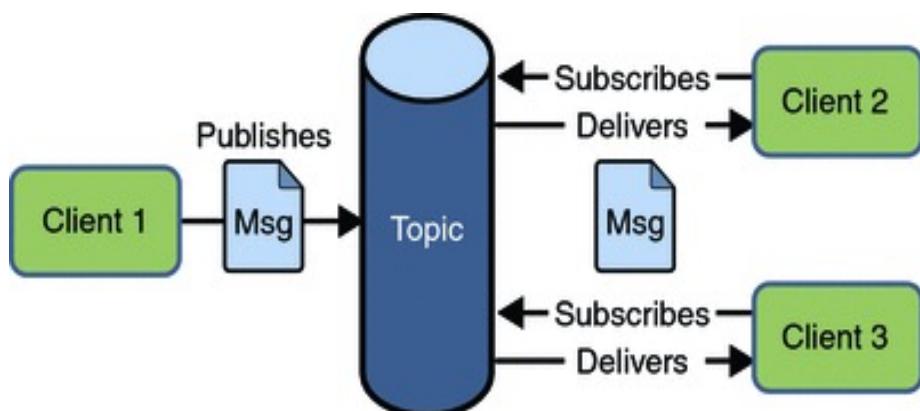
- We divide our software into individual components
- We define “topics” (think chat rooms) where components can broadcast information to anyone listening
- Each component can:
 - *Publish*: send messages to a topic regardless of whether someone is listening or not
 - *Subscribe*: receive messages on a topic if anyone is sending them regardless of who

The Pub/Sub Design Pattern



Note: there are countless ways to **IMPLEMENT** pub/sub!

The Pub/Sub Design Pattern



Note: there are countless ways to **IMPLEMENT** pub/sub!

You already use
Pub/Sub every day!
Where???

Alternatives to Pub/Sub

- Request/Reply (RPC)
- Push/Pull
- Data binding (e.g. shared data members)
- Observers

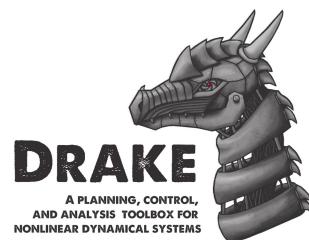
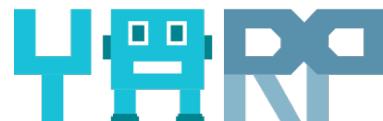
What is ROS?

Depending on who you are talking to...

- An **implementation of pub/sub** geared towards robotic applications and that is network-aware
- Lots of open-source software shared by the community:
 - SLAM (gmapping, amcl)
 - Vision (OpenCV, PCL, OpenNI)
 - Arm Navigation (MoveIt)
 - Simulation (Gazebo)

Are there “Alternatives” to ROS?

- LCM
- Drake
- Player
- YARP
- Orocos
- MRPT
- And many others!



9/29/22

AA 274A | Lecture 2

9

Why is ROS popular in industry?

- Not reinventing the wheel is generally good
- Robotics is hard! It's great to offload some of the work to smart people
- ROS is now 12 years old and still going strong



9/29/22

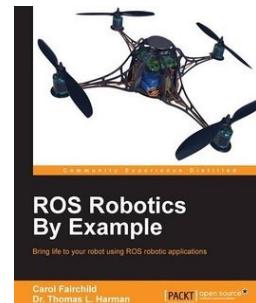
AA 274A | Lecture 2

10

Why are we using ROS in AA274?

- The closest thing we have to an “industry standard”
- It’s an insurance policy for you (stability, online teaching resources)

The screenshot shows the official ROS website at ros.org. The header includes a logo, a search bar, and links for About, Support, Status, and answers.ros.org. Below the header is a navigation menu with Documentation, Browse Software, News, and Download. A sidebar on the left contains sections for Documentation (with sub-links like Install, Getting Started, Tutorials, Contributing, and Support), Software (with sub-links like Distributions, Packages, Core Libraries, and Gazebo), and Resources (with sub-links like API Reference, Tools, and Glossary). The main content area features a large image of a sea turtle swimming over a coral reef, with the text "Community, Collaboration, Distilled".



9/29/22

AA 274A | Lecture 2

11

ROS – Robot Operating System

- 2007-Today
 - Stanford AI Robot (STAIR)
 - Willow Garage founded by Scott Hassan (eGroups, Google, Stanford Digital Libraries)
 - Willow awards 11 \$400k PR2 robots to Universities
 - OSRF (Open Source Robotics Foundation) created to maintain ROS and Gazebo
 - ROS is everywhere!

9/29/22

AA 274A | Lecture 2

13

ROS Integrates Existing Projects

- OpenCV (computer vision)
- Stage, Gazebo (simulation)
- OpenSLAM (navigation)
- Orococos KDL (arm navigation)
- Many ROS “wrappers” to existing software

The Main Software Components

- 1) Master
- 2) Nodes

- Nodes talk to each other over topics (think chat rooms). Master coordinates the whole thing
- Message types: abstraction away from specific hardware
 - Camera image
 - Laser scan data
 - Motion control

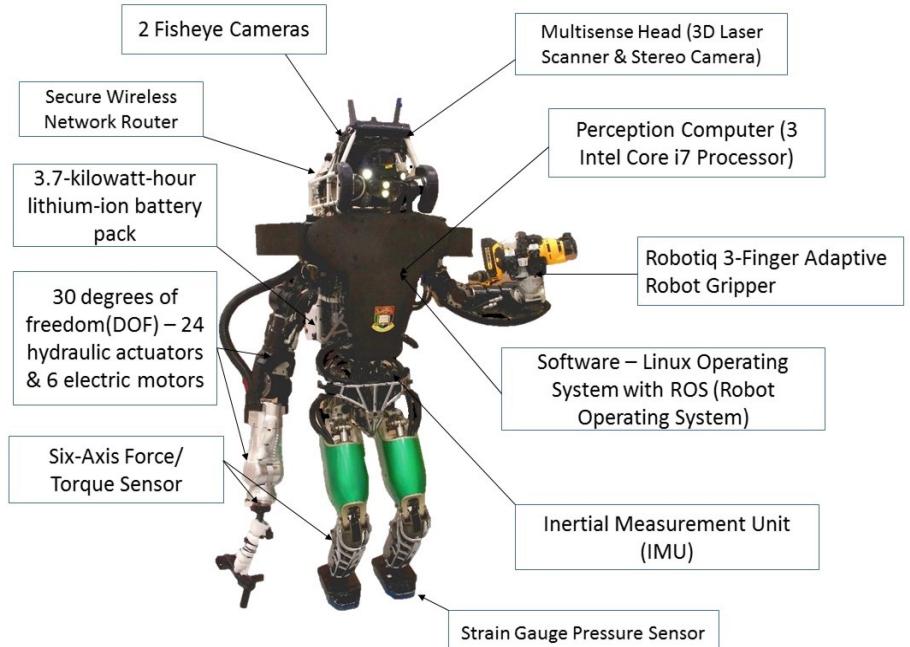
ROS Node

- A process (typically Python or C++) that runs some computation
- The “fundamental” building block
- Can act as a subscriber, publisher or both
- Nodes talk to each other over “topics”
- Run them using `rosrun <package> <node>`
- Initialize using `rospy.init_node()`

Note: nodelets are different. They are not individual processes, they share memory

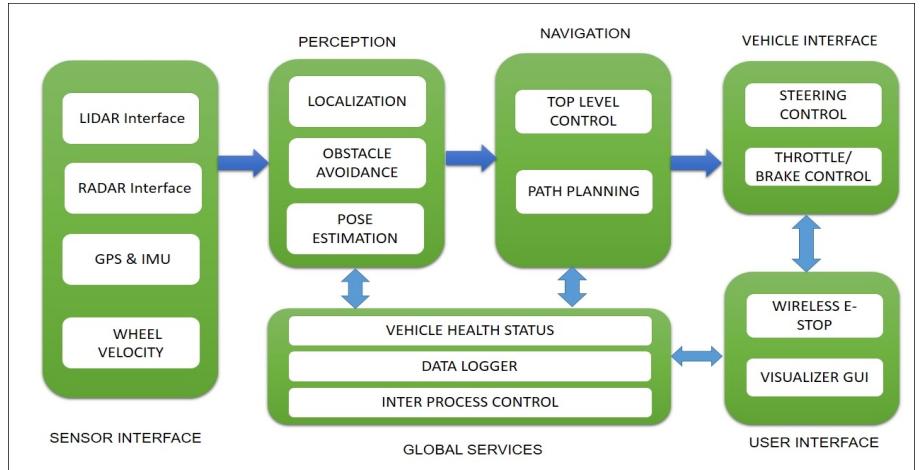
Node Examples

Sensors and actuators are wrapped in self-contained, reusable software containers called “nodes”



Node Examples

Higher level operations also become nodes in the ROS computational architecture



More Concrete Node Examples

- LiDAR node publishes laser scan arrays
- Camera node publishes RGB images (+depth if RGBD) and camera info (resolution, distortion coefficients)
- Mobile robot controller publishes odometry values (e.g. x-y coordinates and velocities, +z for UAVs or underwater vehicles)
- Navigation node subscribes to LiDAR and odometry messages, publishes motion control messages

ROS Master

- A process that is in charge of coordinating nodes, publishers and subscribers
- Also provides a global parameter server
- Exactly one of them running at any time
- Messages do NOT go through Master (i.e. peer-to-peer)
- Nodes will not be able to find each other without Master

Sending Messages

- `pub = rospy.Publisher()`
- `msg = ...`
- `pub.publish(msg)`

ROS Node - Publisher

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def talker():
    rospy.init_node('talker', anonymous=True)

    pub = rospy.Publisher('chatter', String, queue_size=10)

    rate = rospy.get_param('~rate', 1)
    ros_rate = rospy.Rate(rate)

    rospy.loginfo("Starting ROS node talker...")

    while not rospy.is_shutdown():
        msg = "Greetings humans!"

        pub.publish(msg)
        ros_rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

9/29/22

AA 274A | Lecture 2

22

Monitoring Messages

- You can check if you are sending messages using the *rostopic* command line tool:
 - *rostopic list* – lists all the active topics
 - *rostopic echo <topic>* – prints messages received on <topic>
 - *rostopic hz <topic>* – measures topic publishing rate

9/29/22

AA 274A | Lecture 2

23

Receiving Messages

- `rospy.Subscriber("chatter", String, callback)`
- `def callback(msg) : ...`

(in C++ need to call `spinOnce()`, not in Python)

ROS Node - Subscriber

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(msg):
    rospy.loginfo("Received: %s", msg.data)

def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("chatter", String, callback)
    rospy.loginfo("Listening on the chatter topic...")
    rospy.spin()

if __name__ == '__main__':
    listener()
```

ROS Launch Files

- Simple XML files that allow you to
 - Launch multiple nodes at once
 - Set parameters for those nodes
 - Start Master
- `roslaunch <package> <file>.launch`

ROS Launch File Example

```
<launch>
  <!-- Start the talker node -->
  <node name="talker" pkg="aa274" type="talker.py" output="screen">
    <param name="rate" value="5"/>
  </node>
</launch>
```

A Case Study

- Edge detection in camera images

Node 1 – Camera Driver
Subscribes to: Nothing
Publishes: Camera images

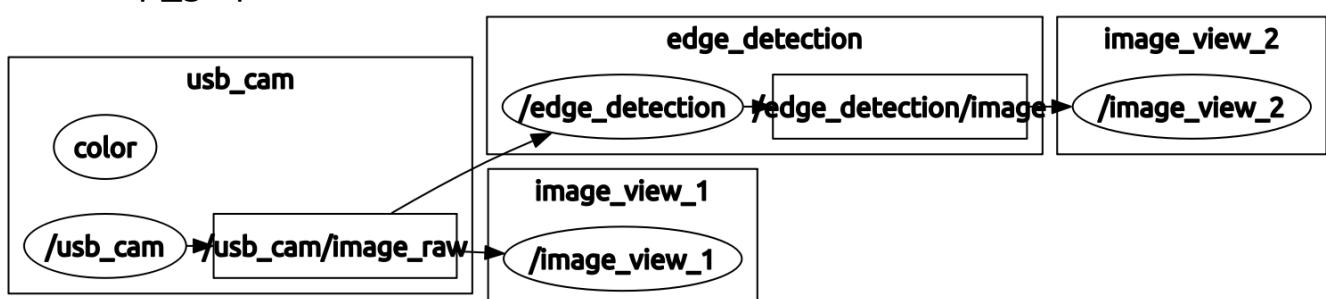
Node 2 – Edge Detection
Subscribes to: Camera images
Publishes: Image with edges

Node 3 – image_view
Subscribes to: Camera images
Publishes: Nothing

Node 4 – image_view
Subscribes to: Image with edges
Publishes: Nothing

A Case Study

- Edge detection in camera image
- rqt_graph



ROS Launch File for Edge Detection

```
<launch>
  <arg name="video_device" default="/dev/video0" />

  <include file="$(find aa274)/launch/usbcam_driver.launch">
    <arg name="video_device" value="$(arg video_device)" />
  </include>

  <node name="image_view_1" pkg="image_view" type="image_view">
    <remap from="image" to="/camera/image_color" />
    <param name="autosize" value="true"/>
  </node>

  <node name="image_view_2" pkg="image_view" type="image_view">
    <remap from="image" to="/edge_detection/image" />
    <param name="autosize" value="true" />
  </node>

  <node name="edge_detection" pkg="opencv_apps" type="edge_detection">
    <remap from="image" to="/camera/image_color" />
    <param name="debug_view" value="false" />
  </node>
</launch>
```

Developing with ROS

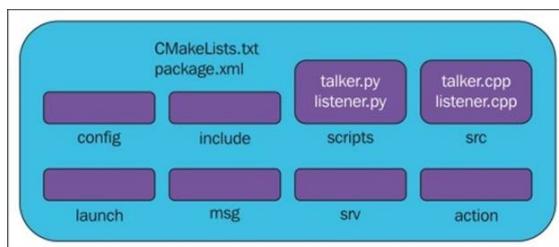
- **Catkin workspace:** a directory that contains all your ROS development
- It sets the right environment variables
- It knows how to compile your nodes (*using cmake which in turn uses a compiler*)

The commands you need to know:

- `mkdir -p ~/catkin_ws/src`
- `cd ~/catkin_ws`
- `catkin_make`

ROS Packages

- The basic organization structure for your nodes
- Usually corresponds to a “functionality” (e.g. a SLAM package)
- Can contain code for multiple nodes
- Directory structure:

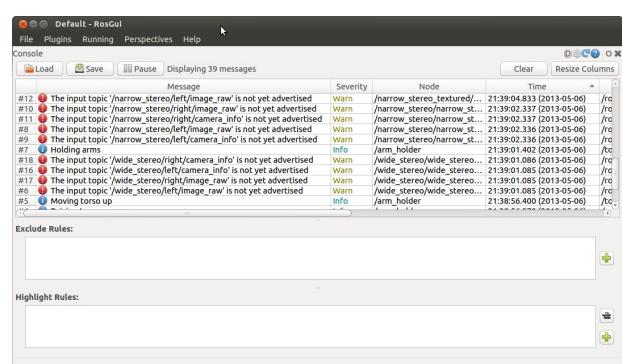


The command you need to know:

```
catkin_create_pkg <name> roscpp rospy std_msgs
```

Debugging

- `rospy.loginfo()`
- `rqt_console`
- `rosbag record <topic>`
- `rosbag play file.bag`
- **pdb – Python Debugger**
 - `import pdb`
 - `pdb.set_trace()`



Creating Custom Messages

- Write message definitions (.msg) that are language agnostic
- ROS generates the right files so that roscpp and rospy can use your message
- `rosmsg show student`

```
[aa274/Student] :  
string name_first  
string name_last  
uint8 age  
uint32 grade
```

Primitive Type	Serialization	C++	Python
bool (1)	unsigned 8-bit int	<code>uint8_t</code> (2)	<code>bool</code>
int8	signed 8-bit int	<code>int8_t</code>	<code>int</code>
uint8	unsigned 8-bit int	<code>uint8_t</code>	<code>int (3)</code>
int16	signed 16-bit int	<code>int16_t</code>	<code>int</code>
uint16	unsigned 16-bit int	<code>uint16_t</code>	<code>int</code>
int32	signed 32-bit int	<code>int32_t</code>	<code>int</code>
uint32	unsigned 32-bit int	<code>uint32_t</code>	<code>int</code>
int64	signed 64-bit int	<code>int64_t</code>	<code>long</code>
uint64	unsigned 64-bit int	<code>uint64_t</code>	<code>long</code>
float32	32-bit IEEE float	<code>float</code>	<code>float</code>
float64	64-bit IEEE float	<code>double</code>	<code>float</code>
string	ascii string (4)	<code>std::string</code>	<code>str</code>
time	secs/nsecs unsigned 32-bit ints	<code>ros::Time</code>	<code>rospy.Time</code>
duration	secs/nsecs signed 32-bit ints	<code>ros::Duration</code>	<code>rospy.Duration</code>

ROS Services

- A different way for nodes to pass messages to each other
- Request/Response scheme (not Pub/Sub!)
- Examples:
 - Turn a light or LED on or off
 - Assign a name to a face and retrain face recognizer
 - Spawn a new model in the Gazebo simulator

The Parameter Server

- Parameters are stored under namespaces; e.g.
 - /move_base/local_costmap/height
 - /usb_cam/framerate
 - /gazebo/time_step
- Setting and getting parameters:
 - rosparam set param_name param_value
 - param_value = rospy.get_param("param_name")
- NOTE: Setting a parameter does not affect a running node!

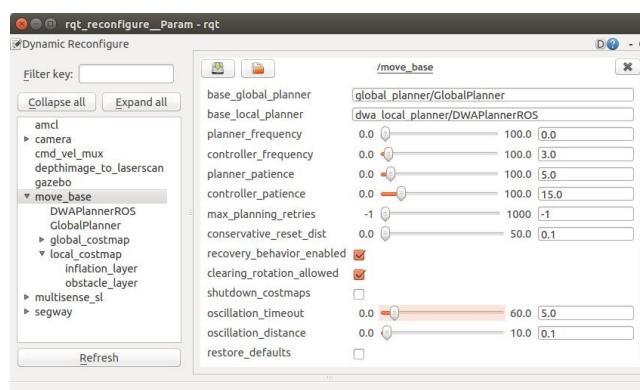
9/29/22

AA 274A | Lecture 2

36

Dynamic Reconfigure

- Some nodes provide dynamically changeable parameters
 - rosrun rqt_reconfigure rqt_reconfigure



9/29/22

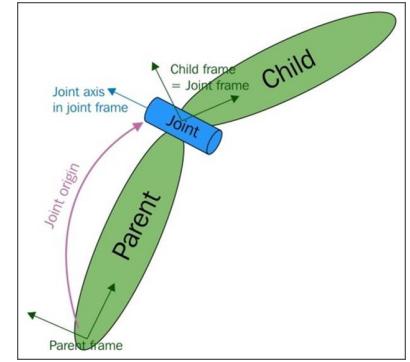
AA 274A | Lecture 2

37

URDF

- Universal Robot Description Format
- An XML file that describes the kinematic chain of your robot

```
<link name="base_link">
  <visual>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
    <material name="blue">
      <color rgba="0 0 .8 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="10"/>
    <inertia ixx="0.4" ixy="0.0" ixz="0.0" iyy="0.4" iyz="0.0" izz="0.2"/>
  </inertial>
</link>
```



9/29/22

AA 274A | Lecture 2

38

Gazebo

- Same code that will run in production
- Physics is mostly accurate



9/29/22

AA 274A | Lecture 2

39

Some more libraries you will hear about...

- TF: coordinate frame transform library
- Actionlib: processes with goals and feedback
- dynamic_reconfigure: making nodes configurable on the fly

Getting help

- ROS wiki (<http://wiki.ros.org/>)
- Github
- Stack Overflow
- The Construct / Robot Ignite Academy
- Google :)

Principles of Robot Autonomy I

Open-loop motion control and differential flatness



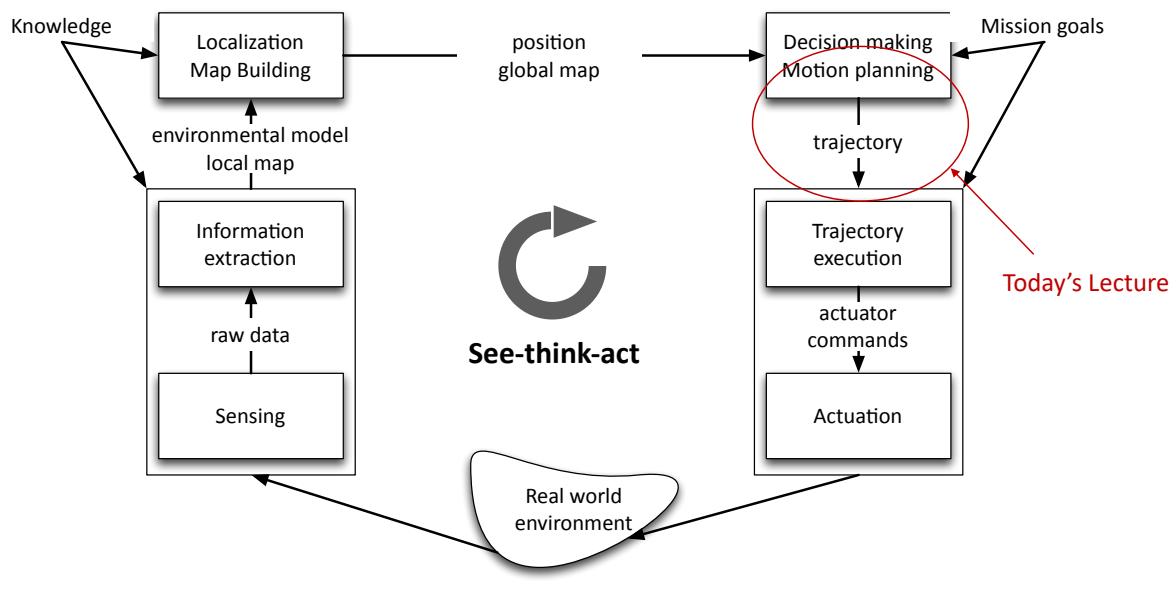
Stanford
University



Logistics

- Masks
- New TA: Aniket Bhatia
- Homework 1 due in one week: Tuesday, 10/11 (11:59PM)
- Sections start this week
 - Section 1: intro to programming tools; ROS teaser
- Waitlist is being processed
 - permission codes will be sent soon according to criteria we outlined
 - After that, waitlist process will run automatically
- High Resolution Course feedback
- iPad notes

The see-think-act cycle



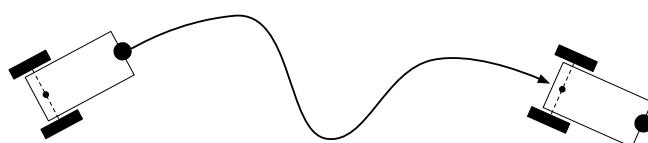
10/3/22

AA 274A | Lecture 3

3

Motion control

- Given a nonholonomic system, how to control its motion from an initial configuration to a final, desired configuration



- Aim**
 - Learn about main techniques in optimal control and trajectory optimization
 - Learn about differential flatness and its use for trajectory optimization
- Readings**
 - B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Robotics: modelling, planning and control. 2010. Chapter 11.

10/3/22

AA 274A | Lecture 3

4

Kinematic / dynamic models

- In lecture 1 we saw how to derive models that describe the equations of motion of a robot in the form of differential equations (DE)

$$\dot{\mathbf{x}}(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t)$$

- DEs are equations relating the derivatives of an unknown function to the unknown function itself and known quantities
- DEs can be *integrated* numerically, for example, via the **Euler method**

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t_i \mathbf{a}(\mathbf{x}_i, \mathbf{u}_i, t_i), \quad i = 0, \dots, N - 1$$

where $\Delta t_i = t_{i+1} - t_i$, $\mathbf{u}_i = \mathbf{u}(t_i)$, and $\mathbf{x}_0 = \mathbf{x}(t_0)$

Optimal control problem

The problem:

$$\begin{aligned} \min_{\mathbf{u}} \quad & h(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t) dt \\ \text{subject to} \quad & \dot{\mathbf{x}}(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t) \\ & \mathbf{x}(t) \in \mathcal{X}, \quad \mathbf{u}(t) \in \mathcal{U} \end{aligned}$$

where $\mathbf{x}(t) \in R^n$, $\mathbf{u}(t) \in R^m$, and $\mathbf{x}(t_0) = \mathbf{x}_0$

- We'll focus on the case $\mathcal{X} = R^n$; state constraints will be addressed in the context of **motion planning**
- Good reference: D. K. Kirk. Optimal Control Theory: An introduction. 2004.

Form of optimal control

- If a functional relationship of the form

$$\mathbf{u}^*(t) = \pi(\mathbf{x}(t), t)$$

can be found, then the optimal control is said to be in *closed-loop* form

- If the optimal control law is determined as a function of time for a specified initial state value

$$\mathbf{u}^*(t) = \mathbf{f}(\mathbf{x}(t_0), t)$$

then the optimal control is said to be in *open-loop* form

- A good compromise: two-step design

$$\mathbf{u}^*(t) = \mathbf{u}_d(t) + \pi(\mathbf{x}(t), \mathbf{x}(t) - \mathbf{x}_d(t))$$

Reference trajectory
Reference control (open-loop) Trajectory-tracking law (closed-loop) Tracking error

Open-loop control

- We want to find

$$\mathbf{u}^*(t) = \mathbf{f}(\mathbf{x}(t_0), t)$$

- In general, two broad classes of methods:

1. **Direct methods:** transcribe infinite problem into finite dimensional, nonlinear programming (NLP) problem, and solve NLP \Rightarrow “First discretize, then optimize”
2. **Indirect methods:** attempt to find a minimum point “indirectly,” by solving the necessary conditions of optimality \Rightarrow “First optimize, then discretize”

Open-loop control

- We want to find

$$\mathbf{u}^*(t) = \mathbf{f}(\mathbf{x}(t_0), t)$$

- In general, two broad classes of methods:

1. **Direct methods**: transcribe infinite problem into finite dimensional, nonlinear programming (NLP) problem, and solve NLP \Rightarrow “First discretize, then optimize”
 2. **Indirect methods**: attempt to find a minimum point “indirectly,” by solving the necessary conditions of optimality \Rightarrow “First optimize, then discretize”
- For an in-depth study of direct and indirect methods, see AA203 “Optimal and Learning-based Control” (Spring 2023)

Direct methods: nonlinear programming transcription

$$\min \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t) dt$$

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t), \quad t \in [t_0, t_f] \\ (\textbf{OCP}) \end{aligned}$$

$$\mathbf{x}(0) = \mathbf{x}_0, \quad \mathbf{x}(t_f) \in M_f$$

$$\mathbf{u}(t) \in U \subseteq \mathbb{R}^m, \quad t \in [t_0, t_f]$$

Direct methods: nonlinear programming transcription

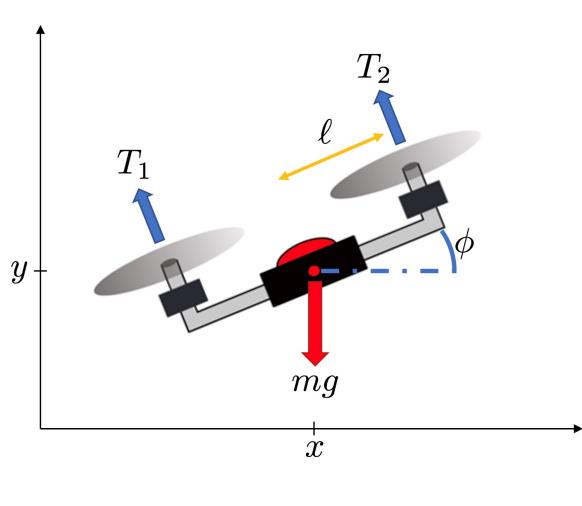
Forward Euler time discretization

$$\begin{aligned} \min & \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t) dt \\ (\mathbf{OCP}) \quad & \dot{\mathbf{x}}(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t), \quad t \in [t_0, t_f] \\ & \mathbf{x}(0) = \mathbf{x}_0, \quad \mathbf{x}(t_f) \in M_f \\ & \mathbf{u}(t) \in U \subseteq \mathbb{R}^m, \quad t \in [t_0, t_f] \end{aligned}$$

1. Select a discretization $0 = t_0 < t_1 < \dots < t_N = t_f$ for the interval $[t_0, t_f]$ and, for every $i = 0, \dots, N - 1$, define $\mathbf{x}_i \sim \mathbf{x}(t_i)$, $\mathbf{u}_i \sim \mathbf{u}(t_i)$, $t \in (t_i, t_{i+1}]$ and $\mathbf{x}_0 \sim \mathbf{x}(0)$
2. By denoting $\Delta t_i = t_{i+1} - t_i$, (\mathbf{OCP}) is transcribed into the following nonlinear, constrained optimization problem

$$\begin{aligned} \min_{(\mathbf{x}_i, \mathbf{u}_i)} & \sum_{i=0}^{N-1} \Delta t_i g(\mathbf{x}_i, \mathbf{u}_i, t_i) \\ (\mathbf{NLOP}) \quad & \mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t_i \mathbf{a}(\mathbf{x}_i, \mathbf{u}_i, t_i), \quad i = 0, \dots, N - 1 \\ & \mathbf{u}_i \in U, i = 0, \dots, N - 1, \quad F(\mathbf{x}_N) = 0 \end{aligned}$$

Illustrative example: planar quadrotor



$$\min \int_0^{t_f} T_1(t)^2 + T_2(t)^2 dt \quad (\text{energy objective})$$

subject to dynamics

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ \frac{-(T_1+T_2) \sin \phi}{m} \\ v_y \\ \frac{(T_1+T_2) \cos \phi}{m} - g \\ \omega \\ \frac{(T_2-T_1)\ell}{I_{zz}} \end{bmatrix}$$

Direct methods: software packages

Some software packages:

- DIDO: <http://www.elissarglobal.com/academic/products/>
- PROPT: <http://tomopt.com/tomlab/products/propt/>
- GPOPS: <http://www.gpops2.com/>
- CasADI: <https://github.com/casadi/casadi/wiki>
- ACADO: <http://acado.github.io/>
- Trajax: <https://github.com/google/trajax>

In addition to implementing efficient trajectory optimization algorithms, many of these tools provide easier-to-use modeling languages for problem specification.

Differential flatness

- Computing “good” feasible trajectories is often sufficient for trajectory generation purposes, and typically much faster than computing optimal ones
- A class of systems for which trajectory generation is particularly easy are the so-called **differentially flat systems**
- Reference: M. J. Van Nieuwstadt and R. M. Murray. Real-time trajectory generation for differentially flat systems. 1998.

Motivating example: simple car

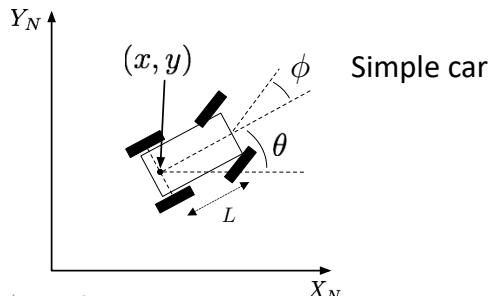
Consider the problem of finding a *feasible* solution that satisfies the dynamics:

$$\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u}), \quad \mathbf{x}(0) = \mathbf{x}_0, \quad \mathbf{x}(t_f) = \mathbf{x}_f$$

Example: simple car steering

$$\dot{x} = \cos \theta v \quad \dot{y} = \sin \theta v, \quad \dot{\theta} = \frac{v}{L} \tan \phi$$

- State: (x, y, θ)
- Inputs: (v, ϕ)



Structure of the dynamics for simple car steering

- Suppose we are given a (smooth) trajectory for the rear wheels of the system, $x(t)$ and $y(t)$
 1. we can use this solution to solve for the angle of the car by writing

$$\frac{\dot{y}}{\dot{x}} = \frac{\sin \theta}{\cos \theta} \quad \Rightarrow \quad \theta = \tan^{-1} \left(\frac{\dot{y}}{\dot{x}} \right)$$

2. we can solve for the velocity

$$\dot{x} = v \cos \theta \quad \Rightarrow \quad v = \dot{x} / \cos \theta \quad (\text{or } v = \dot{y} / \sin \theta)$$

3. and finally

$$\dot{\theta} = \frac{v}{L} \tan \phi \quad \Rightarrow \quad \phi = \tan^{-1} \left(\frac{L \dot{\theta}}{v} \right)$$

Structure of the dynamics for simple car steering

- **Bottom line:** all of the state variables and the inputs can be determined by the trajectory of the rear wheels and its derivatives!
- We say that the system is *differentially flat* with *flat output* $\mathbf{z} = (x, y)$
- This provides a dramatic simplification for the purposes of trajectory generation (more on this later)

Differential flatness

Differential flatness: A nonlinear system $\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u})$ is differentially flat if there exists a function α such that

$$\mathbf{z} = \alpha(\mathbf{x}, \mathbf{u}, \dots, \mathbf{u}^{(p)})$$

and we can write the solutions of the nonlinear system as functions of \mathbf{z} and a finite number of derivatives

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \dots, \mathbf{z}^{(q)})$$

$$\mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \dots, \mathbf{z}^{(q)})$$

In words, a system is differentially flat if we can find a set of outputs (equal in number to the number of inputs) such that all states and inputs can be determined from these outputs *without integration*

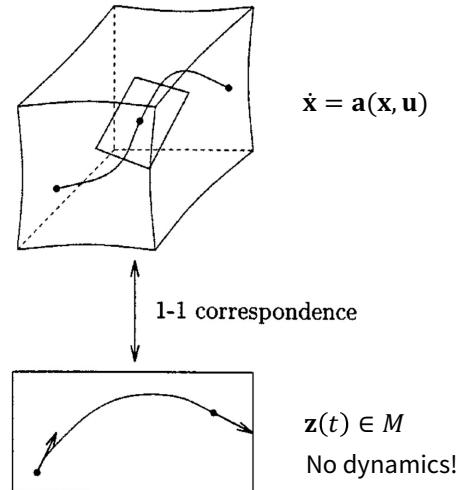
Differential flatness

- Implication for trajectory generation: to every smooth, differentiable curve $t \rightarrow \mathbf{z}(t)$, there is a corresponding trajectory

$$t \rightarrow \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{u}(t) \end{pmatrix} = \begin{pmatrix} \beta(\mathbf{z}(t), \dot{\mathbf{z}}(t), \dots, \mathbf{z}^{(q)}(t)) \\ \gamma(\mathbf{z}(t), \dot{\mathbf{z}}(t), \dots, \mathbf{z}^{(q)}(t)) \end{pmatrix}$$

that identically satisfies the system equations

- The simple car is differentially flat with the position of the rear wheels as the flat output



From Nieuwstadt, Murray. 1998.

Another example: planar quadrotor

$$\mathbf{z} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\left. \begin{array}{l} x = x \\ v_x = \dot{x} \\ v_y = \dot{y} \\ \phi = \tan^{-1} \left(-\frac{\ddot{x}}{\ddot{y} + g} \right) \\ \omega = \frac{\ddot{y} \ddot{x} - (\ddot{y} + g) \ddot{x}}{(\ddot{y} + g)^2 + \ddot{x}^2} \end{array} \right\} \beta$$

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \ddot{\mathbf{z}})$$

$$\mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \ddot{\mathbf{z}}, \ddot{\mathbf{z}})$$

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ \frac{-(T_1+T_2) \sin \phi}{m} \\ v_y \\ \frac{(T_1+T_2) \cos \phi}{m} - g \\ \omega \\ \frac{(T_2-T_1)\ell}{I_{zz}} \end{bmatrix}$$

Practical implications

This leads to a simple, yet effective strategy for trajectory generation

1. Find the initial and final conditions for the flat output:

Given	Find
$(t_0, \mathbf{x}(t_0), \mathbf{u}(t_0))$	$(\mathbf{z}(t_0), \dot{\mathbf{z}}(t_0), \dots, \mathbf{z}^{(q)}(t_0))$
$(t_f, \mathbf{x}(t_f), \mathbf{u}(t_f))$	$(\mathbf{z}(t_f), \dot{\mathbf{z}}(t_f), \dots, \mathbf{z}^{(q)}(t_f))$

2. Build a smooth curve $t \rightarrow \mathbf{z}(t)$ for $t \in [t_0, t_f]$ by interpolation, possibly satisfying further constraints
3. Deduce the corresponding trajectory $t \rightarrow (\mathbf{x}(t), \mathbf{u}(t))$

More on step 2

- We can parameterize the flat output trajectory using a set of smooth basis functions $\psi_i(t)$

$$z_j(t) = \sum_{i=1}^N \alpha_i^{[j]} \psi_i(t)$$

- and then solve (Problem 1 in pset)

$$\begin{bmatrix} \psi_1(t_0) & \psi_2(t_0) & \dots & \psi_N(t_0) \\ \dot{\psi}_1(t_0) & \dot{\psi}_2(t_0) & \dots & \dot{\psi}_N(t_0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_0) & \psi_2^{(q)}(t_0) & \dots & \psi_N^{(q)}(t_0) \\ \psi_1(t_f) & \psi_2(t_f) & \dots & \psi_N(t_f) \\ \dot{\psi}_1(t_f) & \dot{\psi}_2(t_f) & \dots & \dot{\psi}_N(t_f) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_f) & \psi_2^{(q)}(t_f) & \dots & \psi_N^{(q)}(t_f) \end{bmatrix} \begin{bmatrix} \alpha_1^{[j]} \\ \alpha_2^{[j]} \\ \vdots \\ \alpha_N^{[j]} \end{bmatrix} = \begin{bmatrix} z_j(t_0) \\ \dot{z}_j(t_0) \\ \vdots \\ z_j^{(q)}(t_0) \\ z_j(t_f) \\ \dot{z}_j(t_f) \\ \vdots \\ z_j^{(q)}(t_f) \end{bmatrix}$$

Key points

- Nominal trajectories and inputs can be computed in a computationally-efficient way (solving a set of *algebraic equations*)
- Other constraints on the system, such as input bounds, can be transformed into the flat output space and (typically) become limits on the curvature or higher order derivative properties of the curve
 - Alternative: **time scaling**, i.e., break down trajectory planning in (1) finding a path (via differential flatness) and (2) defining a timing law on the path (**Problem 1 in pset**) -- more on this next time
- If there is a performance index for the system, this index can be transformed and becomes a functional depending on the flat outputs and their derivatives up to some order

When is a system differentially flat?

- The existence of a general, computable criterion to decide if the dynamical system $\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u})$ is differentially flat remains open
- Some results in this direction are, however, available
- Further readings:
 - Application to trajectory optimization:
 1. M. J. Van Nieuwstadt and R. M. Murray. Real-time trajectory generation for differentially flat systems. 1998
 2. R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. 1995
 3. B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Robotics: modelling, planning and control. 2010
 4. D. Mellinger. Trajectory Generation and Control for Quadrotors. 2012.
 - Theory:
 1. J. Levine. Analysis and control of nonlinear systems: A flatness-based approach. 2009
 2. G. G. Rigatos, Gerasimos. Nonlinear control and filtering using differential flatness approaches: applications to electromechanical systems. 2015

Principles of Robot Autonomy I

Trajectory tracking and closed-loop control



Stanford
University

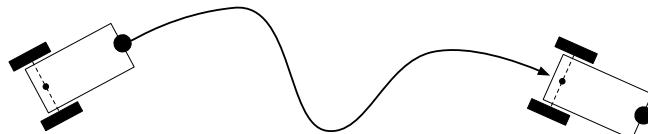


Logistics

- Masks
- Homework 1 due on Tuesday, 10/11 (11:59PM)
- Waitlist is being processed
 - If you got a permission code, please use it right now if you haven't yet
 - Any issues: Let Brian know!

Motion control

- Given a nonholonomic system, how to control its motion from an initial configuration to a final, desired configuration



- Aim
 - Learn how to handle bound constraints via space-time separation
 - Learn about trajectory tracking
 - Learn about closed-loop control
- Readings
 - B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Robotics: modelling, planning and control. 2010. Chapter 11.

Summary of previous lecture

- A nonlinear system $\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}, \mathbf{u})$ is differentially flat if there exists a set of outputs $\mathbf{z} = \alpha(\mathbf{x}, \mathbf{u}, \dots, \mathbf{u}^{(p)})$ such that

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \dots, \mathbf{z}^{(q)})$$

$$\mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \dots, \mathbf{z}^{(q)})$$

- One can then use any interpolation scheme (e.g., piecewise polynomial (spline)) to plan the trajectory of \mathbf{z} in such a way as to satisfy the appropriate boundary conditions
- The evolution of the state variables \mathbf{x} , together with the associated control inputs \mathbf{u} , can then be computed algebraically from \mathbf{z}

Summary of previous lecture

- Constraints on the system can be transformed into the flat output space and (typically) become limits on the curvature or higher order derivative properties of the curve
- An important class of constraints is represented by bounds on some of the system variables, and in particular the inputs, for example:
 $|v(t)| \leq v_{\max}$ and $|\omega(t)| \leq \omega_{\max}$
- Bound constraints can be effectively addressed via **time scaling**

Path and time scaling law

- The problem of planning a trajectory can be divided into two steps:
 1. computing a **path**, that is, a purely geometric description of the sequence of configurations achieved by the robot, and
 2. devising a **time scaling law**, which specifies the times when those configurations are reached
- Mathematically, a trajectory $\mathbf{x}(t)$ can be broken down into a geometric path $\mathbf{x}(s)$ and a timing law $s = s(t)$, with the parameter s varying between $s(t_0) = s_0$ and $s(t_f) = s_f$ in a monotonic fashion, i.e., with $\dot{s}(t) > 0$
- A possible choice for s is the arc length along the path (in this case, $s_0 = 0$, and $s_f = L$, the length of the path)

Enforcing bound constraints

- Such a space-time separation implies that

$$\dot{\mathbf{x}}(t) = \frac{d\mathbf{x}(t)}{dt} = \frac{d\mathbf{x}(s(t))}{ds} \dot{s}(t)$$

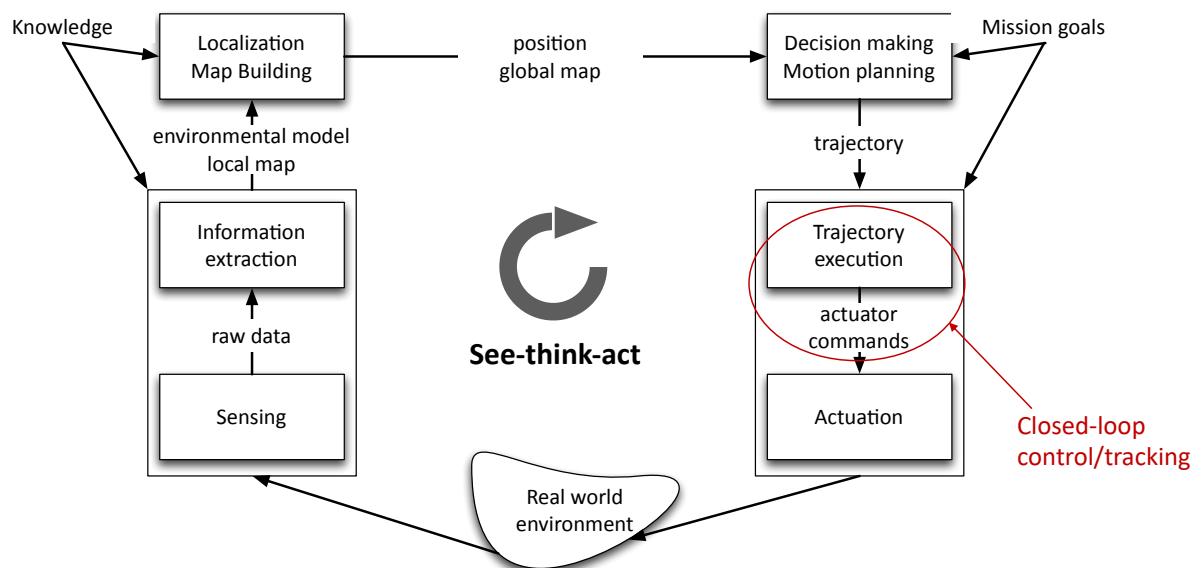
- Thus, once the geometric path is determined, the choice of a timing law $s = s(t)$ will identify a particular trajectory along this path, with a corresponding set of **time-scaled inputs** (**Problem 1 in pset**)

- Example, for unicycle model

- $v(t) = \frac{d|\mathbf{x}(t)|}{dt} = \frac{d|\mathbf{x}(s(t))|}{ds} \dot{s}(t) = \tilde{v}(s)\dot{s}(t)$
- $\omega(t) = \frac{d\theta(t)}{dt} = \frac{d\theta(s(t))}{ds} \dot{s}(t) = \tilde{\omega}(s)\dot{s}(t) = \tilde{\omega}(s) \frac{v(t)}{\tilde{v}(s)}$

- Simplest choice, with s being arc length: $s(t) = t L/T$

The see-think-act cycle



Trajectory tracking

- Back to two-step design strategy

$$\mathbf{u}^*(t) = \mathbf{u}_d(t) + \boxed{\pi(\mathbf{x}(t), \mathbf{x}(t) - \mathbf{x}_d(t))}$$

Tracking control law

- Reference trajectory and control history (i.e., $\mathbf{x}_d(t)$ and $\mathbf{u}_d(t)$) are computed via open-loop techniques (e.g., differential flatness)
- For reference tracking (**Problem 3 in pset**)
 - Geometric (e.g., pursuit) strategies
 - Linearization (either approximate or exact) + linear structure
 - Non-linear control
 - Optimization-based techniques (e.g., MPC)

Trajectory tracking for differentially flat systems

- Key fact (see, e.g., Levine 2009): a differentially flat system can be **linearized** by (dynamic) feedback and coordinate change, that is it can be equivalently transformed into the system

$$\mathbf{z}^{(q+1)} = \mathbf{w}$$

- One can then design a tracking controller for the linearized system by using **linear** control techniques; in particular, for a given reference flat output \mathbf{z}_d , define the *component-wise* error

$$e_i := z_i - z_{i,d}, \text{ which implies } e_i^{(q+1)} = w_i - w_{i,d}$$

- For guaranteed convergence to zero of tracking error, one can set

$$w_i = w_{i,d} - \sum_{j=0}^q k_{i,j} e_i^{(j)},$$

with the gains $\{k_{i,j}\}$ chosen so as to enforce stability

Trajectory tracking for differentially flat systems

- Example: dynamically extended unicycle model

$$\dot{x}(t) = V \cos(\theta(t))$$

$$\dot{y}(t) = V \sin(\theta(t))$$

$$\dot{V}(t) = a(t)$$

$$\dot{\theta}(t) = \omega(t)$$

- The system is differentially flat with flat outputs (x, y) , in particular

$$\begin{bmatrix} \ddot{x}(t) \\ \ddot{y}(t) \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -V \sin(\theta) \\ \sin(\theta) & V \cos(\theta) \end{bmatrix}}_{:=J} \begin{bmatrix} a \\ \omega \end{bmatrix} := \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

Trajectory tracking for differentially flat systems

- Then one can use the following virtual control law for trajectory tracking:

$$w_1 = \ddot{x}_d + k_{px}(x_d - x) + k_{dx}(\dot{x}_d - \dot{x})$$

$$w_2 = \ddot{y}_d + k_{py}(y_d - y) + k_{dy}(\dot{y}_d - \dot{y})$$

where $k_{px}, k_{dx}, k_{py}, k_{dy} > 0$ are control gains

- Such a law guarantees exponential convergence to zero of the Cartesian tracking error

Closed-loop control

- General closed-loop control: we want to find

$$\mathbf{u}^*(t) = \pi(\mathbf{x}(t), t)$$

- Main techniques:

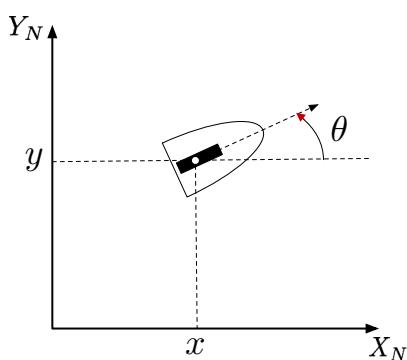
- Hamilton–Jacobi–Bellman equation, dynamic programming
- Lyapunov analysis

For an in-depth study of this topic, see AA203 “Optimal and Learning-based Control” (Spring 2020)

Closed-loop control: posture regulation

- Consider a differential drive mobile robot

$$\begin{aligned}\dot{x}(t) &= V(t) \cos(\theta(t)) \\ \dot{y}(t) &= V(t) \sin(\theta(t)) \\ \dot{\theta}(t) &= \omega(t)\end{aligned}$$



- Inputs: \$V\$ (linear velocity of the wheel) and \$\omega\$ (angular velocity around the vertical axis)
- Goal: drive the robot to the origin \$[0, 0, 0]

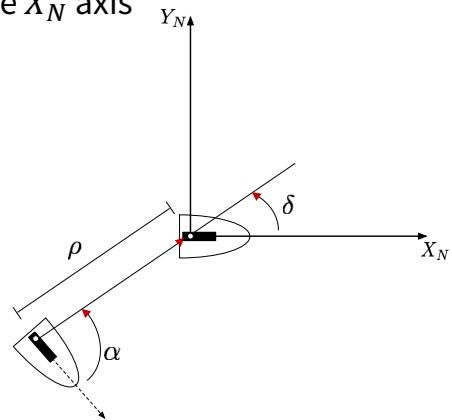
Control based on polar coordinates

- Polar coordinates

- ρ : distance of the reference point of the unicycle from the goal
- α : angle of the pointing vector to the goal w.r.t. the unicycle main axis
- δ : angle of the same pointing vector w.r.t. the X_N axis

- Coordinate transformation

- $\rho = \sqrt{x^2 + y^2}$
- $\alpha = \text{atan}2(y, x) - \theta + \pi$
- $\delta = \alpha + \theta$



10/5/22

AA 274A | Lecture 4

15

Equations in polar coordinates

- In polar coordinates, the unicycle equations become

$$\begin{aligned}\dot{\rho}(t) &= -V(t) \cos(\alpha(t)) \\ \dot{\alpha}(t) &= V(t) \frac{\sin(\alpha(t))}{\rho(t)} - \omega(t) \\ \dot{\delta}(t) &= V(t) \frac{\sin(\alpha(t))}{\rho(t)}\end{aligned}$$

- In order to achieve the goal posture, variables (ρ, α, δ) should all converge to zero

10/5/22

AA 274A | Lecture 4

16

Control law

- Closed-loop control law (**Problem 2 in pset**):

$$V = k_1 \rho \cos(\alpha)$$

$$\omega = k_2 \alpha + k_1 \frac{\sin(\alpha) \cos(\alpha)}{\alpha} (\alpha + k_3 \delta),$$

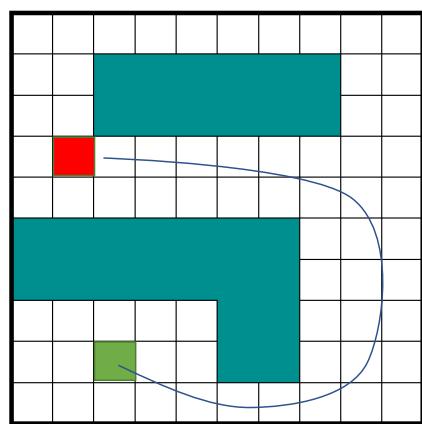
- If $k_1, k_2, k_3 > 0$, then closed-loop system is globally asymptotically driven to the posture $(0,0,0)$!

- For more details, see M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino (1995). Closed loop steering of unicycle like vehicles via Lyapunov techniques. IEEE Robotics & Automation Magazine.

Summary

- We covered closed-loop control along two main dimensions
 1. Trajectory tracking (useful to infuse robustness of point-to-point motion)
 2. Posture regulation (useful for final phase of motion)
- We'll see in Pset 2 how the topics of differential flatness, trajectory tracking, posture regulation, and motion planning will lead to an end-to-end trajectory optimization module

Next time: graph search methods for motion planning



Principles of Robot Autonomy I

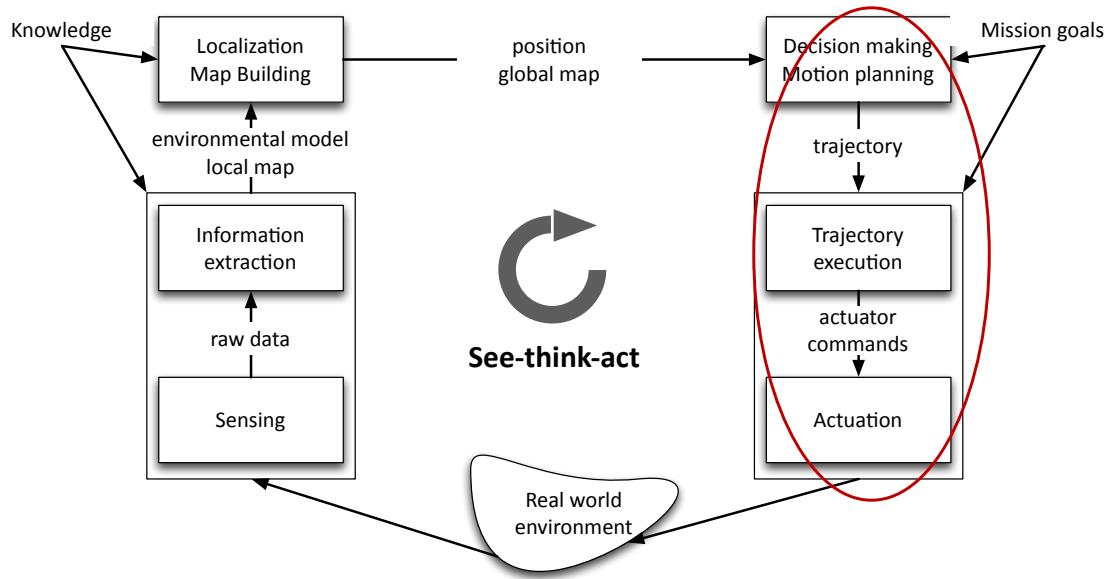
Motion planning I: graph search methods



Logistics

- Masks
- Homework 1 due today (11:59PM)
- Homework 2 will be released today
- Students are being moved off the waitlist
 - If you got a permission code, please use it right now if you haven't yet
 - Any issues: Let Brian know!
 - Decided to drop? Let Brian know!
 - We cannot answer e-mails of students with their spots on the unified waitlist
- Check out the lecture notes!

The see-think-act cycle



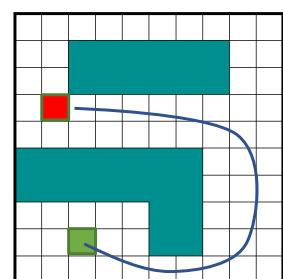
10/12/22

AA 274 | Lecture 5

3

Motion planning

Compute sequence of actions that drives a robot from an initial condition to a terminal condition while avoiding obstacles, respecting motion constraints, and possibly optimizing a cost function



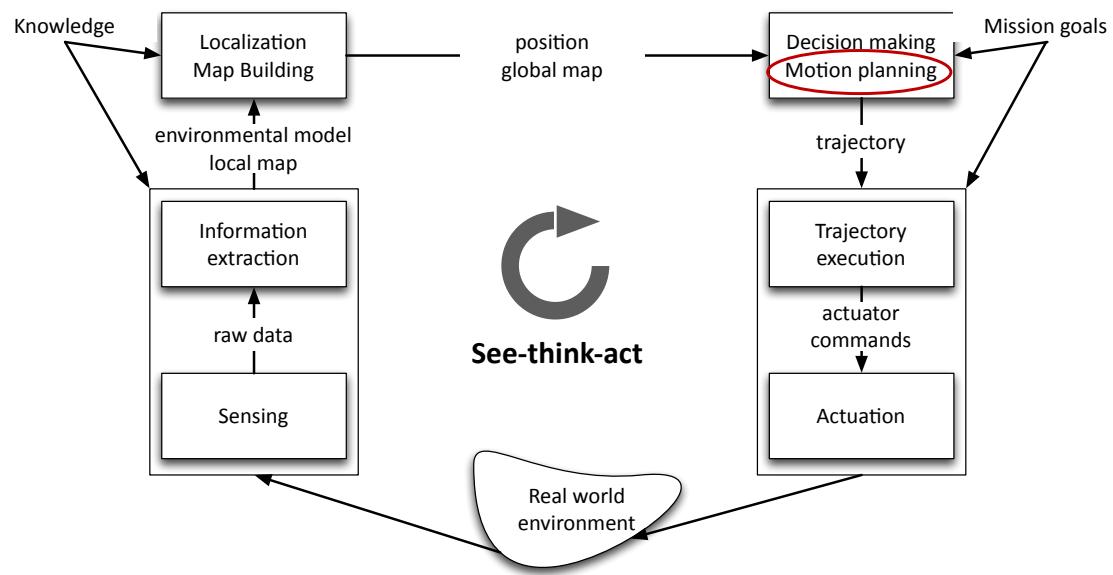
- Aim
 - Introduction to motion planning
 - Learn about search-based methods for motion planning
- Readings:
 - D. Bertsekas. Dynamic Programming and Optimal Control, Vol I. Section 2.3.
 - S. LaValle. Planning Algorithms. Sections 6.1-6.3, 6.5.

10/12/22

AA 274 | Lecture 5

4

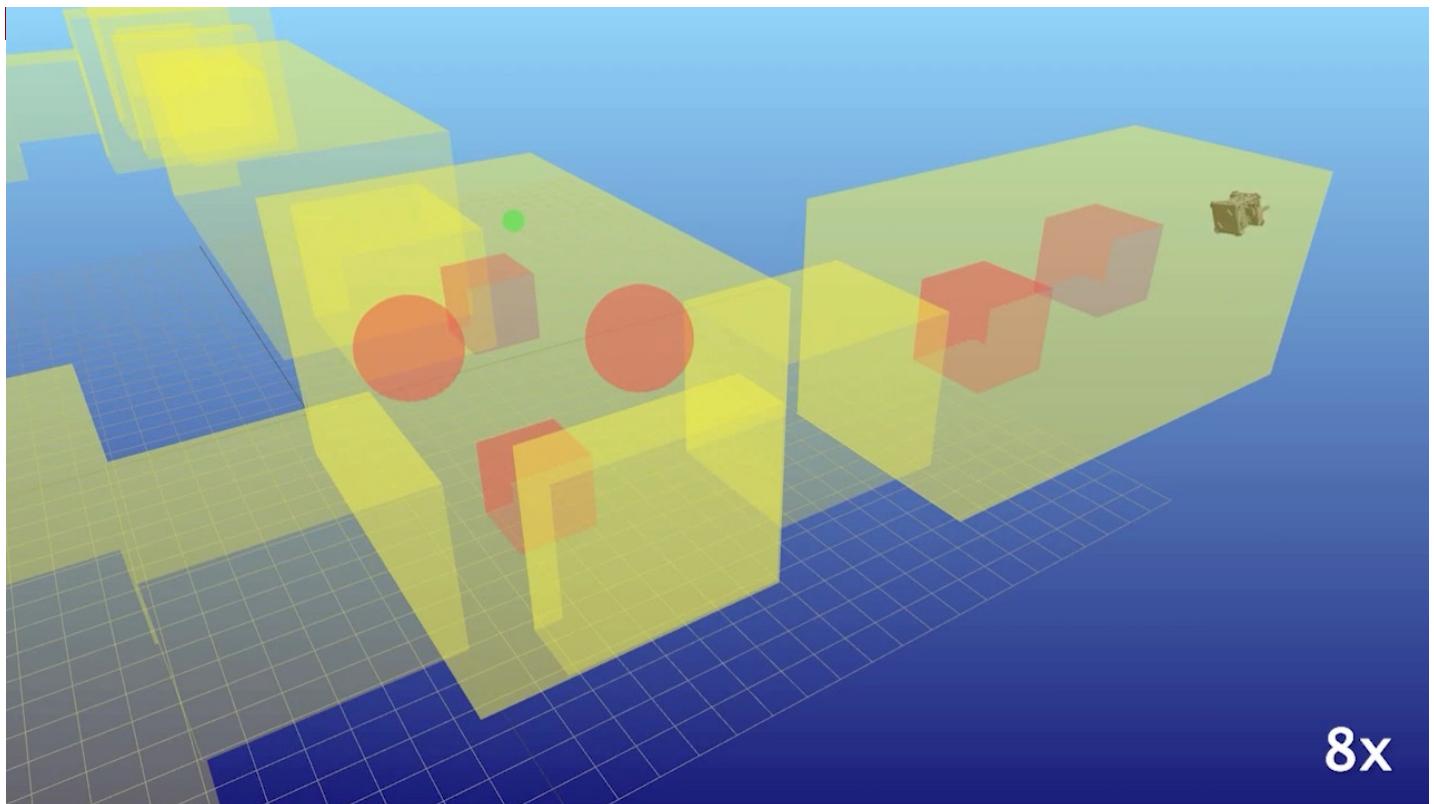
The see-think-act cycle



10/12/22

AA 274 | Lecture 5

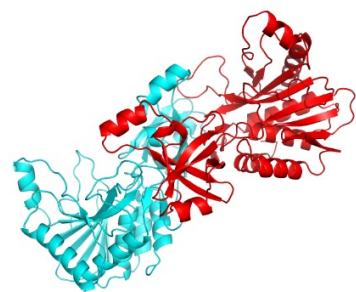
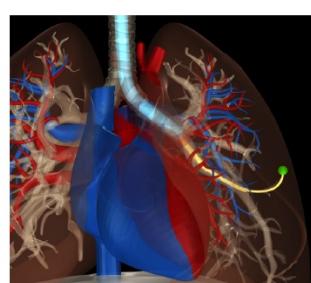
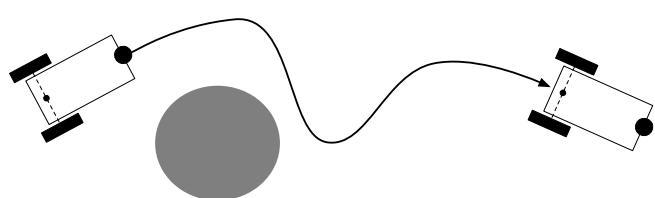
5



Examples from:
<https://ompl.kavrakilab.org/gallery.html>

More examples of motion planning

- Steering autonomous vehicles
- Controlling humanoid robot
- Surgery planning
- Protein folding
- ...

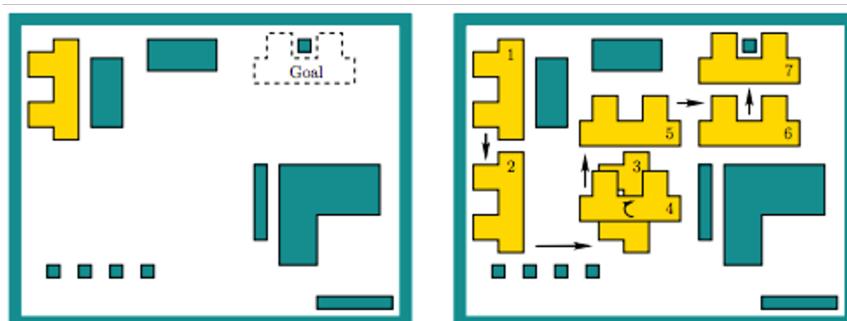


Some history

- Formally defined in the 1970s
- Development of exact, combinatorial solutions in the 1980s
- Development of sampling-based methods in the 1990s
- Deployment on real-time systems in the 2000s
- Current research: inclusion of differential and logical constraints, planning under uncertainty, parallel implementation, feedback plans and more

Simplest setup

- Assume 2D workspace: $\mathcal{W} \subseteq \mathbb{R}^2$
- $\mathcal{O} \subset \mathcal{W}$ is the obstacle region with polygonal boundary
- Robot is a rigid polygon
- **Problem:** given initial placement of robot, compute how to gradually move it into a desired goal placement so that it never touches the obstacle region

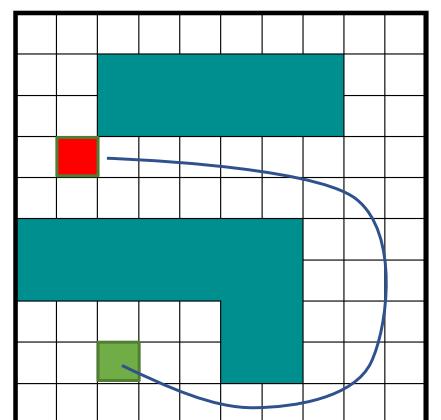


Popular approaches

- *Potential fields* [Rimon, Koditschek, '92]: create forces on the robot that pull it toward the goal and push it away from obstacles
- *Grid-based planning* [Stentz, '94]: discretizes problem into grid and runs a graph-search algorithm (Dijkstra, A*, ...)
- *Combinatorial planning* [LaValle, '06]: constructs structures in the configuration (C-) space that completely capture all information needed for planning
- *Sampling-based planning* [Kavraki et al, '96; LaValle, Kuffner, '06, etc.]: uses collision detection algorithms to probe and incrementally search the C-space for a solution, rather than completely characterizing all of the C_{free} structure

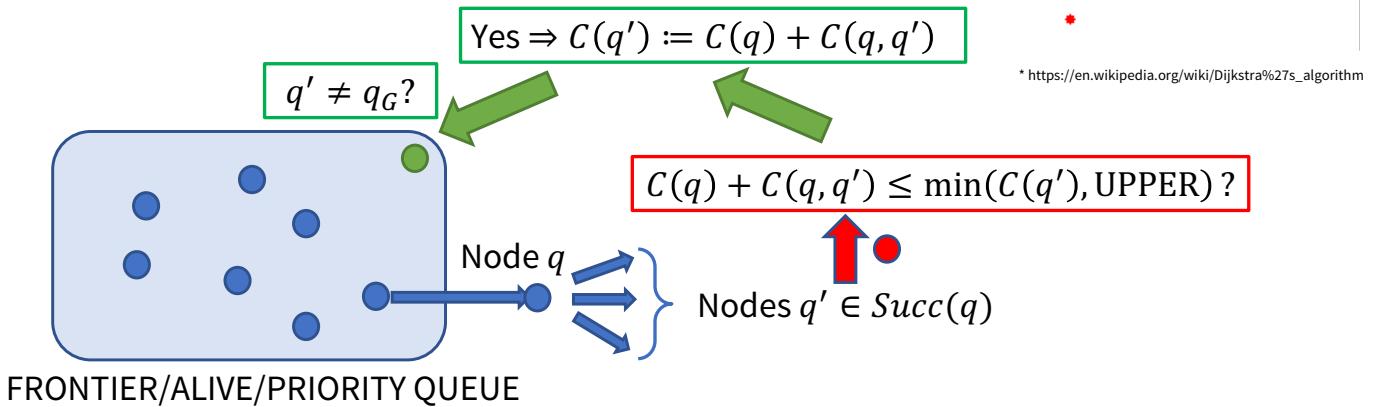
Grid-based approaches

- Discretize the continuous world into a grid
 - Each grid cell is either free or forbidden
 - Robot moves between adjacent free cells
 - **Goal:** find sequence of free cells from start to goal
- Mathematically, this corresponds to pathfinding in a discrete graph $G = (V, E)$
 - Each vertex $v \in V$ represents a free cell
 - Edges $(v, u) \in E$ connect adjacent grid cells



Graph search algorithms

- Having determined decomposition, how to find “best” path?
- Label-Correcting Algorithms:** $C(q)$: cost-of-arrival from q_I to q



FRONTIER/ALIVE/PRIORITY QUEUE

10/12/22

AA 274 | Lecture 5

13

Label correcting algorithm

Step 1. Remove a node q from frontier queue and for each child q' of q , execute step 2

Step 2. If $C(q) + C(q, q') \leq \min(C(q'), \text{UPPER})$, set $C(q') := C(q) + C(q, q')$ and set q to be the parent of q' . In addition, if $q' \neq q_G$, place q' in the frontier queue if it is not already there, while if $q' = q_G$, set UPPER to the new value $C(q) + C(q, q_G)$

Step 3. If the frontier queue is empty, terminate, else go to step 1

Initialization: set the labels of all nodes to ∞ , except for the label of the origin node, which is set to 0

10/12/22

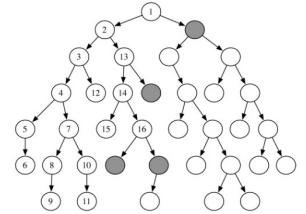
AA 274 | Lecture 5

14

GetNext() ?

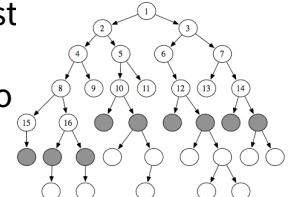
Depth-First-Search (DFS): Maintain Q as a **stack** – Last in/first out

- Lower memory requirement (only need to store part of graph)



Breadth-First-Search (BFS, Bellman-Ford): Maintain Q as a **list** – First in/first first out

- Update cost for all edges up to current depth before proceeding to greater depth
- Can deal with negative edge (transition) costs



Best-First (BF, Dijkstra): Greedily select next q : $q = \operatorname{argmin}_{q \in Q} C(q)$

- Node will enter the frontier queue at most *once*
- Requires costs to be non-negative

10/12/22

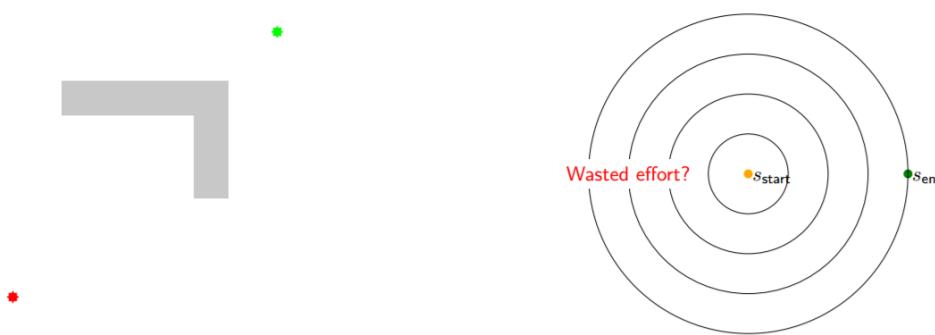
AA 274 | Lecture 5

15

Correctness and improvements

Theorem

If a feasible path exists from q_I to q_G , then algorithm terminates in finite time with $C(q_G)$ equal to the optimal cost of traversal, $C^*(q_G)$.



10/12/22

AA 274 | Lecture 5

16

A*: Improving Dijkstra

Dijkstra

- Dijkstra orders by optimal “cost-to-arrival”
- Faster results if order by “cost-to-arrival”+ (approximate) “cost-to-go”
- That is, strengthen test

$$C(q) + C(q, q') \leq \text{UPPER}$$

to

$$C(q) + C(q, q') + h(q') \leq \text{UPPER}$$

where $h(q)$ is a heuristic for optimal cost-to-go (specifically, a positive *underestimate*)

- In this way, fewer nodes will be placed in the frontier queue
- This modification still guarantees that the algorithm will terminate with a shortest path

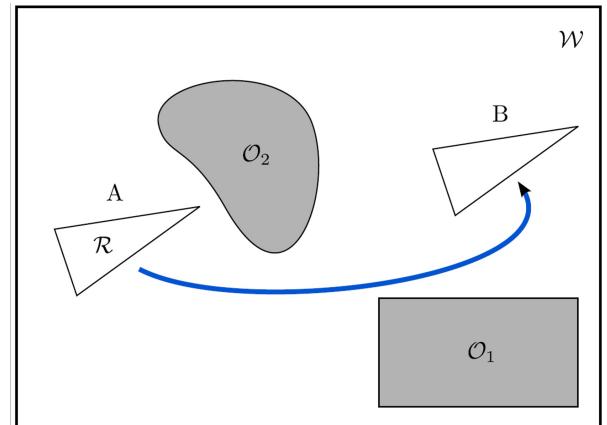
A*

Grid-based approaches: summary

- Pros:
 - Simple and easy to use
 - Fast (for some problems)
- Cons:
 - Resolution dependent
 - Not guaranteed to find solution if grid resolution is not small enough
 - Limited to simple robots
 - Grid size is exponential in the number of DOFs

Back to continuous motion planning

- A robot is a geometric entity operating in continuous space
- *Combinatorial techniques* for motion planning capture the structure of this continuous space
 - Particularly, the regions in which the robot is not in collision with obstacles
- Such approaches are typically complete
 - i.e., guaranteed to find a solution;
 - and sometimes even an optimal one



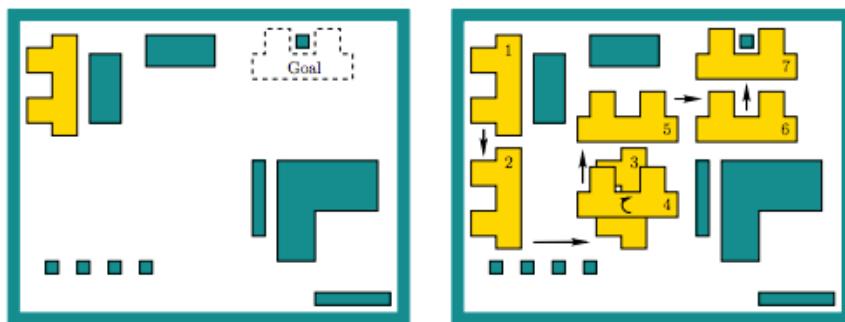
10/12/22

AA 274 | Lecture 5

19

Simplest setup revisited

- Assume 2D workspace: $\mathcal{W} \subseteq \mathbb{R}^2$
- $\mathcal{O} \subset \mathcal{W}$ is the obstacle region with polygonal boundary
- Robot is a rigid polygon
- **Problem:** Given initial placement of robot, compute how to gradually move it into a desired goal placement so that it never touches the obstacle region



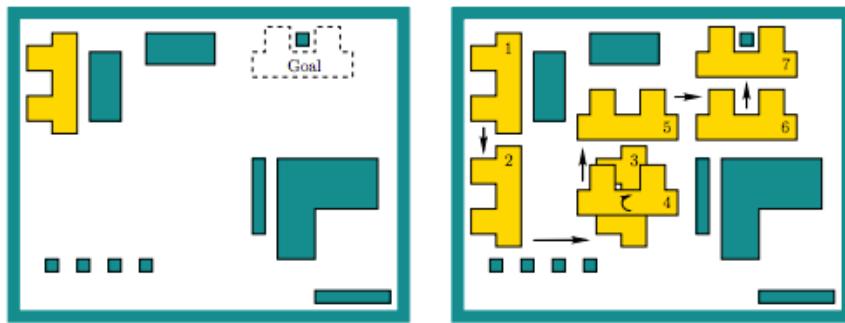
10/12/22

AA 274 | Lecture 5

20

Simplest setup

Key point: motion planning problem described in the real-world, but it really lives in another space -- the **configuration (C-) space**!



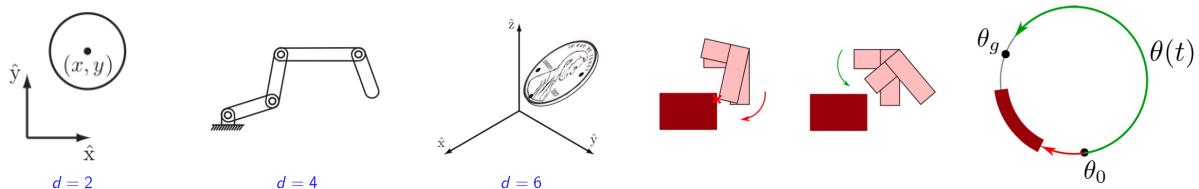
10/12/22

AA 274 | Lecture 5

21

Configuration space

- **C-space:** captures all degrees of freedom (all rigid body transformations)
- More in detail, let $\mathcal{R} \subset \mathbb{R}^2$ be a polygonal robot (e.g., a triangle)
- The robot can rotate by angle θ or translate $(x_t, y_t) \subset \mathbb{R}^2$
- Every combination $q = (x_t, y_t, \theta)$ yields a *unique* robot placement: **configuration**
- So, C-space is a subset of \mathbb{R}^3
- Note: $\theta \pm 2\pi$ yields equivalent rotations \Rightarrow C-space is: $\mathbb{R}^2 \times S^1$
- Concept of C-space extends naturally to higher dimensions (e.g., robot linkages)



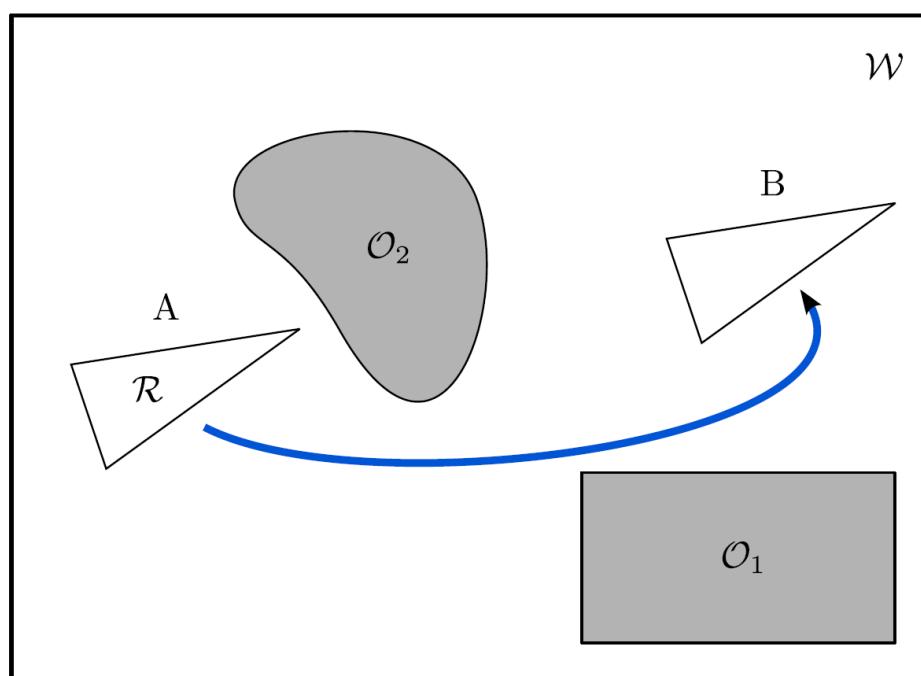
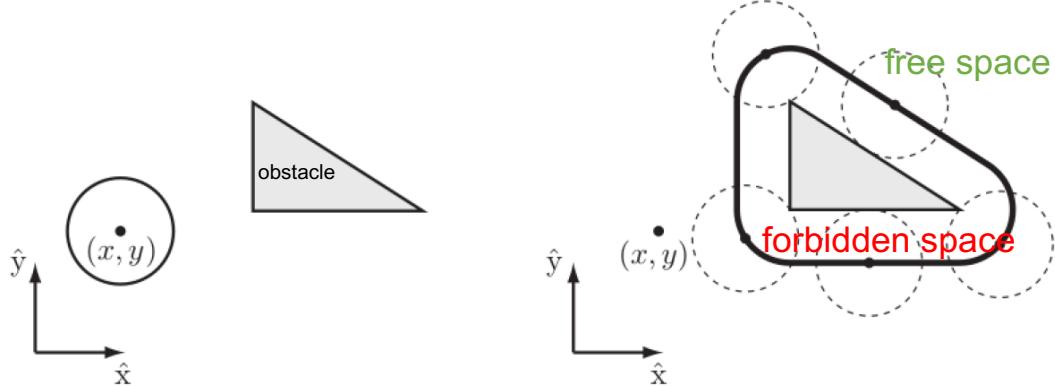
10/12/22

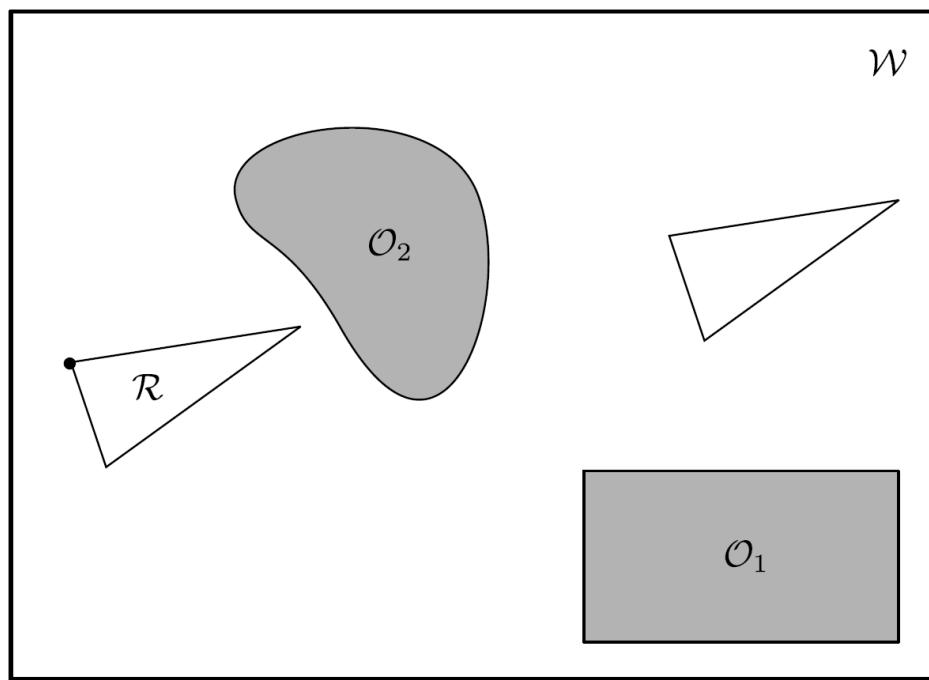
AA 274 | Lecture 5

22

Configuration free space

- The subset $\mathcal{F} \subseteq \mathcal{C}$ of all collision free configurations is the **free space**

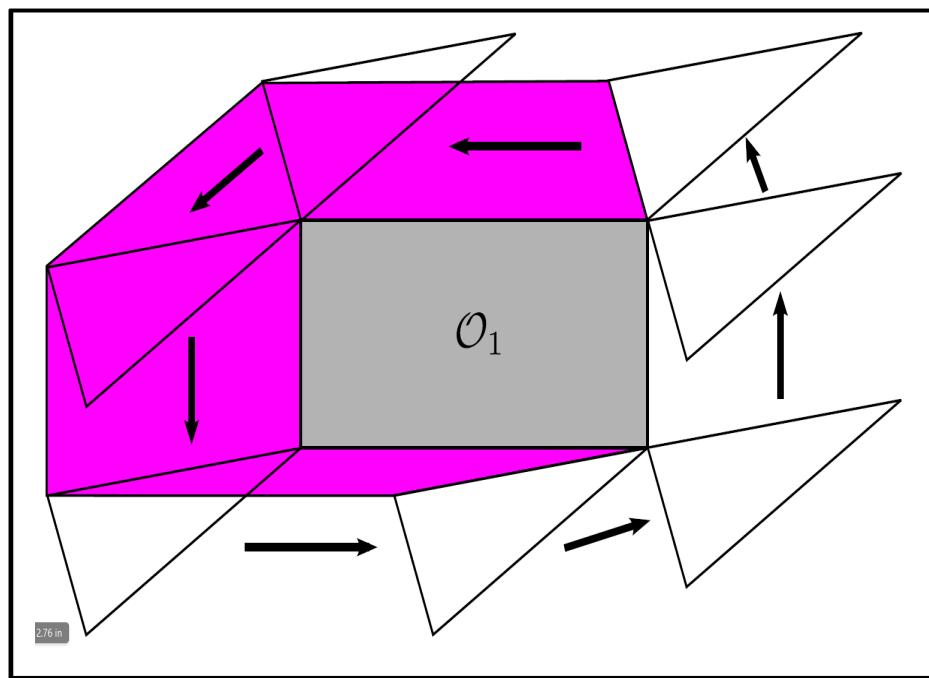




10/13/22

AA 274 | Lecture 5

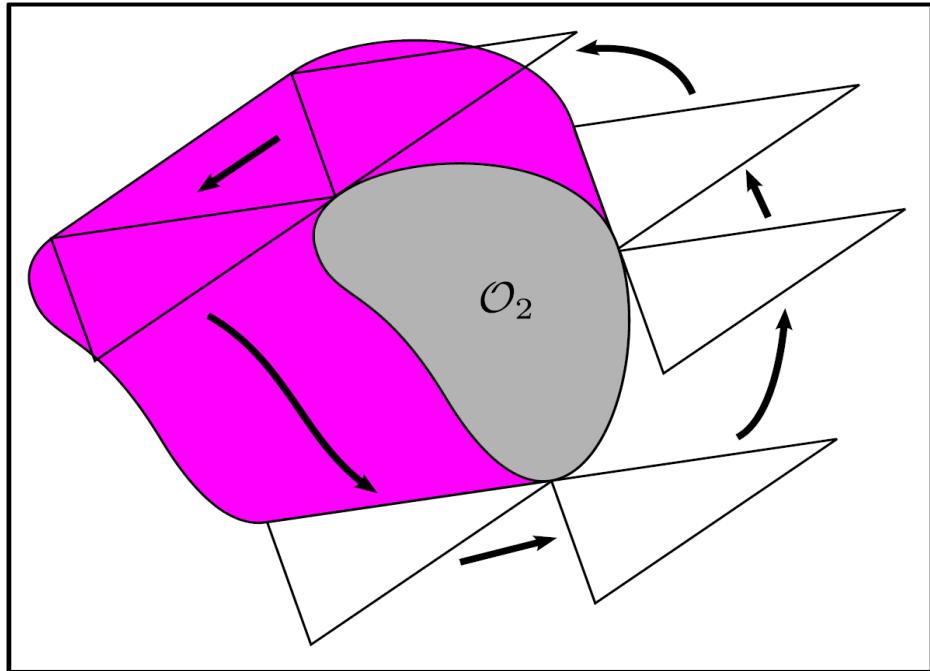
24



10/13/22

AA 274 | Lecture 5

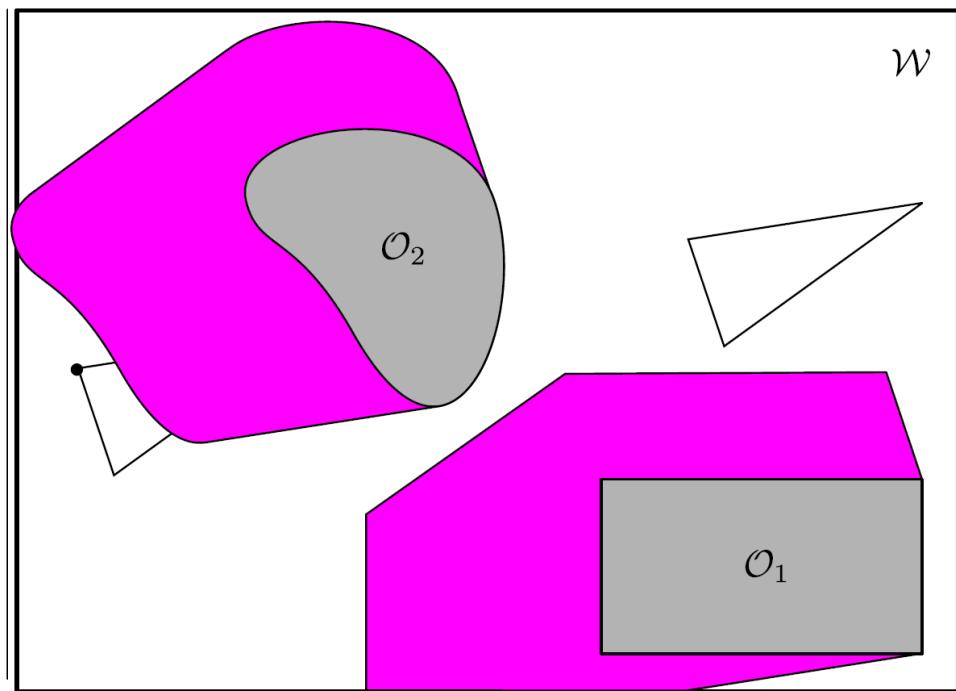
24



10/13/22

AA 274 | Lecture 5

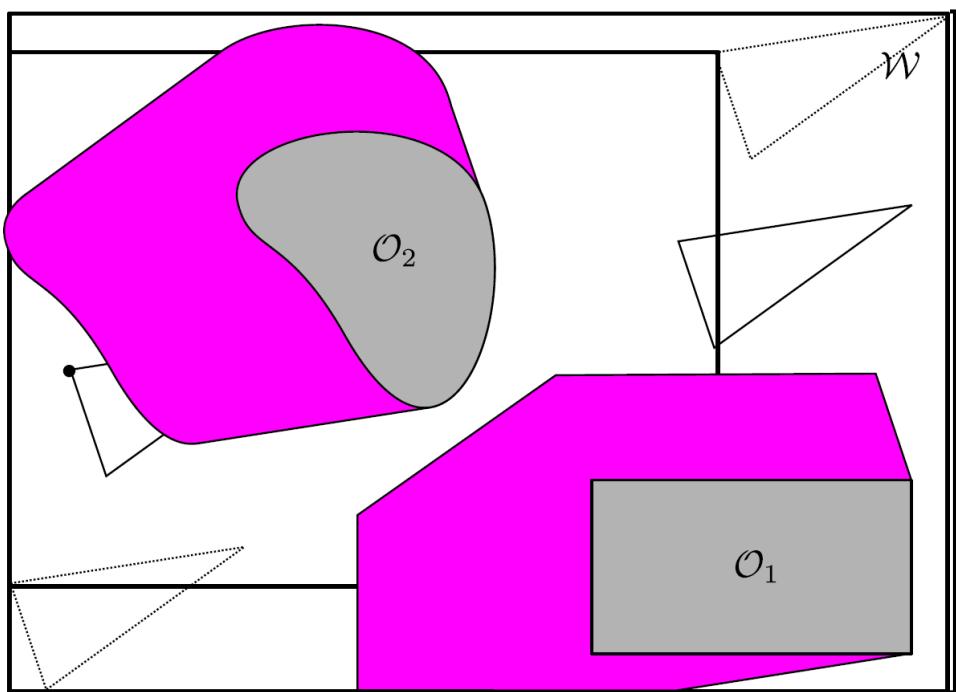
24



10/13/22

AA 274 | Lecture 5

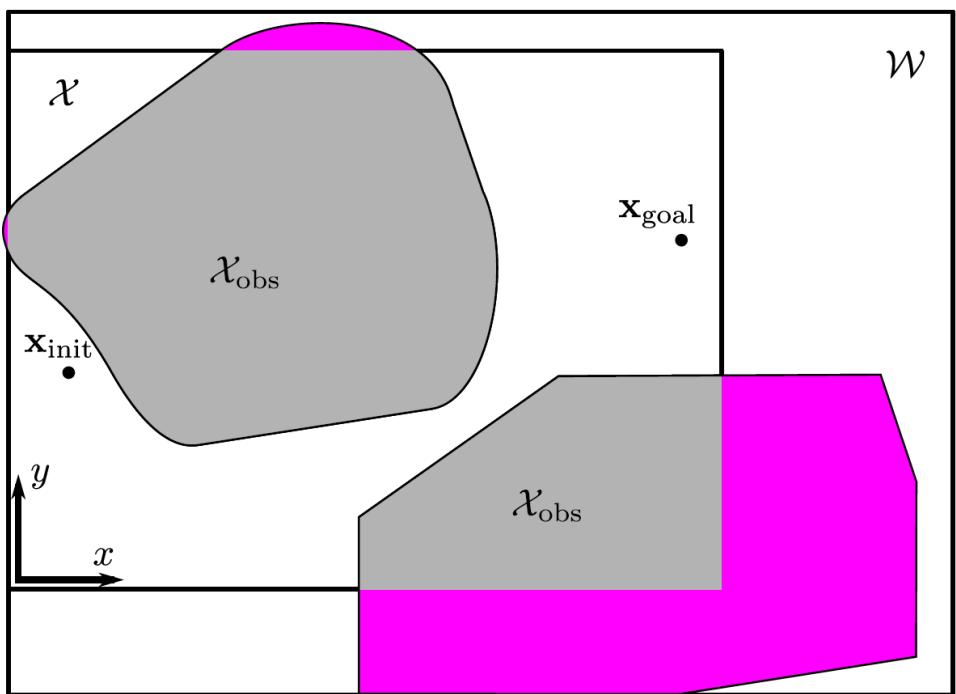
24



10/13/22

AA 274 | Lecture 5

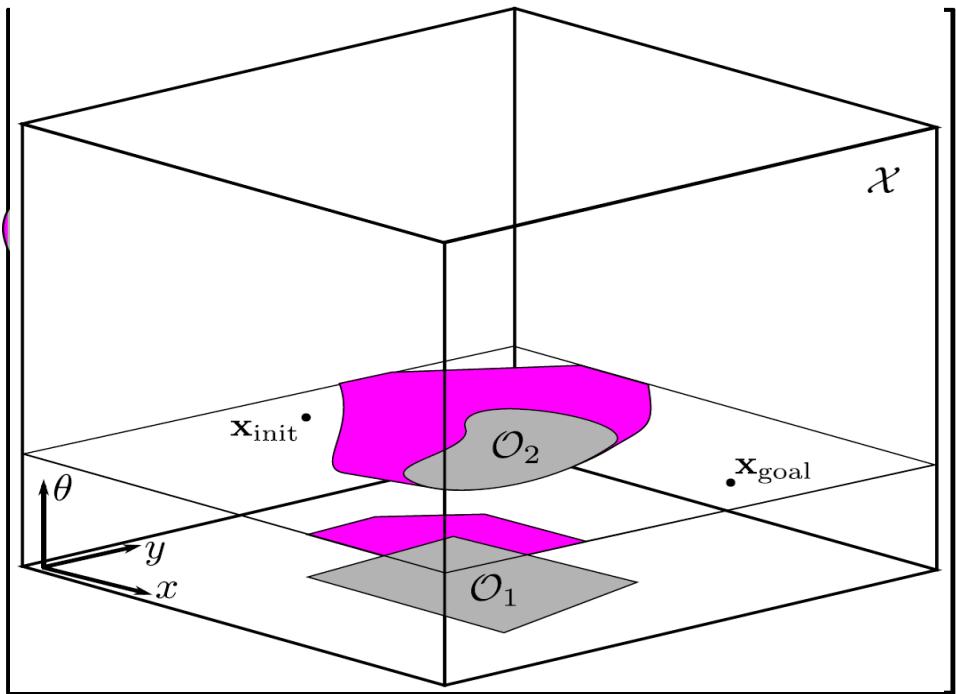
24



10/13/22

AA 274 | Lecture 5

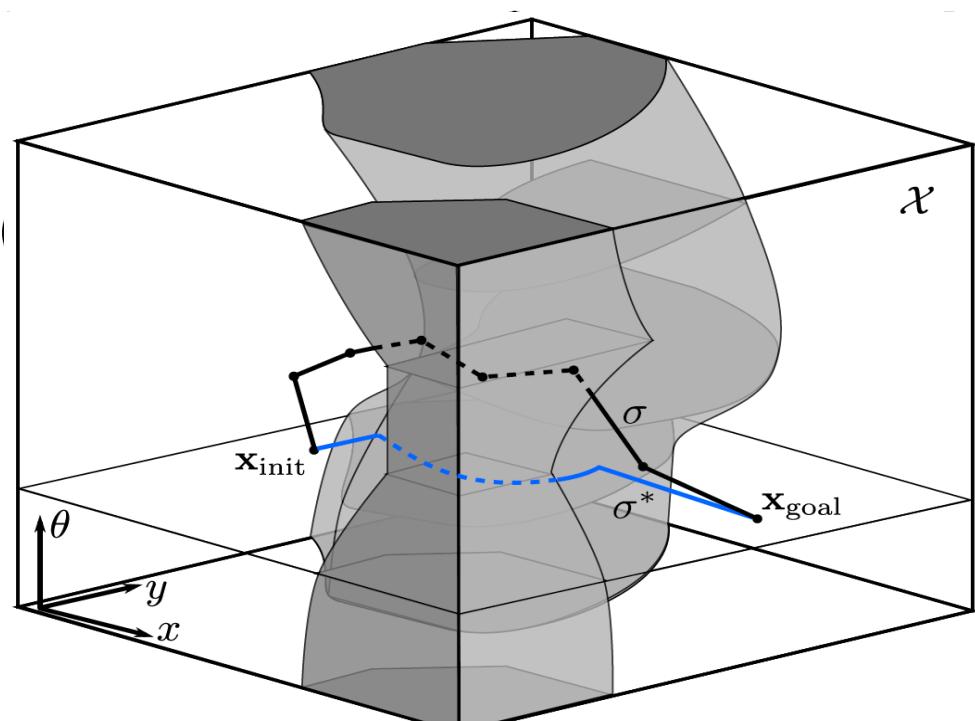
24



10/13/22

AA 274 | Lecture 5

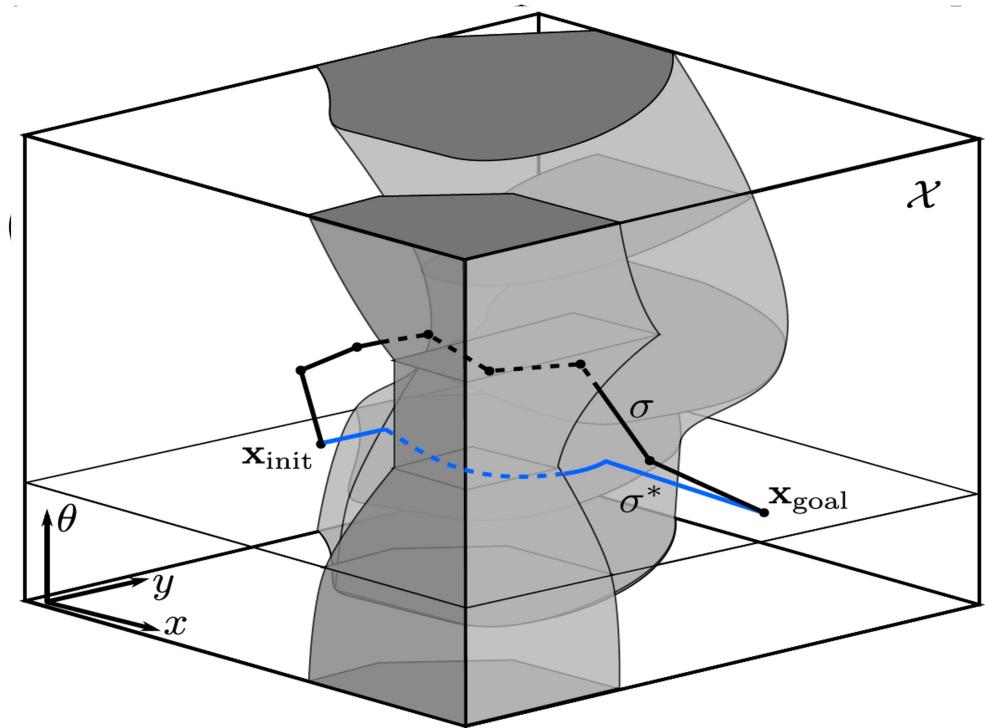
24



10/13/22

AA 274 | Lecture 5

24



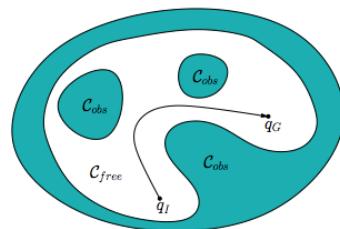
10/12/22

AA 274 | Lecture 5

24

Planning in C-space

- Let $R(q) \subset W$ denote the set of points in the world occupied by the robot when in configuration q
- Robot in collision $\Leftrightarrow R(q) \cap O \neq \emptyset$
- Accordingly, free space is defined as: $C_{free} = \{q \in C \mid R(q) \cap O = \emptyset\}$
- Path planning problem in C-space: compute a **continuous** path: $\tau: [0,1] \rightarrow C_{free}$, with $\tau(0) = q_I$ and $\tau(1) = q_G$



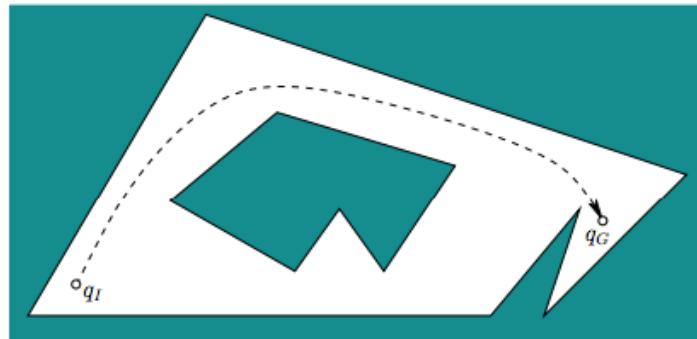
10/12/22

AA 274 | Lecture 5

25

Combinatorial planning

Key idea: compute a roadmap, which is a graph in which each vertex is a configuration in C_{free} and each edge is a path through C_{free} that connects a pair of vertices



Free-space roadmaps

Given a complete representation of the free space, we compute a roadmap that captures its connectivity

A roadmap should preserve:

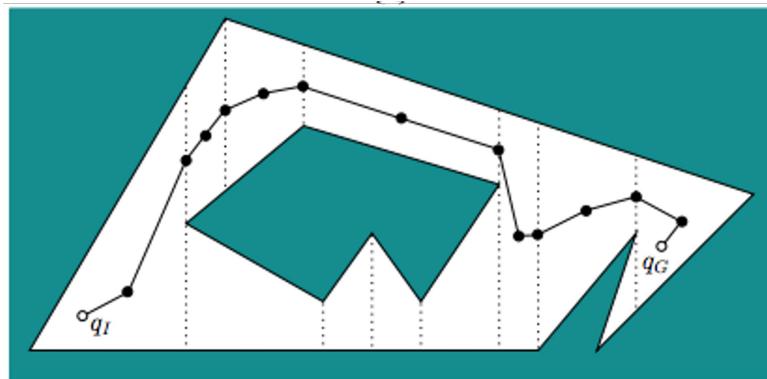
1. **Accessibility:** it is always possible to connect some q to the roadmap (e.g., $q_I \rightarrow s_1, q_G \rightarrow s_2$)
2. **Connectivity:** if there exists a path from q_I to q_G , there exists a path on the roadmap from s_1 to s_2

Main point: a roadmap provides a discrete representation of the continuous motion planning problem *without losing* any of the original connectivity information needed to solve it

Cell decomposition

Typical approach: **cell decomposition**. General requirements:

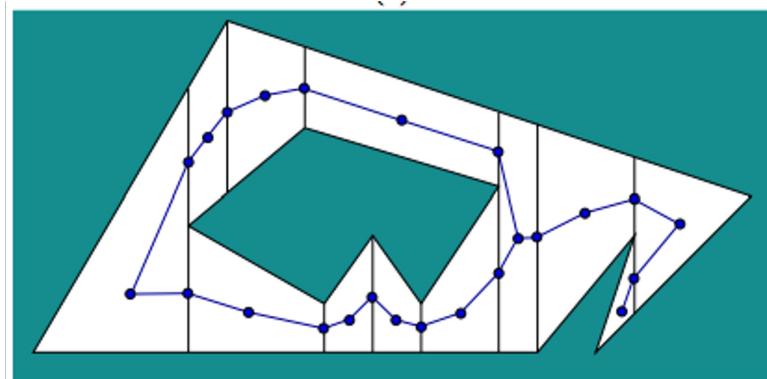
- Decomposition should be easy to compute
- Each cell should be easy to traverse (ideally convex)
- Adjacencies between cells should be straightforward to determine



Computing a trapezoidal cell decomposition

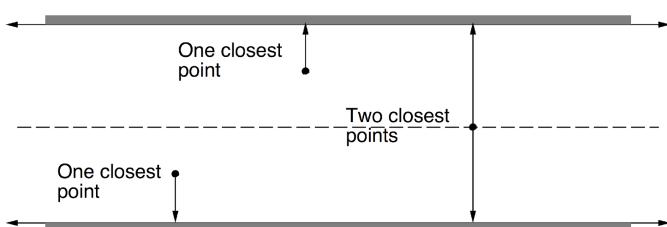
For every vertex (corner) of the forbidden space:

- Extend a vertical ray until it hits the first edge from top and bottom
 - Compute intersection points with all edges, and take the closest ones
 - More efficient approaches exists

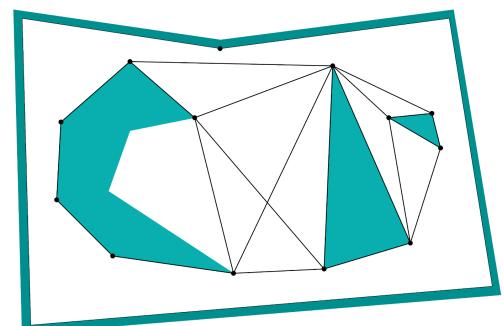


Other roadmaps

Maximum clearance (medial axis)



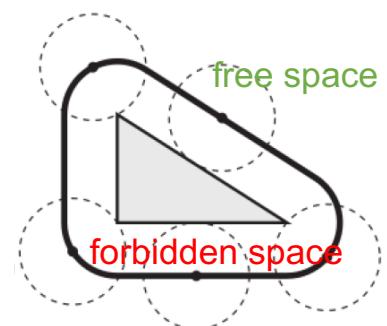
Minimum distance (visibility graph)



Note: No loss in optimality for a proper choice of discretization

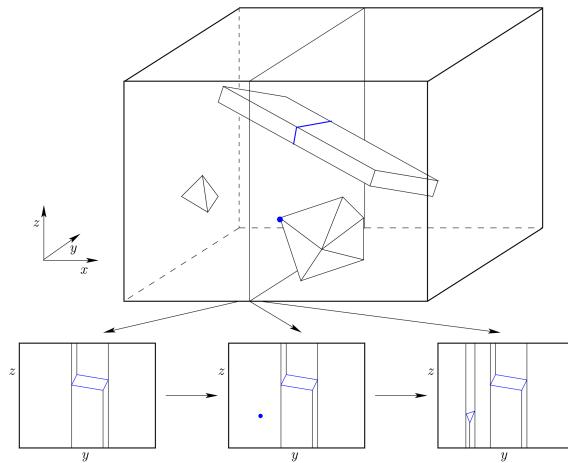
Caveat: free-space computation

- The free space is **not known** in advance
- We need to compute this space given the ingredients
 - Robot representation, i.e., its shape (polygon, polyhedron, ...)
 - Representation of obstacles
- To achieve this, we do the following:
 - Contract the robot into a point
 - In return, inflate (or stretch) obstacles by the shape of the robots



Higher dimensions

- Extensions to higher dimensions is challenging \Rightarrow algebraic decomposition methods



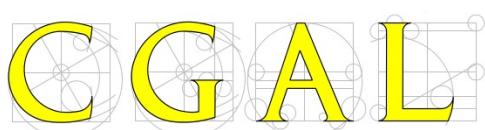
10/12/22

AA 274 | Lecture 5

32

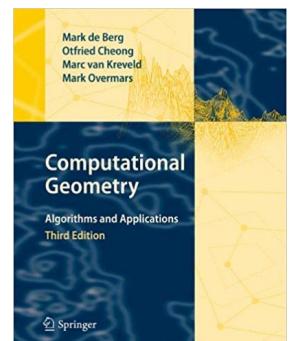
Additional resources on combinatorial planning

- Visualization of C-space for polygonal robot:
<https://www.youtube.com/watch?v=SBFwgR4K1Gk>
- Algorithmic details for Minkowski sums and trapezoidal decomposition: de Berg et al., “Computational geometry: algorithms and applications”, 2008
- Implementation in C++:
Computational Geometry Algorithms Library



10/12/22

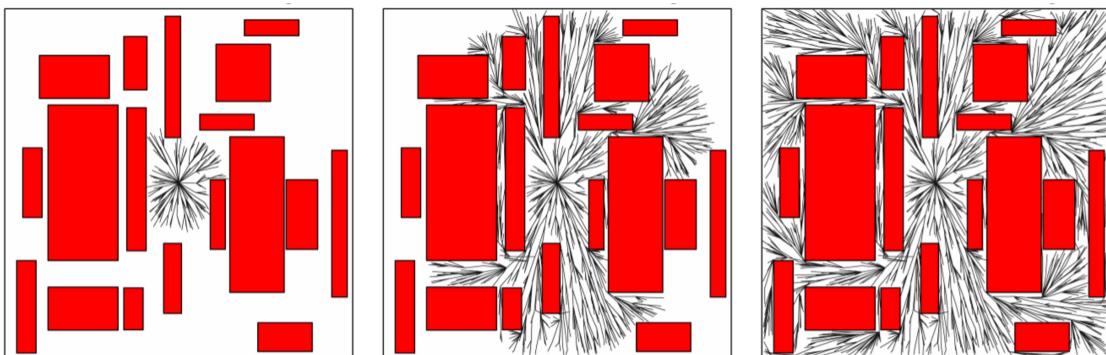
AA 274 | Lecture 5



Combinatorial planning: summary

- These approaches are complete and even optimal in some cases
 - Do not discretize or approximate the problem
- Have theoretical guarantees on the running time
 - I.e., computational complexity is known
- Usually limited to small number of DOFs
 - Computationally intractable for many problems
- Problem specific: each algorithm applies to a specific type of robot/problem
- Difficult to implement; requires special software to reason about geometric data structures (CGAL)

Next time: sampling-based planning



Principles of Robot Autonomy I

Motion planning II: sampling-based methods

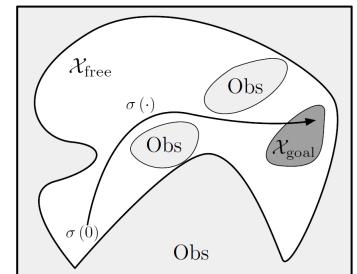


Logistics

- Masks
- Homework 1 is being graded; Honor code cases
- Homework 2 re-released
- Students are being moved off the waitlist
 - If you decided to drop the course, please do so well before drop deadline on Friday 5pm
- Check out the lecture notes!

Motion planning

Compute sequence of actions that drives a robot from an initial condition to a terminal condition while avoiding obstacles, respecting motion constraints, and possibly optimizing a cost function

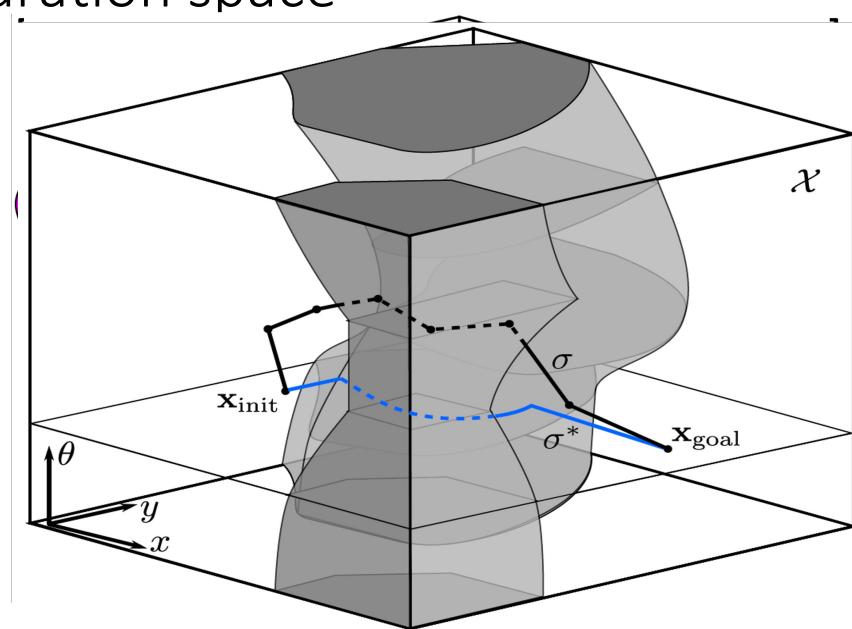


- Aim
 - Learn about sampling-based motion planning algorithms
- Readings:
 - S. LaValle. Planning Algorithms. Chapter 5.

10/17/22

AA274 | Lecture 6

Configuration space



10/17/22

AA274 | Lecture 6

Motion planning algorithms

- **Key point:** motion planning problem described in the real-world, but it really lives in another space - the configuration (C-)space!
- Two main approaches to *continuous* motion planning:
 - *Combinatorial planning:* constructs structures in the C-space that discretely and completely capture all information needed to perform planning
 - *Sampling-based planning:* uses collision detection algorithms to probe and incrementally search the C-space for a solution, rather than completely characterizing all of the C_{free} structure

Sampling-based motion planning

Limitations of combinatorial approaches stimulated the development of sampling-based approaches

- Abandon the idea of explicitly characterizing C_{free} and C_{obs}
- Instead, capture the structure of C by **random sampling**
- Use a black-box component (collision checker) to determine which random configurations lie in C_{free}
- Use such a probing scheme to build a roadmap and then plan a path

Reference: LaValle, S. M. Motion planning. 2011.

Sampling-based motion planning

Pros:

- Conceptually simple
- Relatively-easy to implement
- **Flexible**: one algorithm applies to a variety of robots and problems
- **Beyond the geometric case**: can cope with complex differential constraints, uncertainty, etc.

(Mild) cons:

- Unclear how many samples should be generated to retrieve a solution
- Cannot determine whether a solution does not exist

Outline

- The geometric case
- The kinodynamic case
- Alternative sampling strategies (de-randomized; biased)

Outline

- The geometric case
- The kinodynamic case
- Alternative sampling strategies (de-randomized; biased)

Review of sampling-based methods

Traditionally, two major approaches:

- Probabilistic Roadmap (PRM): graph-based
 - **Multi-query planner**, i.e., designed to solve multiple path queries on the same scenario
 - Original version: [Kavraki et al., '96]
 - “Lazy” version: [Bohlin & Kavraki, '00]
 - Dynamic version: [Jaillet & T. Simeon, '04]
 - Asymptotically optimal version: [Karaman & Frazzoli, '11]
- Rapidly-exploring Random Trees (RRT): tree-based
 - **Single-query** planner
 - Original version: [LaValle & Kuffner, '01]
 - RDT: [LaValle, '06]
 - SRT: [Plaku et al., '05]
 - Asymptotically optimal version [Karaman & Frazzoli, '11]

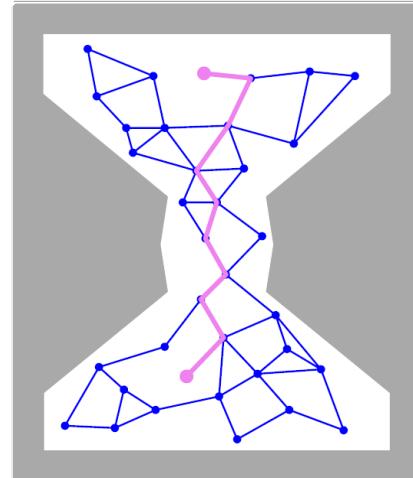
Probabilistic roadmaps (PRM)

A **multi-query** planner, which generates a roadmap (**graph**) G , embedded in the free space

Preprocessing step:

1. Sample a collection of n configurations X_n ; discard configurations leading to collisions
2. Draw an edge between each pair of samples $x, x' \in X_n$ such that $\|x - x'\| \leq r$ and straight-line path between x and x' is collision free

Given a query $s, t \in C_{free}$, connect them to G and find a path on the roadmap



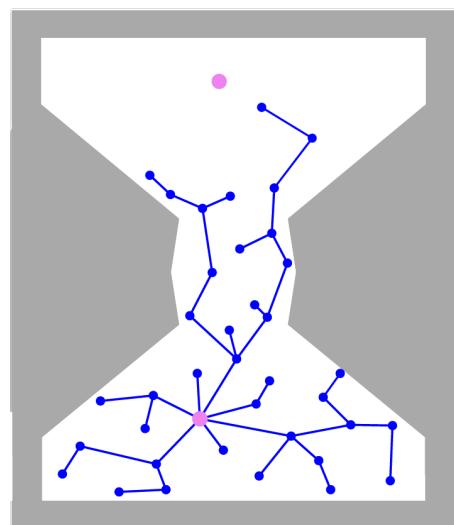
Rapidly-exploring random trees (RRT)

A **single-query** planner, which grows a **tree** T , rooted at the start configuration s , embedded in C_{free}

Algorithm works in n iterations:

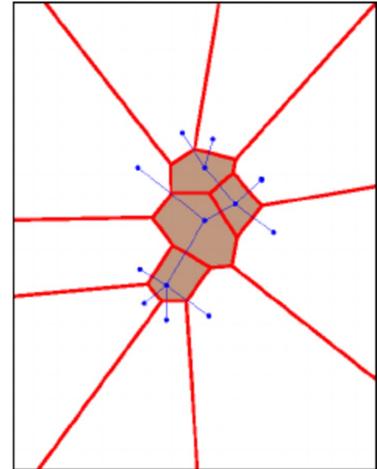
1. Sample configuration x_{rand}
2. Find nearest vertex x_{near} in T to x_{rand}
3. Generate configuration x_{new} in direction of x_{rand} from x_{near} , such that $\overline{x_{near}x_{new}} \subset C_{free}$
4. Update tree: $T = T \cup \{x_{new}, (x_{near}, x_{new})\}$

Occasionally, set x_{rand} to be the target vertex t ; terminate when $x_{new} = T$



Rapidly-exploring random trees (RRT)

- RRT is known to work quite well in practice
- Its performance can be attributed to its **Voronoi bias**:
 - Consider a Voronoi diagram with respect to the vertices of the tree
 - For each vertex, its Voronoi cell consists of all points that are closer to that vertex than to any other
 - Vertices on the frontier of the tree have larger Voronoi cells – hence sampling in those regions is more likely



10/17/22

AA274 | Lecture 6

Theoretical guarantees: probabilistic completeness

Question: how large should the number of samples n be? We can say something about the **asymptotic behavior**:

Kavraki et al. 96: PRM, with $r = \text{const}$, will eventually (as $n \rightarrow \infty$) find a solution if one exists

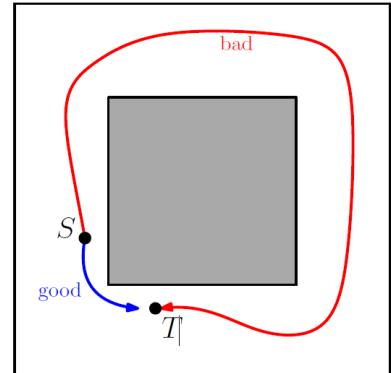
LaValle, 98; Kleinbort et al., 18: RRT will eventually (as $n \rightarrow \infty$) find a solution if one exists

* Unless stated otherwise, the configuration space is assumed to be the d -dimensional Euclidean unit hypercube $[0,1]^d$, with $2 \leq d \leq \infty$

Theoretical guarantees: quality

Question: what can be said about the **quality** of the returned solution for PRM and RRT, in terms of length, energy, etc.?

Nechushtan et al. (2011) and Karaman and Frazzoli (2011) proved that RRT can produce arbitrarily-bad paths with non-negligible probability: for example, RRT would prefer to take the long (red) way



10/17/22

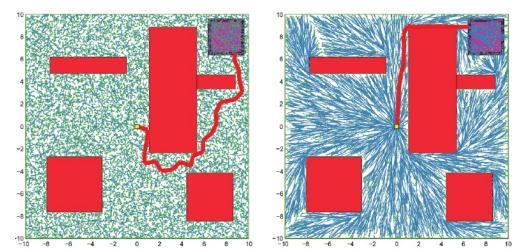
AA274 | Lecture 6

Theoretical guarantees: quality

Karaman and Frazzoli in 2011 provided the first rigorous study of optimality in sampling-based planners:

Theorem: The **cost** of the solution returned by PRM converges, as $n \rightarrow \infty$, to the optimum, when $r_n = \gamma \left(\frac{\log n}{n} \right)^{\frac{1}{d}}$, where γ only depends on d

- KF11 also introduced an asymptotically optimal variant of RRT called RRT* (right)
- Result was later updated to [Solovey et al. 2019]:
$$r_n = \gamma \left(\frac{\log n}{n} \right)^{\frac{1}{d+1}}$$
- Now back to **1/d** [preprint, Lukyanenko et al. 2021]



10/17/22

AA274 | Lecture 6

Observations

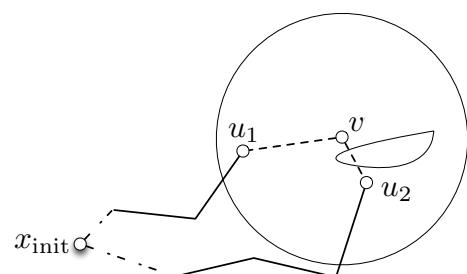
- PRM-like motion planning algorithms
 - For a give number of nodes n , they find “good” paths
 - ...however, require many costly collision checks
- RRT-like motion planning algorithms
 - Finds a feasible path quickly
 - ...however, the quality of that path is, in general, poor
 - “traps” itself by disallowing new better paths to emerge - RRT* performs local label correction as samples are added to help remedy this

10/17/22

AA274 | Lecture 6

Fast Marching Tree Algorithm (FMT*)

- **Key idea:** run dynamic programming (DP) on sampled nodes, skipping any step in which the attempted connection causes a collision
 - lazy DP operator:
$$c(v) = \min_{u: \|u-v\| < r_n} \text{Cost}(u, v) + c(u)$$
- Laziness introduces “suboptimal” connections, but such connections are vanishingly rare and FMT* is **asymptotically optimal**
- Ratio of # of collision-checks for FMT* versus PRM* goes to zero!



Reference: Janson et al. Fast Marching Tree: A Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions. 2015

10/17/22

AA274 | Lecture 6

Sampling-based planning: summary

- Sampling-based planners transform the difficult global problem into a large set of **local** and **easy** problems
- A key ingredient is **collision detection**, which is conceptually easy, as it can be solved in the workspace (2D or 3D)
- **Local planning** (edge validation) is typically performed by dense sampling of path and collision detection
- Another key ingredient is nearest-neighbor search: given a query point find its nearest neighbor(s) within a set of points -- also well studied theoretically and practically

Outline

- The geometric case
- The kinodynamic case
- Alternative sampling strategies (de-randomized; biased)

Flexible trajectories in kinodynamic motion planning

- https://github.com/rikkimelissa/baxter_throw
- Rikki Valverde, Northwestern University



10/17/22

AA274 | Lecture 6

Kinodynamic motion planning for vehicle high speed maneuvering

- Yanbo Li



10/17/22

AA274 | Lecture 6

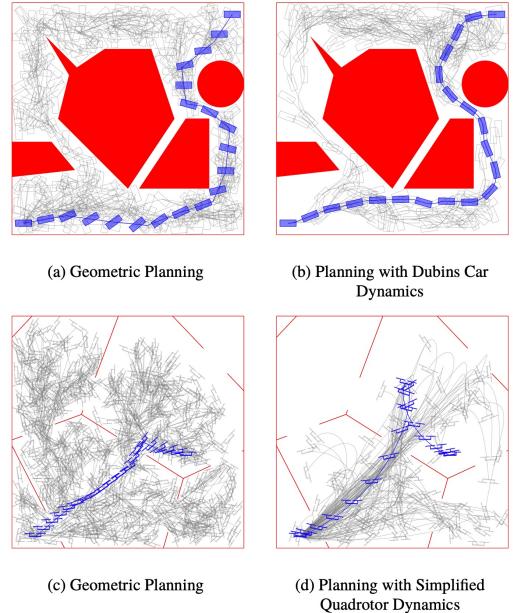
Kinodynamic planning

Kinodynamic motion planning problem:
in addition to obstacle avoidance, paths
are subject to differential constraints

- The robot operates in the state space X
- To move the robot applies control $u \in U$
- Motion needs to satisfy the system's constraints:

$$\dot{x} = f(x, u) \text{ for } x \in X, u \in U$$

Reference: Schmerling and Pavone. Kinodynamic Planning.
2019



10/17/22

AA274 | Lecture 6

Forward-propagation-based algorithms

RRT can be extended to the kinodynamic case in a relatively easy way:

1. Draw a random state and find its nearest neighbor x_{near}
2. Sample a random control $u \in U$ and random duration t
3. **Forward propagate** the control u for t time from x_{near}

```

1:  $\mathcal{T}.\text{init}(x_{\text{init}})$ 
2: for  $i = 1$  to  $k$  do
3:    $x_{\text{rand}} \leftarrow \text{RANDOM\_STATE}()$ 
4:    $x_{\text{near}} \leftarrow \text{NEAREST\_NEIGHBOR}(x_{\text{rand}}, \mathcal{T})$ 
5:    $t \leftarrow \text{SAMPLE\_DURATION}(0, T_{\text{prop}})$ 
6:    $u \leftarrow \text{SAMPLE\_CONTROL\_INPUT}(\mathbb{U})$ 
7:    $x_{\text{new}} \leftarrow \text{PROPAGATE}(x_{\text{near}}, u, t)$ 
8:   if  $\text{COLLISION\_FREE}(x_{\text{near}}, x_{\text{new}})$  then
9:      $\mathcal{T}.\text{add\_vertex}(x_{\text{new}})$ 
10:     $\mathcal{T}.\text{add\_edge}(x_{\text{near}}, x_{\text{new}})$ 
11: return  $\mathcal{T}$ 

```

Reference: Kleinbort et al. Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation. 2018.

10/17/22

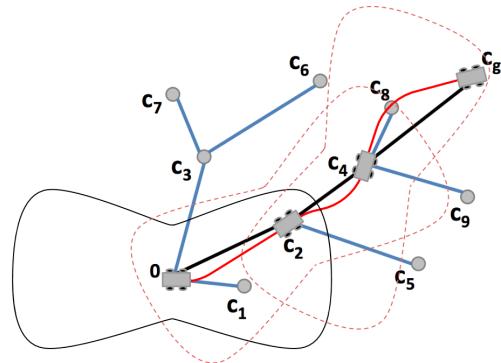
AA274 | Lecture 6

Steering-based algorithms

When efficient online *steering* subroutines exist, kinodynamic planning algorithms may take advantage of this domain knowledge

1. Connect samples by using an optimal trajectory (steering problem)
2. Use reachable sets to find nearest neighbors

Reference: E. Schmerling et al. Optimal Sampling-Based Motion Planning under Differential Constraints: the Driftless Case. 2015



Outline

- The geometric case
- The kinodynamic case
- Alternative sampling strategies (de-randomized; biased)

Should probabilistic planners be probabilistic?

Key question: would theoretical guarantees and practical performance still hold if these algorithms were to be de-randomized, i.e., run on deterministic samples?

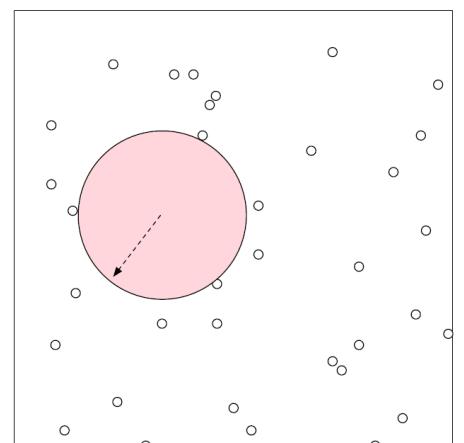
Important question as de-randomization would:

- Ease certification process
- Ease use of offline computation
- Potentially simplify several operations (e.g., NN search)

Designing “good” sequences

ℓ_2 -dispersion: For a finite set S of points contained in $X \subset \mathbb{R}^d$, its ℓ_2 -dispersion $D(S)$ is defined as

$$D(S) := \sup_{x \in X} \min_{s \in S} \|s - x\|_2$$

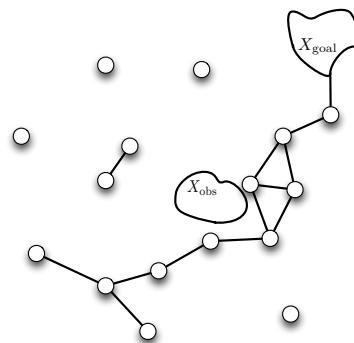


Key facts:

- There exist deterministic sequences with $D(S)$ of order $O(n^{-1/d})$, referred to as **low-dispersion** sequences
- Sequences minimizing ℓ_2 -dispersion only known for $d = 2$

Optimality of deterministic planning

```
1  $V \leftarrow \{x_{\text{init}}\} \cup \text{SampleFree}(n); E \leftarrow \emptyset$ 
2 for all  $v \in V$  do
3    $X_{\text{near}} \leftarrow \text{Near}(V \setminus \{v\}, v, r_n)$ 
4   for  $x \in X_{\text{near}}$  do
5     if CollisionFree( $v, x$ ) then
6        $E \leftarrow E \cup \{(v, x)\} \cup \{(x, v)\}$ 
7     end if
8   end for
9 end for
10 return ShortestPath( $x_{\text{init}}, V, E$ )
```



Optimality: Let c_n denote the arc length of the path returned with n samples.
Then if

1. Samples set S has dispersion $D(S) \leq \gamma n^{-1/d}$ for some $\gamma > 0$,
 2. $n^{1/d} r_n \rightarrow \infty$,
- then $\lim_{n \rightarrow \infty} c_n = c^*$, where c^* is the cost of an optimal path

10/17/22

AA274 | Lecture 6

Deterministic sampling-based motion planning

- Asymptotic optimality can be achieved with **deterministic sequences** and with a **smaller connection radius**
- Deterministic convergence rates: instrumental to the certification of sampling-based planners
- Computational and space complexity: under some assumptions, arbitrarily close to theoretical lower bound
- Deterministic sequences appear to provide superior performance

Reference: Janson et al. Deterministic Sampling-Based Motion Planning: Optimality, Complexity, and Performance. 2018

10/17/22

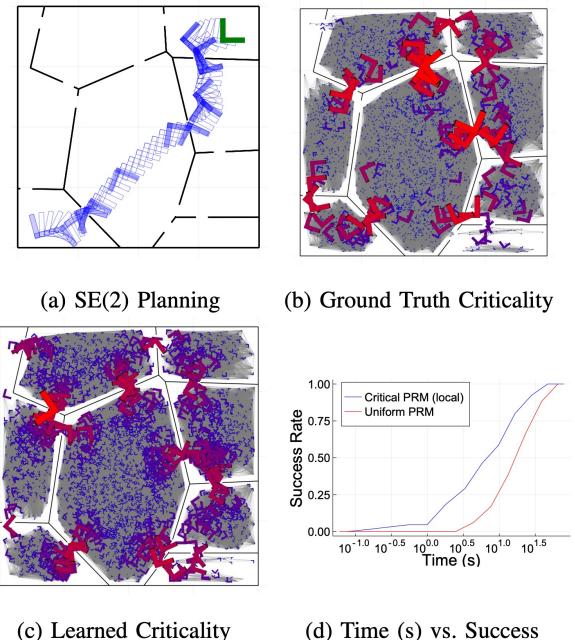
AA274 | Lecture 6

Biased sampling for SBMP

- Potential issue with uniform sampling: narrow corridors in C-space require many samples to identify/traverse
- Key idea: **bias sampling** towards suspected such challenging regions of C-space
- Biased sampling distributions can be hand-constructed and/or adapt online (e.g., Hybrid Sampling PRM), or learned from prior experience solving similar planning problems

References:

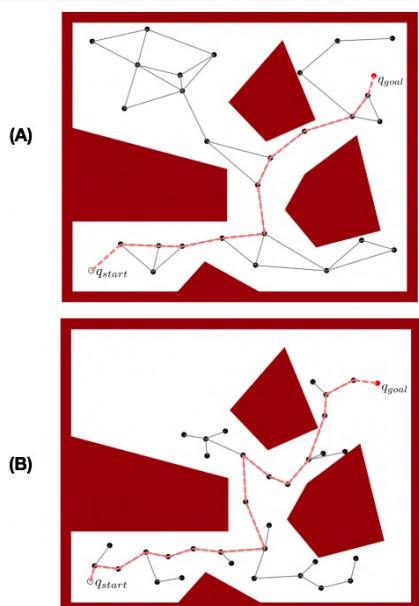
Hsu et al. Hybrid PRM sampling with a cost-sensitive adaptive strategy. 2005.
Ichter et al. Learned Critical Probabilistic Roadmaps for Robotic Motion Planning. 2020



10/17/22

AA274 | Lecture 6

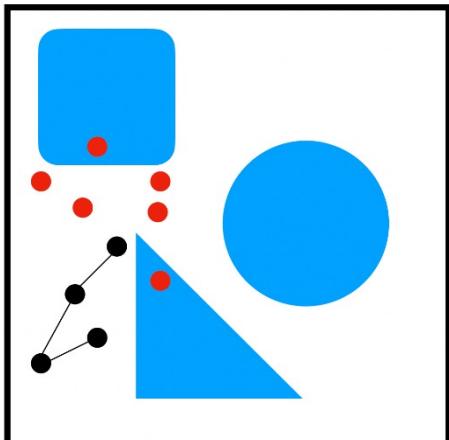
Which of these two roadmaps has been created by RRT?



(A)

(B)

How many of the red sample points could lead to a new node in the roadmap constructed by RRT?



- New sample
- Node in Roadmap

0
1
2
3
4
5
6

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

Next time: robotic sensors and introduction to computer vision



Principles of Robot Autonomy I

Robotic sensors and introduction to computer vision



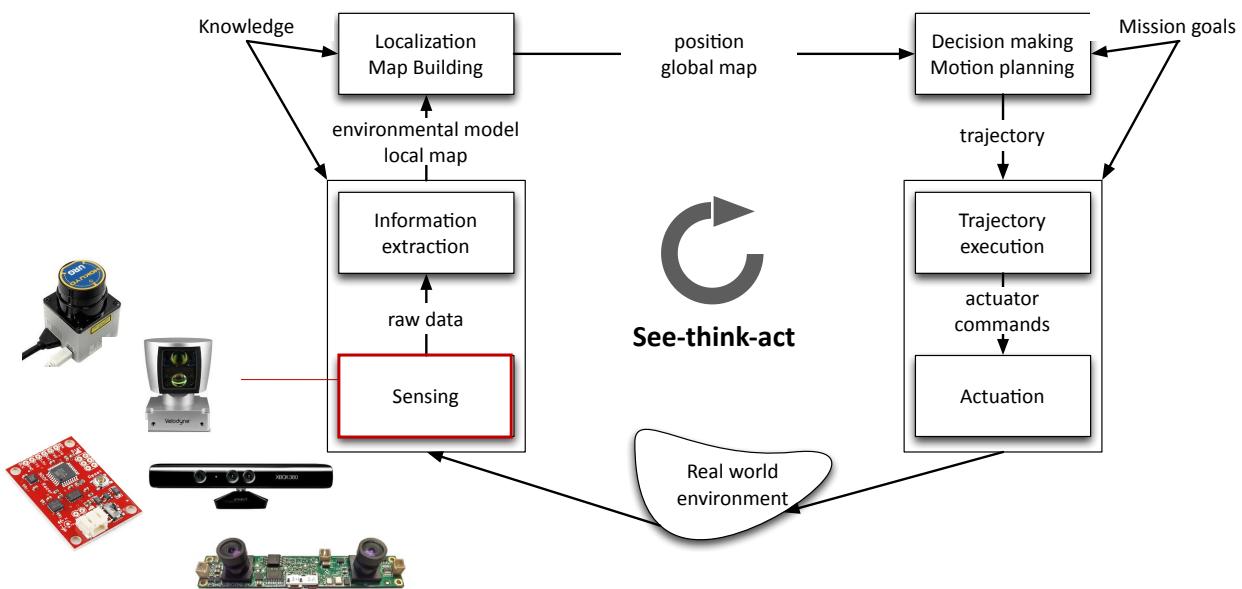
Stanford
University



Logistics

- Masks recommended
- Finalizing Grading of Homework 1
- Start early on Homework 2
- Waitlist finalized
- Check out the lecture notes for worked out examples!

Sensors for mobile robots



10/18/22

AA 274 | Lecture 7

3

Sensors for mobile robots

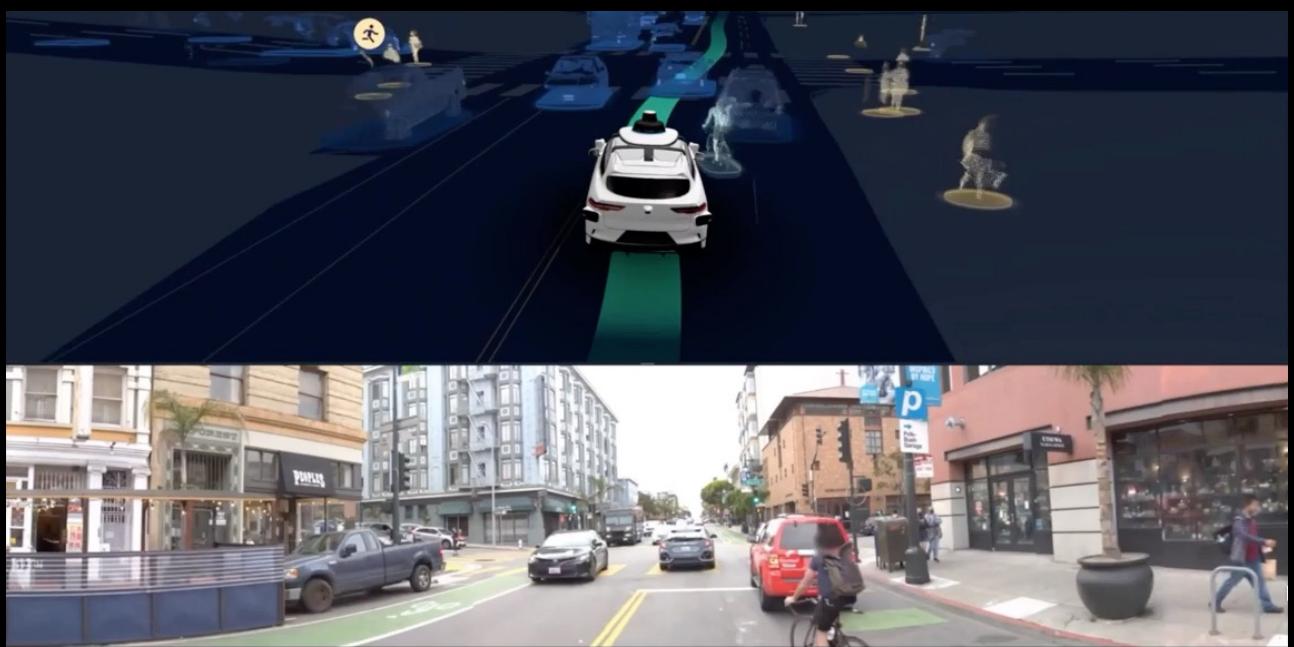
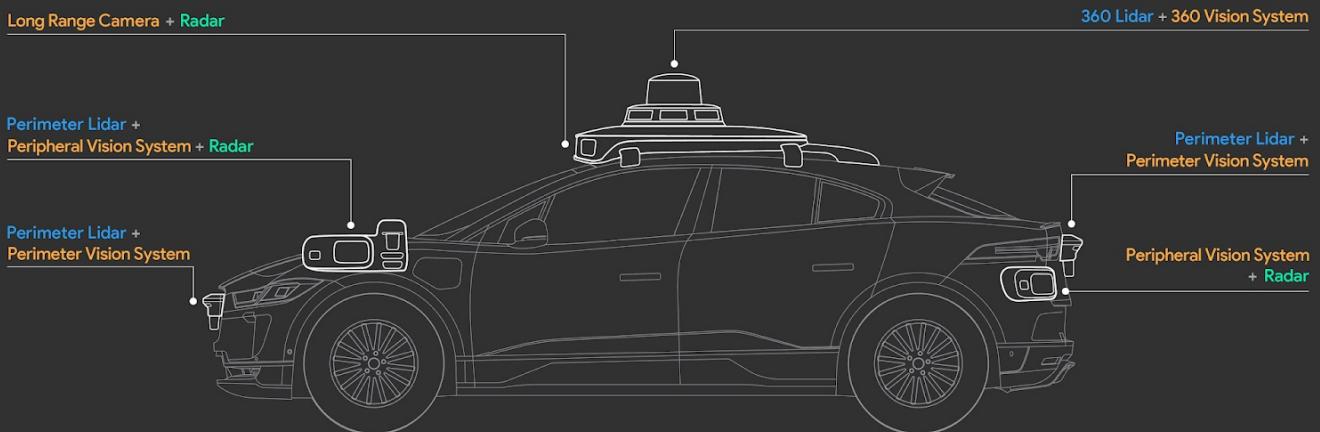
- Aim
 - Learn about key performance characteristics for robotic sensors
 - Learn about a full spectrum of sensors, e.g. proprioceptive / exteroceptive, passive / active
- Readings
 - Siegwart, Nourbakhsh, Scaramuzza. Introduction to Autonomous Mobile Robots. Section 4.1.

10/18/22

AA 274 | Lecture 7

4

Example: self-driving cars



Classification of sensors

- **Proprioceptive**: measure values internal to the robot
 - E.g.: motor speed, robot arm joint angles, and battery voltage
- **Exteroceptive**: acquire information from the robot's environment
 - E.g.: distance measurements and light intensity
- **Passive**: measure ambient environmental energy entering the sensor
 - Challenge: performance heavily depends on the environment
 - E.g.: temperature probes and cameras
- **Active**: emit energy into the environment and measure the reaction
 - Challenge: might affect the environment
 - E.g.: ultrasonic sensors and laser rangefinders

Sensor performance: design specs

- **Dynamic range**: ratio between the maximum and minimum input values (for normal sensor operation)
- **Resolution**: minimum difference between two values that can be detected by a sensor
- **Linearity**: whether the sensor's output response depends linearly on the input
- **Bandwidth or frequency**: speed at which a sensor provides readings (in Hertz)

Sensor performance: in situ specs

- **Sensitivity**: ratio of output change to input change
- **Cross-sensitivity**: sensitivity to quantities that are unrelated to the target quantity
- **Error**: difference between the sensor output m and the true value ν
$$\text{error} := m - \nu$$
- **Accuracy**: degree of conformity between the sensor's measurement and the true value
$$\text{accuracy} := 1 - |\text{error}|/\nu$$
- **Precision**: reproducibility of the sensor results

Sensor errors

- **Systematic errors**: caused by factors that can in theory be modeled; they are deterministic
 - E.g.: calibration errors
- **Random errors**: cannot be predicted with sophisticated models; they are stochastic
 - E.g.: spurious range-finding errors
- **Error analysis**: performed via a probabilistic analysis
 - Common assumption: symmetric, unimodal (and often Gaussian) distributions; convenient, but often a coarse simplification
 - Error propagation characterized by the *error propagation law*

An ecosystem of sensors

- Encoders
- Heading sensors
- Accelerometers and IMU
- Beacons
- Active ranging
- Cameras

10/18/22

AA 274 | Lecture 7

11

Encoders

- **Encoder**: an electro-mechanical device that converts motion into a sequence of digital pulses, which can be converted to **relative** or **absolute** position measurements
 - proprioceptive sensor
 - can be used for robot localization
- **Fundamental principle of optical encoders**: use a light shining onto a photodiode through slits in a metal or glass disc



Wheel encoder
Credit: Pololu



Credit: Honest Sensor

10/18/22

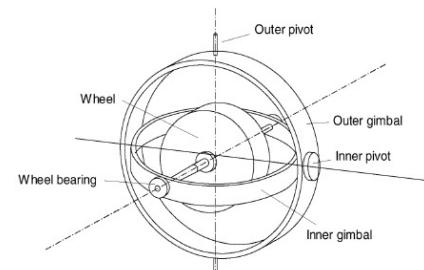
AA 274 | Lecture 7

12

Heading sensors

- Used to determine robot's orientation, it can be:
 1. Proprioceptive, e.g., **gyroscope** (heading sensor that preserves its orientation in relation to a fixed reference frame)
 2. Exteroceptive, e.g., **compass** (shows direction relative to the geographic cardinal directions)
- Fusing measurements with velocity information, one can obtain a position estimate (via integration) -> *dead reckoning*

- **Fundamental principle of mechanical gyroscopes:** angular momentum associated with spinning wheel keeps the axis of rotation inertially stable



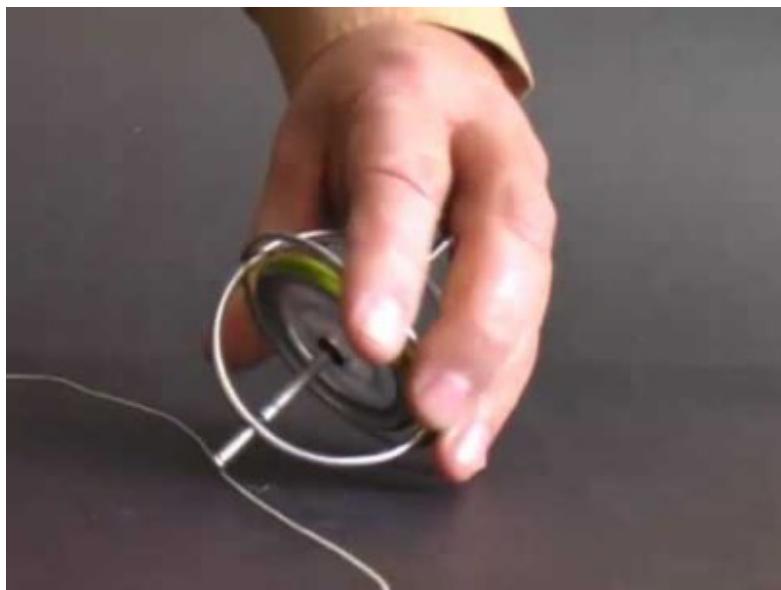
Credit: SNS

10/18/22

AA 274 | Lecture 7

13

Gyroscope example



10/18/22

AA 274 | Lecture 7

14

Accelerometer and IMU

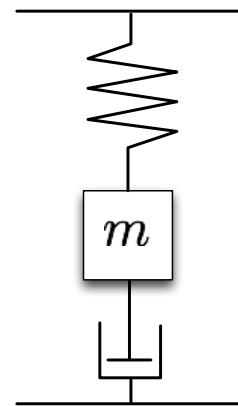
- **Accelerometer**: device that measures all external forces acting upon it
- Mechanical accelerometer: essentially, a spring-mass-damper system

$$F_{\text{applied}} = m\ddot{x} + c\dot{x} + kx$$

with m mass of proof mass, c damping coefficient, k spring constant; in steady state

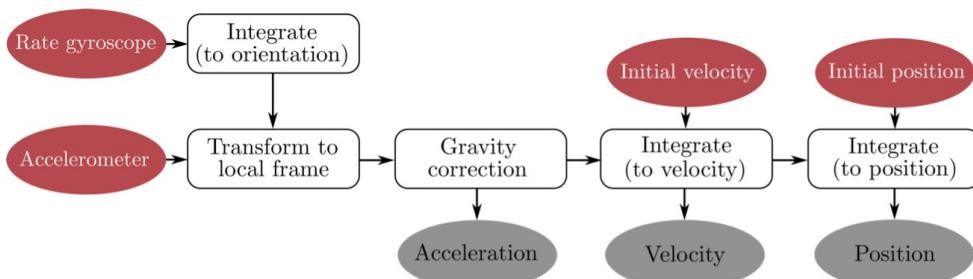
$$a_{\text{applied}} = \frac{kx}{m}$$

- Modern accelerometers use MEMS (cantilevered beam + proof mass); deflection measured via *capacitive* or *piezoelectric* effects



Inertial Measurement Unit (IMU)

- **Definition**: device that uses gyroscopes and accelerometers to estimate the relative position, orientation, velocity, and acceleration of a moving vehicle with respect to an inertial frame
- *Drift* is a fundamental problem: to cancel drift, periodic references to external measurements are required



Beacons

- **Definition:** signaling devices with precisely known positions
- Early examples: stars, lighthouses
- Modern examples: GPS, motion capture systems



10/18/22

AA 274 | Lecture 7

17

Active ranging

- Provide direct measurements of distance to objects in vicinity
- Key elements for both localization and environment reconstruction
- Main types:
 1. Time-of-flight active ranging sensors (e.g., ultrasonic and laser rangefinder)



Credit:
<https://electrosome.com/hc-sr04-ultrasonic-sensor-pic/>



2. Geometric active ranging sensors (optical triangulation and structured light)

10/18/22

AA 274 | Lecture 7

18

Time-of-flight active ranging

- **Fundamental principle:** time-of-flight ranging makes use of the propagation of the speed of sound or of an electromagnetic wave
- Travel distance is given by

$$d = ct$$

where d is the distance traveled, c is the speed of the wave propagation, and t is the time of flight

- Propagation speeds:
 - Sound: 0.3 m/ms
 - Light: 0.3 m/ns
- Performance depends on several factors, e.g., uncertainties in determining the exact time of arrival and interaction with the target

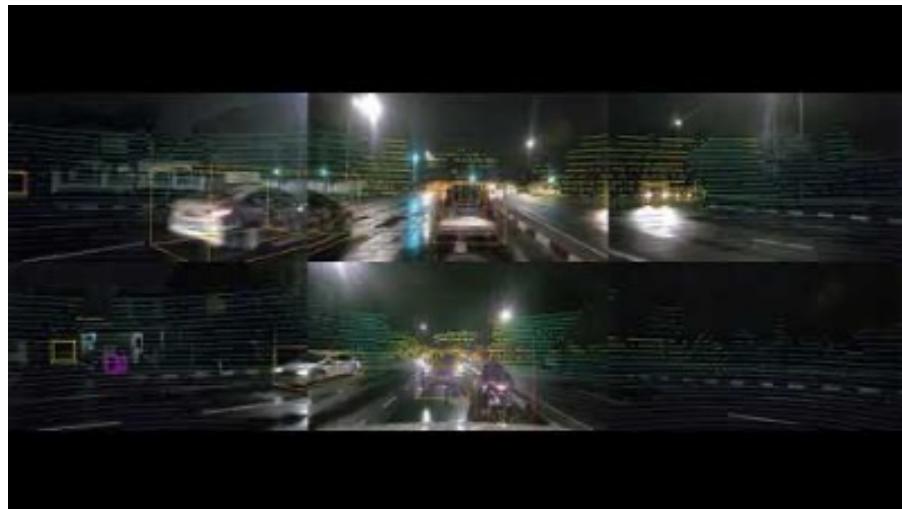
Example Lidar data from nuScenes dataset

<https://www.nuscenes.org/>



Lidar point clouds overlayed with Camera images

<https://www.nuscenes.org/>



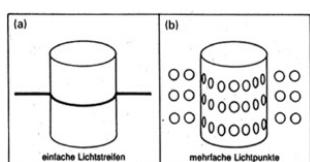
10/18/22

AA 274 | Lecture 7

21

Geometric active ranging

- **Fundamental principle:** use geometric properties in the measurements to establish distance readings
- The sensor projects a known light pattern (e.g., point, line, or texture); the reflection is captured by a receiver and, together with known geometric values, range is estimated via triangulation
- Examples:
 - Optical triangulation (1D sensor)
 - Structured light (2D and 3D sensor)

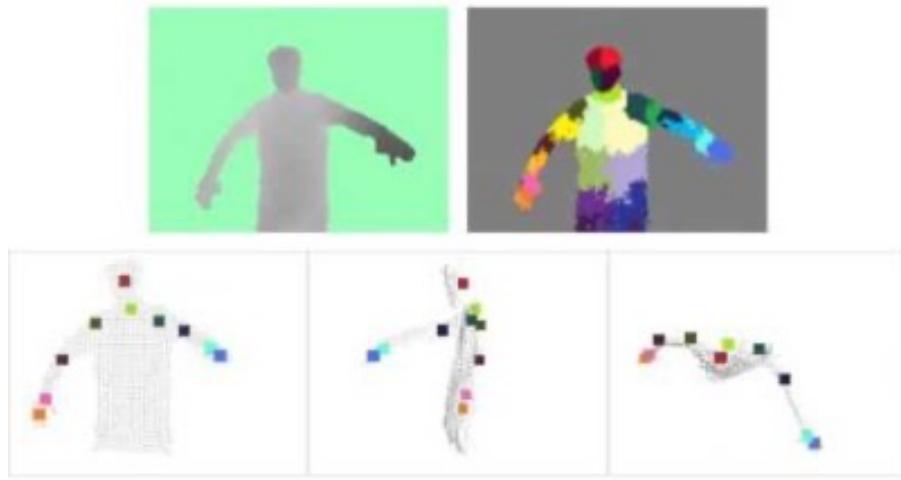


10/18/22

AA 274 | Lecture 7

22

Real-Time Human Pose Recognition in Parts from Single Depth Images. Shotton et al. CVPR 2011



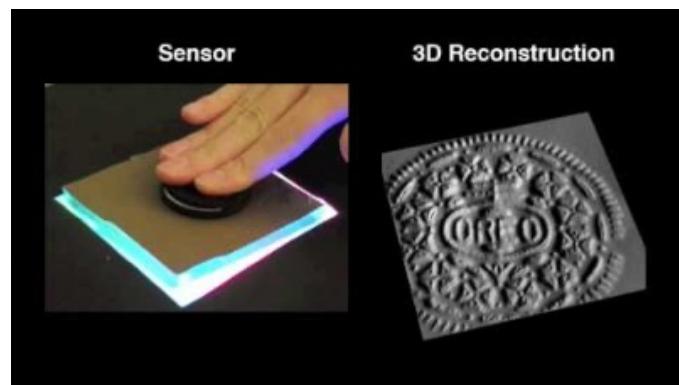
10/18/22

AA 274 | Lecture 7

23

Several other sensors are available

- Classical, e.g.:
 - Radar (possibly using Doppler effect to produce velocity data)
 - Tactile sensors
- Emerging technologies:
 - Artificial skins
 - Neuromorphic cameras



GelSight at Emerging Technologies at SIGGRAPH 2009
<https://www.gelsight.com/>

10/18/22

AA 274 | Lecture 7

24

Introduction to computer vision

- Aim
 - Learn about cameras and camera models



- Readings
 - Siegwart, Nourbakhsh, Scaramuzza. Introduction to Autonomous Mobile Robots. Section 4.2.3.
 - D. A. Forsyth and J. Ponce [FP]. Computer Vision: A Modern Approach (2nd Edition). Prentice Hall, 2011. Chapter 1.
 - R. Hartley and A. Zisserman [HZ]. Multiple View Geometry in Computer Vision. Academic Press, 2002. Chapter 6.1.

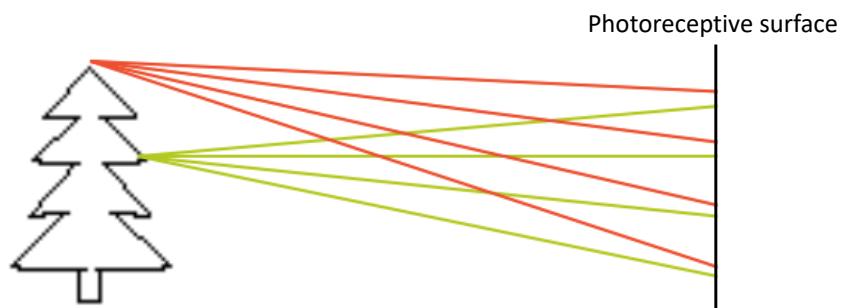
Vision

- Vision: ability to interpret the surrounding environment using light in the visible spectrum reflected by objects in the environment
- Human eye: provides enormous amount of information, ~millions of bits per second
- Cameras (e.g., CCD, CMOS): capture light -> convert to digital image -> process to get relevant information (from geometric to semantic)



How to capture an image of the world?

- Light is reflected by the object and scattered in all directions
- If we simply add a photoreceptive surface, the captured image will be extremely blurred



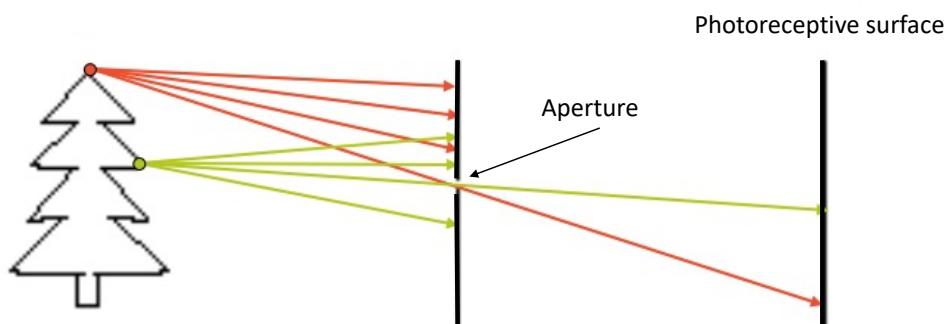
10/18/22

AA 274 | Lecture 7

27

Pinhole camera

- **Idea:** add a barrier to block off most of the rays



- **Pinhole camera:** a camera *without a lens* but with a tiny aperture, a *pinhole*

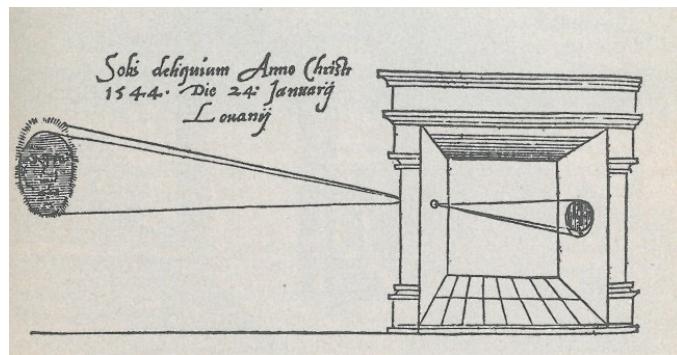
10/18/22

AA 274 | Lecture 7

28

A long history

- Very old idea (several thousands of years BC)
- First clear description from Leonardo Da Vinci (1502)
- Oldest known published drawing of a camera obscura by Gemma Frisius (1544)

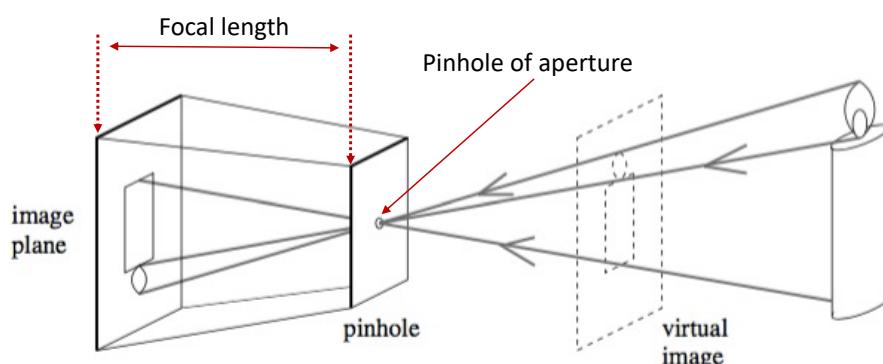


10/18/22

AA 274 | Lecture 7

29

Pinhole camera



Credit: FP Chapter 1

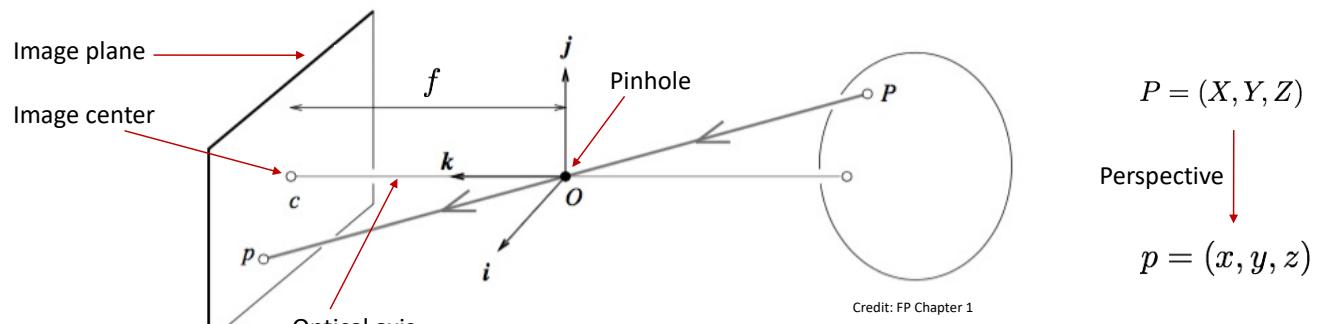
- Perspective projection creates inverted images
- Sometimes it is convenient to consider a *virtual image* associated with a plane lying in front of the pinhole
- Virtual image not inverted but otherwise equivalent to the actual one

10/18/22

AA 274 | Lecture 7

30

Pinhole perspective



- Since P, O , and p are collinear: $\overline{Op} = \lambda \overline{OP}$ for some $\lambda \in R$
- Also, $z=f$, hence

$$\begin{cases} x = \lambda X \\ y = \lambda Y \\ z = \lambda Z \end{cases} \Leftrightarrow \lambda = \frac{x}{X} = \frac{y}{Y} = \frac{z}{Z} \Rightarrow \begin{cases} x = f \frac{X}{Z} \\ y = f \frac{Y}{Z} \end{cases}$$

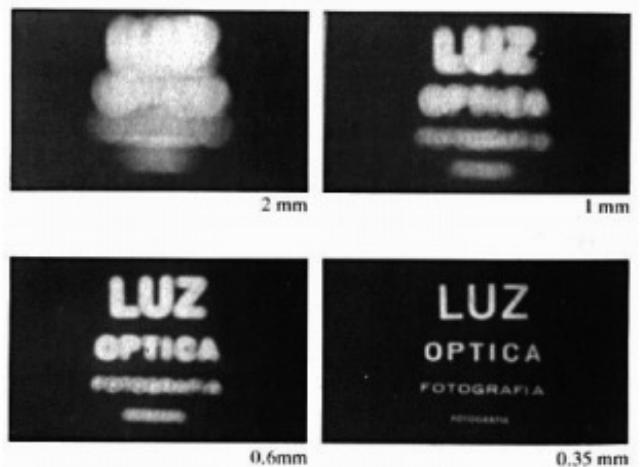
10/18/22

AA 274 | Lecture 7

31

Issues with pinhole camera

- Larger aperture \rightarrow greater number of light rays that pass through the aperture \rightarrow blur
- Smaller aperture \rightarrow fewer number of light rays that pass through the aperture \rightarrow darkness (+ diffraction)
- **Solution:** add a lens to replace the aperture!



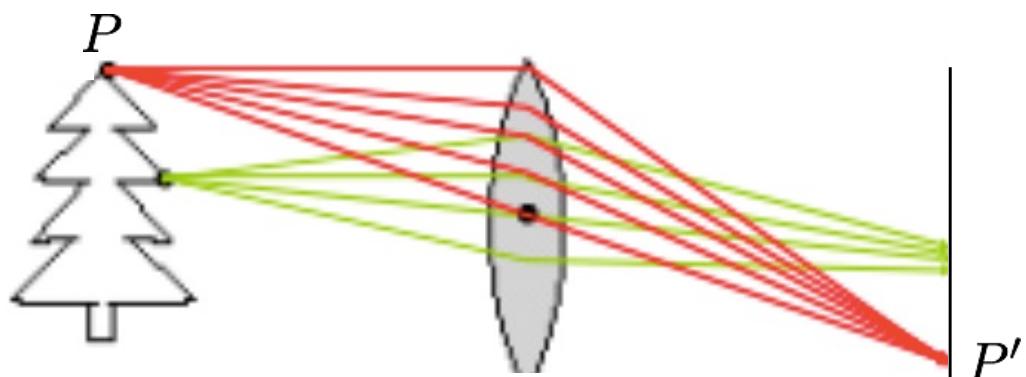
10/18/22

AA 274 | Lecture 7

32

Lenses

- Lens: an optical element that focuses light by means of refraction

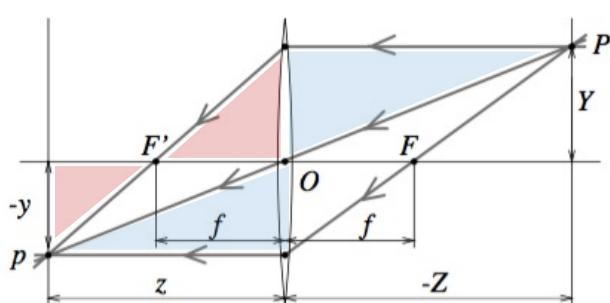


10/18/22

AA 274 | Lecture 7

33

Thin lens model



Credit: FP Chapter 1

Key properties (follows from Snell's law) :

1. Rays passing through O are not refracted
2. Rays parallel to the optical axis are focused on the *focal point* F'
3. All rays passing through P are focused by the thin lens on the point p

- Similar triangles

$$\frac{y}{Y} = \frac{z}{Z} \quad \text{Blue triangles}$$

$$\frac{y}{Y} = \frac{z-f}{f} = \frac{z}{f} - 1 \quad \text{Red triangles}$$

$$\Rightarrow \frac{1}{z} + \frac{1}{Z} = \frac{1}{f}$$

Thin lens equation

10/18/22

AA 274 | Lecture 7

34

Thin lens model

- Key points:
 1. The equations relating the positions of P and p are exactly the same as under pinhole perspective if one considers z as focal length (as opposed to f), since P and p lie on a ray passing through the center of the lens
 2. Points located at a distance $-Z$ from O will be in sharp focus only when the image plane is located at a distance z from O on the other side of the lens that satisfies the thin lens equation
 3. In practice, objects within some range of distances (called depth of field or depth of focus) will be in acceptable focus
 4. Letting $Z \rightarrow \infty$ shows that f is the distance between the center of the lens and the plane where distant objects focus
 5. In reality, lenses suffer from a number of *aberrations*

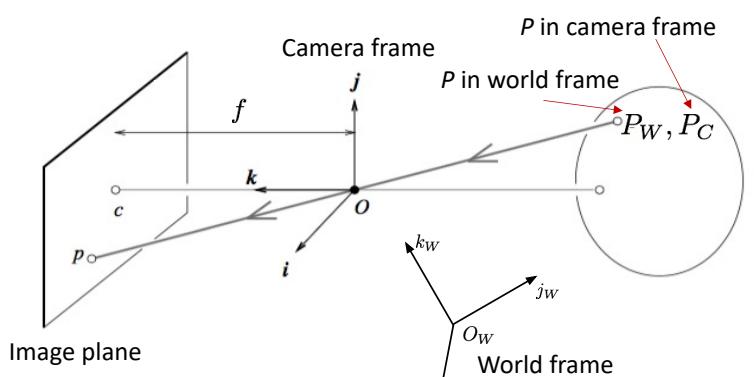
10/18/22

AA 274 | Lecture 7

35

Perspective projection

- **Goal:** find how world points map in the camera image
- Assumption: pinhole camera model (*all results also hold under thin lens model, assuming camera is focused at ∞*)



10/18/22

AA 274 | Lecture 7

Roadmap:

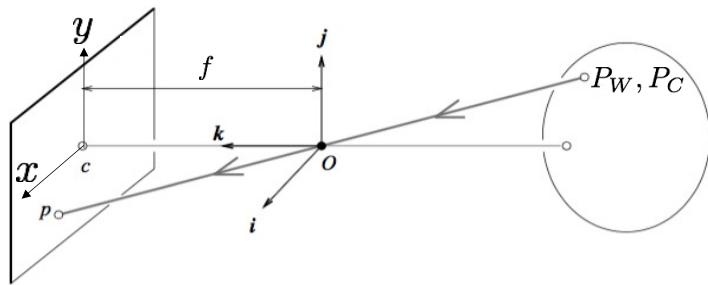
1. Map P_c into p (image plane)
2. Map p into (u,v) (pixel coordinates)
3. Transform P_w into P_c

36

Step 1

- Task: Map $P_c = (X_C, Y_C, Z_C)$ into $p = (x, y)$ (image plane)
- From before

$$\begin{cases} x = f \frac{X_C}{Z_C} \\ y = f \frac{Y_C}{Z_C} \end{cases}$$



10/18/22

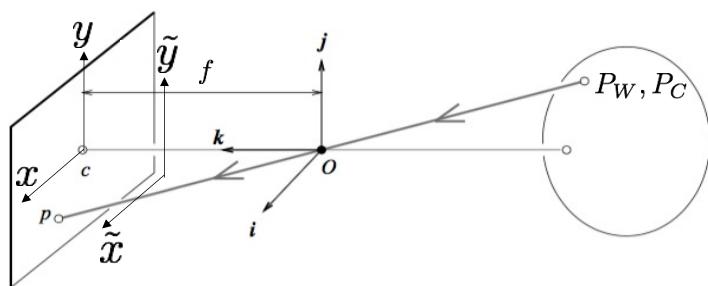
AA 274 | Lecture 7

37

Step 2.a

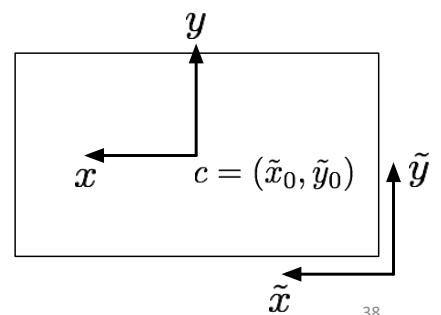
- Actual origin of the camera coordinate system is usually at a corner (e.g., top left, bottom left)

$$\tilde{x} = f \frac{X_C}{Z_C} + \tilde{x}_0, \quad \tilde{y} = f \frac{Y_C}{Z_C} + \tilde{y}_0,$$



10/18/22

AA 274 | Lecture 7



38

Step 2.b

- Task: convert from image coordinates (\tilde{x}, \tilde{y}) to pixel coordinates (u, v)
- Let k_x and k_y be the number of pixels per unit distance in image coordinates in the x and y directions, respectively

$$u = k_x \tilde{x} = k_x f \frac{X_C}{Z_C} + k_x \tilde{x}_0 \quad \Rightarrow \quad u = \alpha \frac{X_C}{Z_C} + u_0$$

$$v = k_y \tilde{y} = k_y f \frac{Y_C}{Z_C} + k_y \tilde{y}_0 \quad \Rightarrow \quad v = \beta \frac{Y_C}{Z_C} + v_0$$

Nonlinear transformation

10/18/22

AA 274 | Lecture 7

39

Homogeneous coordinates

- Goal: represent the transformation as a linear mapping
- Key idea: introduce homogeneous coordinates

Inhomogenous \rightarrow homogeneous

$$\begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} \Rightarrow \lambda \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \mid \quad \begin{pmatrix} x \\ y \\ w \end{pmatrix} \Rightarrow \begin{pmatrix} x/w \\ y/w \\ w \end{pmatrix} \quad \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \Rightarrow \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$$

Homogenous \rightarrow inhomogeneous

10/18/22

AA 274 | Lecture 7

40

Perspective projection in homogeneous coordinates

- Projection can be equivalently written in homogeneous coordinates

$$K \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{pmatrix} \alpha X_c + u_0 Z_c \\ \beta Y_c + v_0 Z_c \\ Z_c \end{pmatrix}$$

Camera matrix/
Matrix of intrinsic parameters P_c in homogeneous
coordinates Homogeneous pixel
coordinates

- In homogeneous coordinates, the mapping is **linear**:

$$\text{Point } p \text{ in homogeneous pixel coordinates} \rightarrow p^h = [K \quad 0_{3 \times 1}] P_C^h \rightarrow \text{Point } P_c \text{ in homogeneous camera coordinates}$$

10/18/22

AA 274 | Lecture 7

41

Skewness

- In some (rare) cases

$$K = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Skew parameter

- When is $\gamma \neq 0$?
 - x- and y-axis of the camera are not perpendicular (unlikely)
 - For example, as a result of taking an image of an image
- Five parameters in total!

10/18/22

AA 274 | Lecture 7

42

Next time: camera models & calibration



10/18/22

AA 274 | Lecture 7

43

Principles of Robot Autonomy I

Camera models and camera calibration

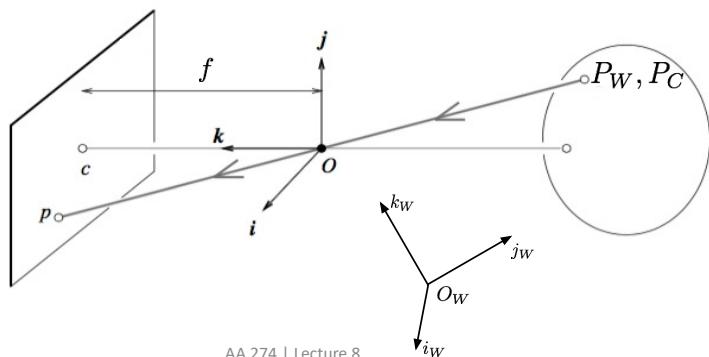


Camera models and camera calibration

- Aim
 - Learn how to calibrate a camera
 - Learn about 3D reconstruction
- Readings
 - SNS: 4.2.3
 - D. A. Forsyth and J. Ponce [FP]. Computer Vision: A Modern Approach (2nd Edition). Prentice Hall, 2011. Chapter 1.
 - R. Hartley and A. Zisserman [HZ]. Multiple View Geometry in Computer Vision. Academic Press, 2002. Chapter 6.1.
 - Z. Zhang. A Flexible New Technique for Camera Calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.

Step 3

- In previous lecture, we have derived a mapping between a point P in the 3D camera reference frame to a point p in the 2D image plane
- Last step is to include in our mapping an additional transformation to account for the difference between the world frame and the 3D camera reference frame

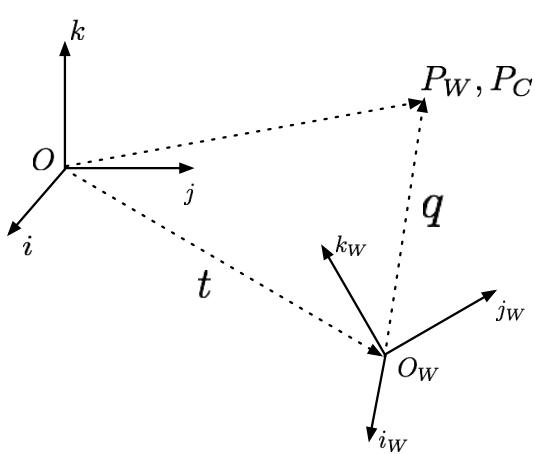


10/20/22

AA 274 | Lecture 8

3

Rigid transformations



$$P_C = t + q$$

$$q = R P_W$$

where R is the rotation matrix relating camera and world frames

$$R = \begin{bmatrix} i_W \cdot i & j_W \cdot i & k_W \cdot i \\ i_W \cdot j & j_W \cdot j & k_W \cdot j \\ i_W \cdot k & j_W \cdot k & k_W \cdot k \end{bmatrix}$$

$$\Rightarrow P_C = t + R P_W$$

10/20/22

AA 274 | Lecture 8

4

Rigid transformations in homogeneous coordinates

$$\begin{pmatrix} P_C \\ 1 \end{pmatrix} = \begin{bmatrix} R & t \\ 0_{1 \times 3} & 1 \end{bmatrix} \begin{pmatrix} P_W \\ 1 \end{pmatrix}$$

Point P_C in homogeneous coordinates Point P_W in homogeneous coordinates

Perspective projection equation

- Collecting all results

$$p^h = [K \quad 0_{3 \times 1}] P_C^h = K[I_{3 \times 3} \quad 0_{3 \times 1}] \begin{bmatrix} R & t \\ 0_{1 \times 3} & 1 \end{bmatrix} P_W^h$$

- Hence

$$p^h = K[R \quad t]P_W^h$$

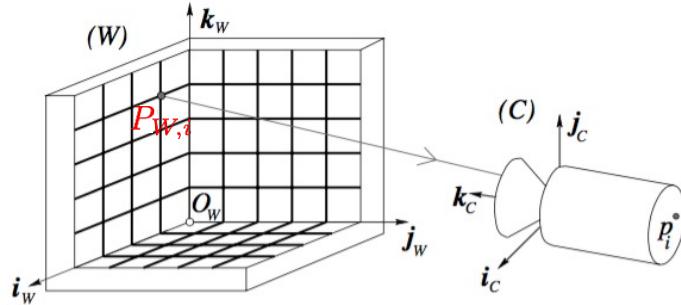
Projection matrix M

Intrinsic parameters Extrinsic parameters

- Degrees of freedom: 4 for K (or 5 if we also include skewness), 3 for R , and 3 for t . Total is 10 (or 11 if we include skewness)

Camera calibration: direct linear transformation method

- **Goal:** find the intrinsic and extrinsic parameters of the camera



Strategy: given known correspondences $p_i \leftrightarrow P_{W,i}$, compute unknown parameters K, R, t by applying perspective projection

$P_{W,1}, P_{W,2}, \dots, P_{W,n}$ with **known** positions in world frame

p_1, p_2, \dots, p_n with **known** positions in image frame

Step 1

- First consider **combined** parameters

$$p_i^h = M P_{W,i}^h, \quad i = 1, \dots, n, \quad \text{where } M = K[R \quad t] = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}$$

- This gives rise to $2n$ component-wise equations, for $i = 1, \dots, n$

$$\begin{aligned} u_i &= \frac{m_1 \cdot P_{W,i}^h}{m_3 \cdot P_{W,i}^h} & \text{or} & \quad u_i (m_3 \cdot P_{W,i}^h) - m_1 \cdot P_{W,i}^h = 0 \\ v_i &= \frac{m_2 \cdot P_{W,i}^h}{m_3 \cdot P_{W,i}^h} & & \quad v_i (m_3 \cdot P_{W,i}^h) - m_2 \cdot P_{W,i}^h = 0 \end{aligned}$$

Calibration problem

- Stacking all equations together

$$\tilde{P}m = 0, \quad \text{where } m = \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix}_{12 \times 1}$$

2n x 12 matrix of known coefficients 12 x 1 vector of unknown coefficients

- \tilde{P} contains in block form the known coefficients stemming from the given correspondences
- To estimate 11 coefficients, we need **at least 6** correspondences

Solution

- To find non-zero solution

$$\begin{aligned} \min_{m \in R^{12}} \quad & \|\tilde{P}m\|^2 \\ \text{subject to} \quad & \|m\|^2 = 1 \end{aligned}$$

- Solution: select eigenvector of $\tilde{P}^T \tilde{P}$ with the smallest eigenvalue
- Readily computed via SVD (singular value decomposition)

Step 2

- Next, we need to extract the camera parameters, i.e., we want to factorize M as

$$M = [KR \quad Kt]$$

- This can be done efficiently (indeed, explicitly) by using RQ factorization, whereby the submatrix $M_{1:3,1:3}$ is decomposed into the product of an upper triangular matrix K and a rotation matrix R
- Calibration will be investigated in **Problem 1 in HW3**

Radial distortion

- So far, we have assumed that a linear model is an accurate model of the imaging process
- For real (non-pinhole) lenses this assumption will not hold



No distortion



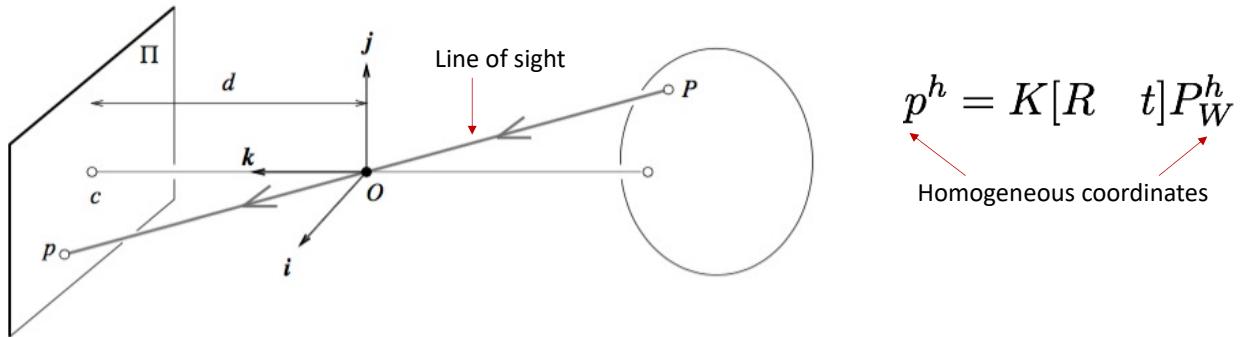
Barrel distortion



Pincushion distortion

Credit: SNS

Measuring depth



Once the camera is calibrated, can we measure the location of a point P in 3D given its known observation p ?

- No: one can only say that P is located *somewhere* along the line joining p and O !

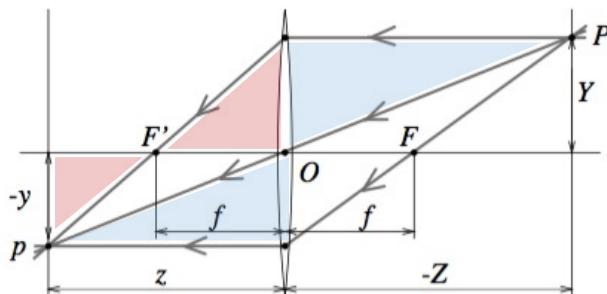
Issues with recovering structure



Recovering structure

- **Structure:** 3D scene to be reconstructed by having access to 2D images
- Common methods
 1. Through recognition of landmarks (e.g., orthogonal walls)
 2. Depth from focus: determines distance to one point by taking multiple images with better and better focus
 3. Stereo vision: processes two distinct images taken at the *same time* and assumes that the relative pose between the two cameras is *known*
 4. Structure from motion: processes two images taken with the same or different cameras at *different times* and from different *unknown* positions

Depth from focus



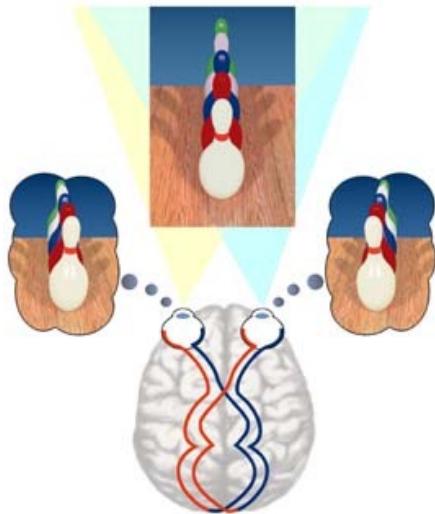
Credit: FP Chapter 1

$$\Rightarrow \frac{1}{z} + \frac{1}{Z} = \frac{1}{f}$$

Thin lens equation

- Take several images until the projection of point P is in focus; let z denote the distance at which the image is in focus
- Since we know z and f , through the thin lens equation we obtain Z

Stereopsis, or why we have two eyes

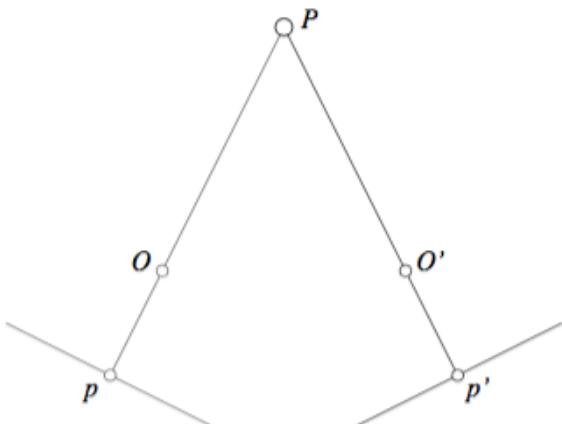


10/20/22

AA 274 | Lecture 8

17

Binocular reconstruction



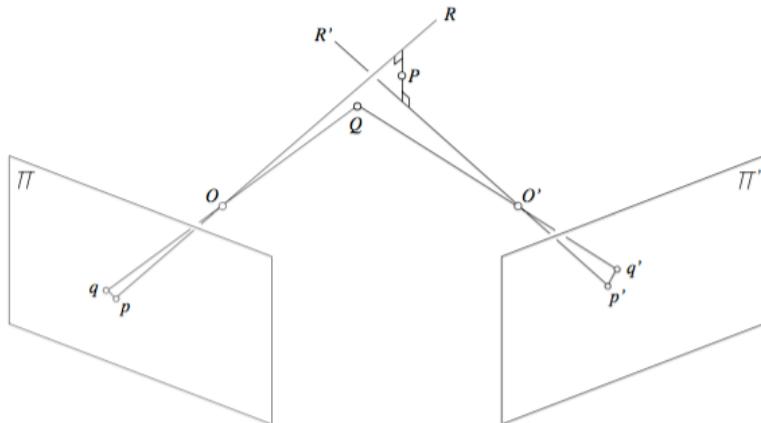
- **Given:** calibrated stereo rig and two image matching points p and p'
- **Find** corresponding scene point by intersecting the two rays \overline{Op} and $\overline{O'p'}$ (process known as **triangulation**)

10/20/22

AA 274 | Lecture 8

18

Approximate triangulation

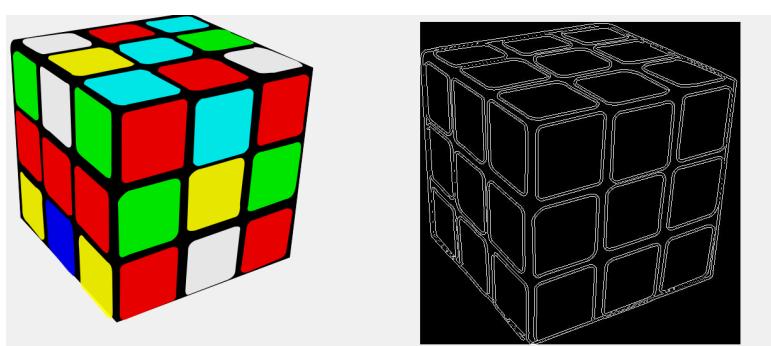


- Due to noise, triangulation problem is often solved as finding the point Q with images q and q' that minimizes

$$d^2(p, q) + d^2(p', q')$$

Re-projection error

Next time: image processing, feature detection & description



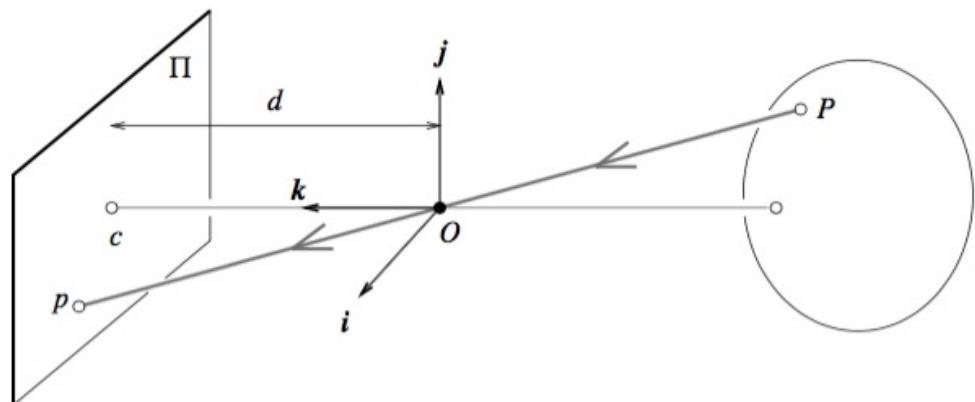
Principles of Robot Autonomy I

Image processing, feature detection, and feature description



From 3D world to 2D images

- So far we have focused on mapping 3D objects onto 2D images and on leveraging such mapping for scene reconstruction
- Next step: how to represent images and infer visual content?



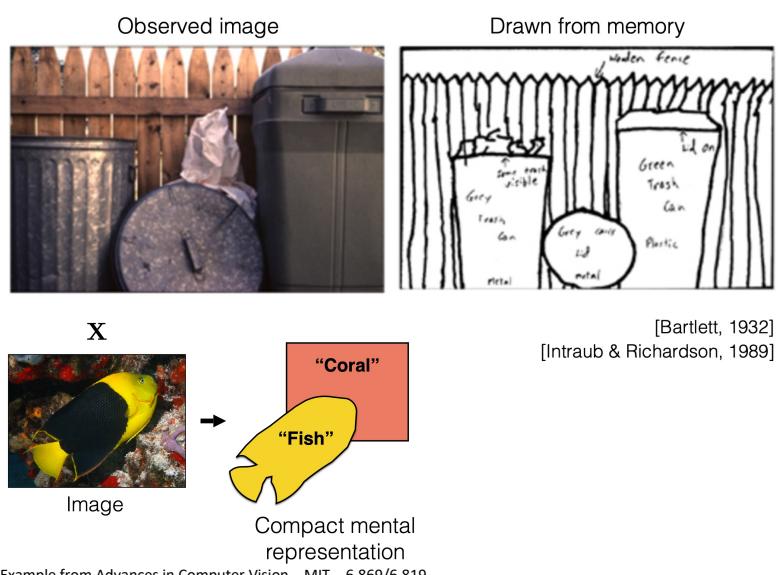
Today's lecture

- Aim
 - Learn fundamental tools in image processing for filtering and detecting similarities
 - Learn how to detect and describe key features in images
- Readings
 - Siegwart, Nourbakhsh, Scaramuzza. Introduction to Autonomous Mobile Robots. Sections 4.3 – 4.5.4.

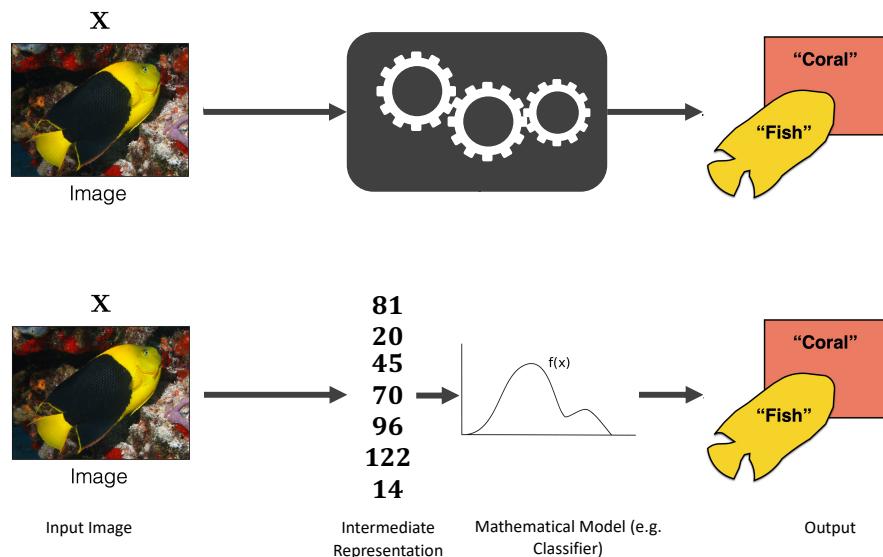
10/25/22

AA 274 | Lecture 9

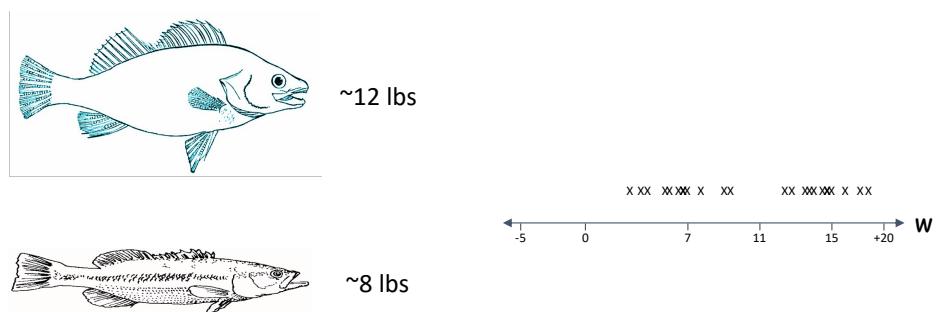
Representations in Computer Vision



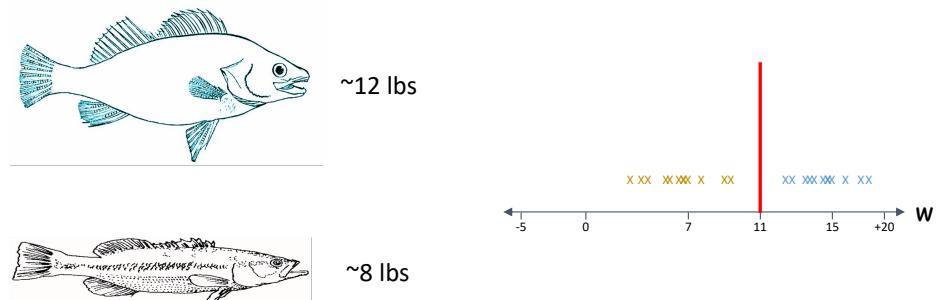
Typical CV Pipeline



Example



Example



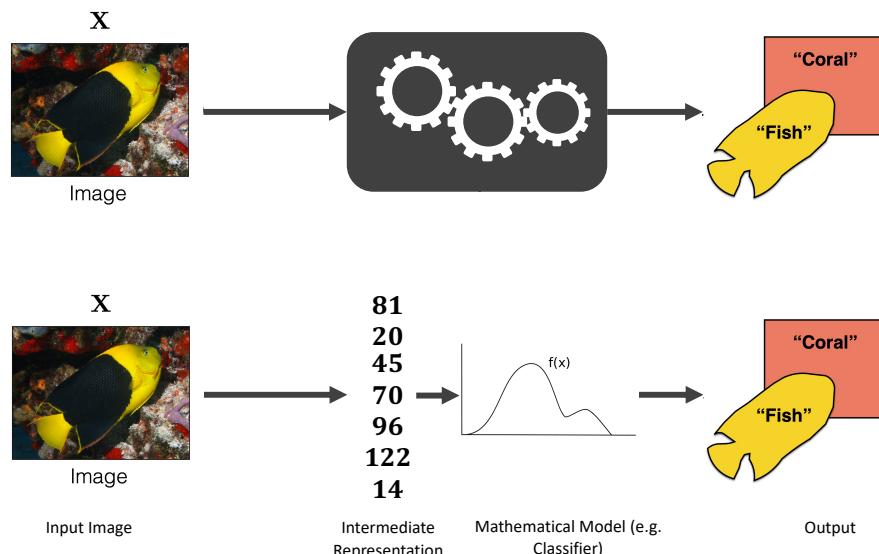
Example from CS331B: Representation Learning in Computer Vision

Example from CS331B: Representation Learning in Computer Vision

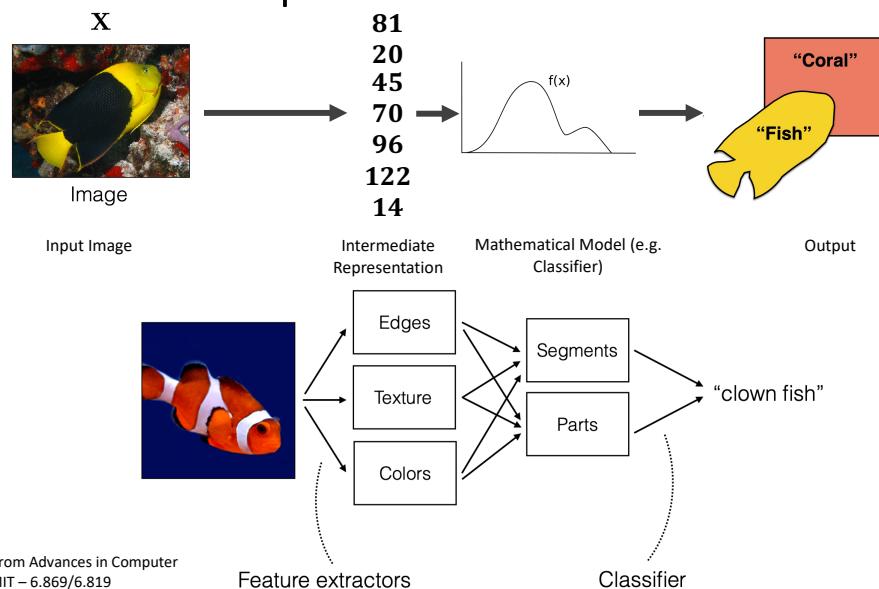
Example



Typical CV Pipeline



Traditional CV Pipeline



Example from Advances in Computer Vision – MIT – 6.869/6.819

Represent these cats with a cat detector!



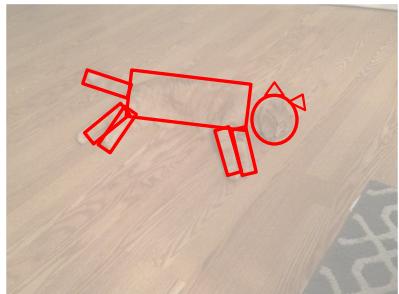
Example from CS331B: Representation Learning in Computer Vision

Represent these cats with a cat detector! (II)



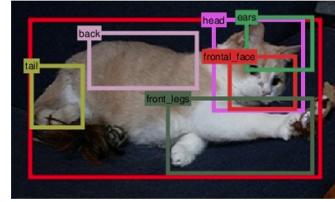
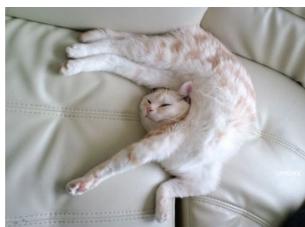
Example from CS331B: Representation Learning in Computer Vision

Represent these cats with a cat detector! (II)



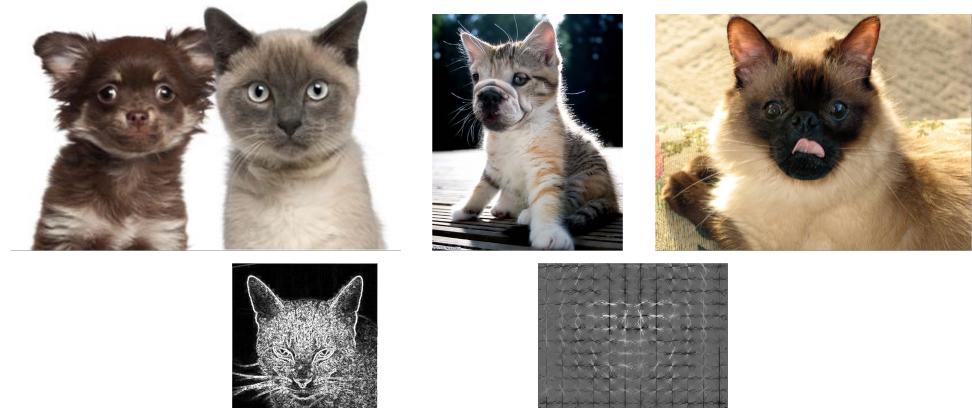
Example from CS331B: Representation Learning in Computer Vision

Represent these cats with a cat detector! (III)



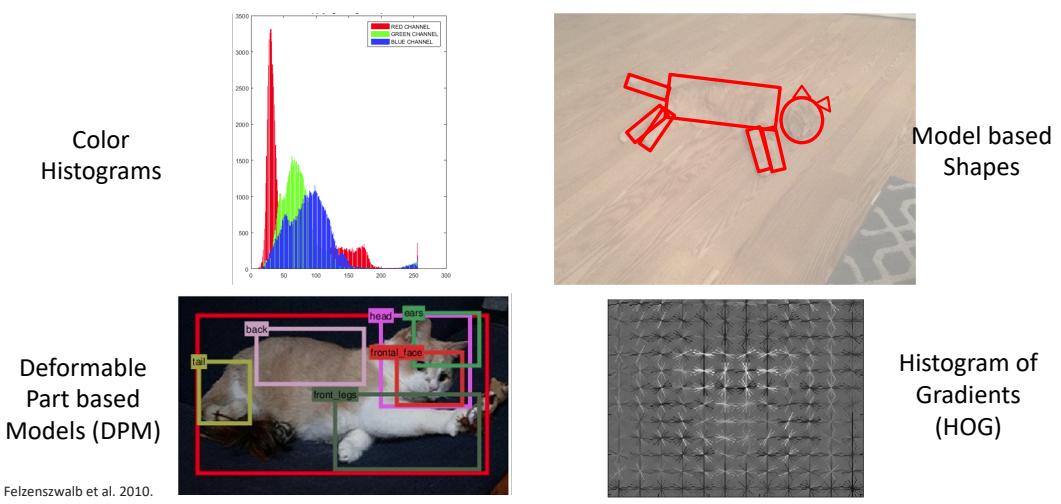
Example from CS331B: Representation Learning in Computer Vision

Represent these cats with a cat detector! (IV)



Example from CS331B: Representation Learning in Computer Vision

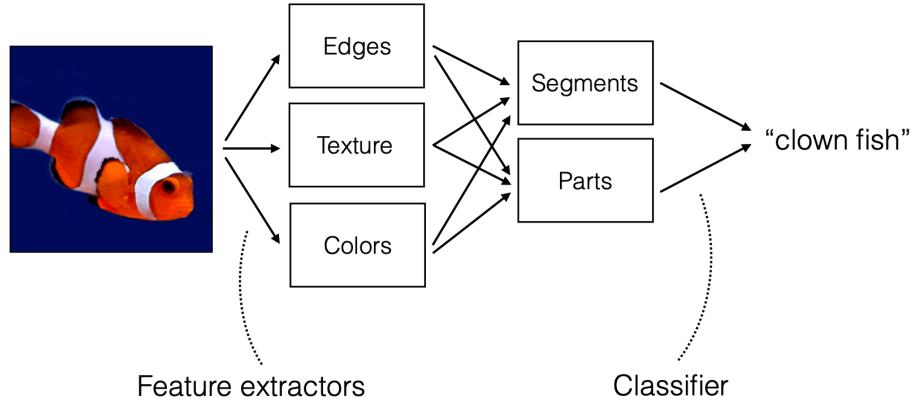
Summary of Traditional Components



Felzenszwalb et al. 2010.
Dalal and Triggs, 2005.
Beis and Lowe, 1997.

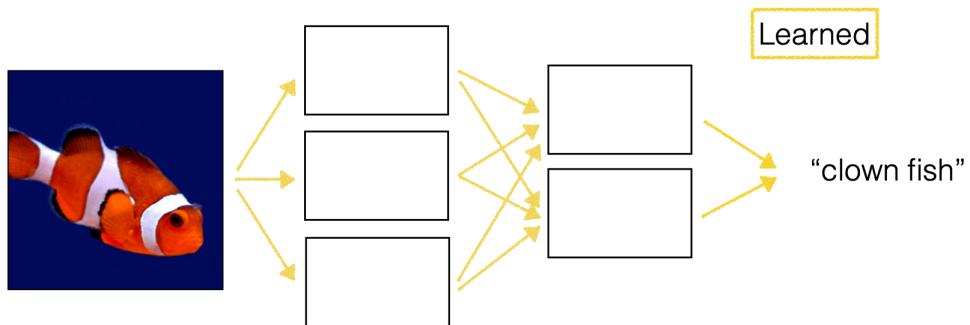
Example from CS331B: Representation Learning in Computer Vision

Traditional CV Pipeline



Example from Advances in Computer Vision – MIT – 6.869/6.819

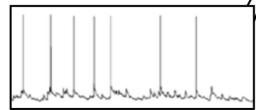
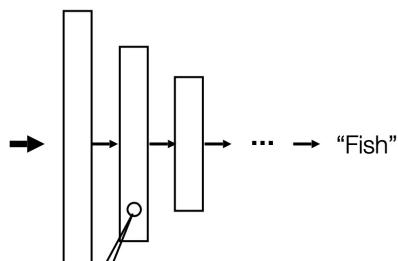
Traditional CV Pipeline



Example from Advances in Computer Vision – MIT – 6.869/6.819

How do you interpret what the network has learned?

Deep Net “Electrophysiology”



[Zeiler & Fergus, ECCV 2014]
[Zhou et al., ICLR 2015]

Example from Advances in Computer Vision – MIT – 6.869/6.819

Visualizing and Understanding CNNs

[Zeiler and Fergus, 2014]

Gabor-like filters learned by **layer 1**

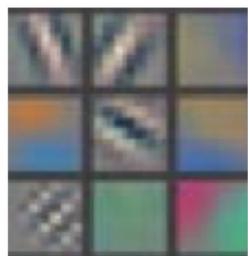


Image patches that activate each of the
layer 1 filters most strongly



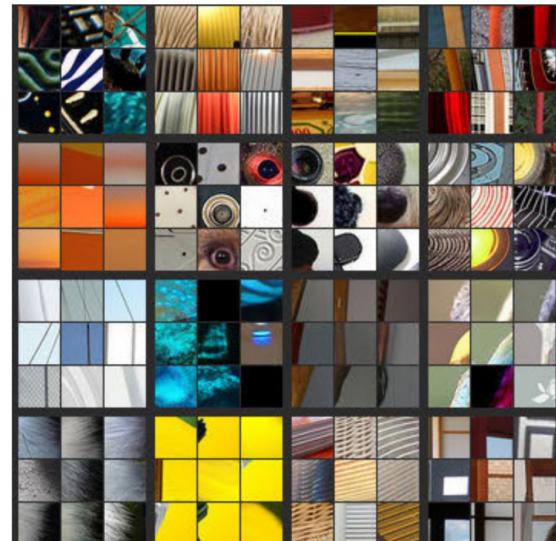
Example from Advances in Computer Vision – MIT – 6.869/6.819

Visualizing and Understanding CNNs

[Zeiler and Fergus, 2014]

Image patches that activate
each of the **layer 2** neurons
most strongly

Example from Advances in Computer Vision – MIT – 6.869/6.819

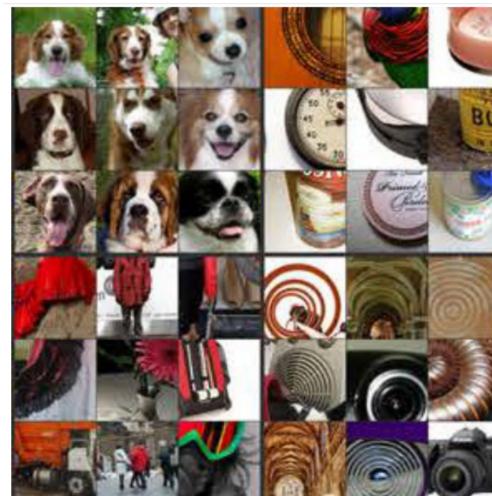


Visualizing and Understanding CNNs

[Zeiler and Fergus, 2014]

Image patches that activate
each of the **layer 4** neurons
most strongly

Example from Advances in Computer Vision – MIT – 6.869/6.819



Visualizing and Understanding CNNs

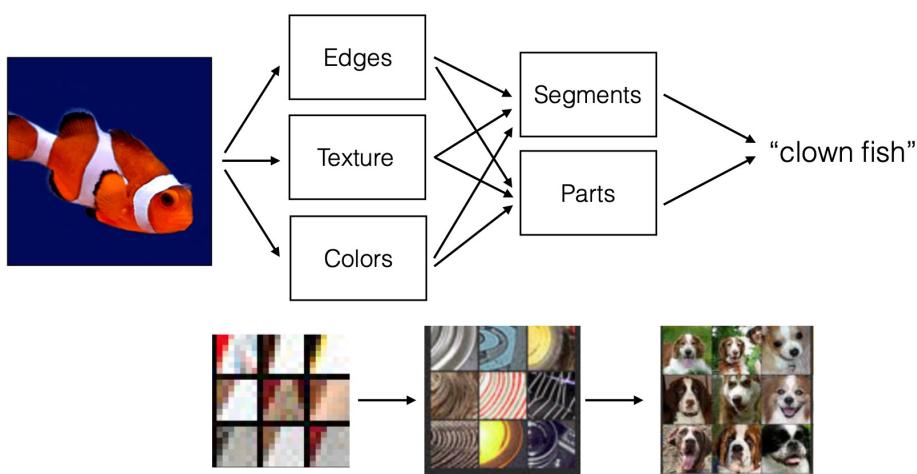
[Zeiler and Fergus, 2014]

Image patches that activate
each of the **layer 5** neurons
most strongly



Visualizing and Understanding CNNs

CNNs *learned* the classical visual recognition pipeline!



Example from Advances in Computer Vision – MIT – 6.869/6.819

How to represent images?



10/25/22

AA 274 | Lecture 9

Image processing pipeline



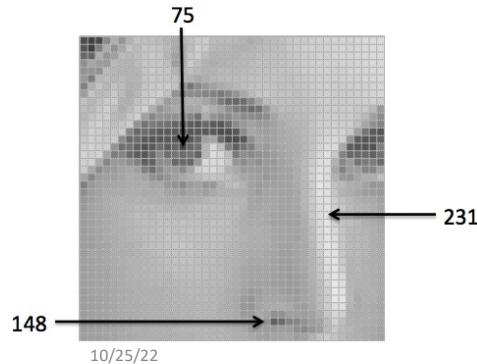
- 1. Signal treatment / filtering
- 2. Feature detection (e.g., DoG)
- 3. Feature description (e.g., SIFT)
- 4. Higher-level processing

10/25/22

AA 274 | Lecture 9

Image filtering

- **Filtering:** process of accepting / rejecting certain frequency components
- Starting point is to view images as functions $I: [a, b] \times [c, d] \rightarrow [0, L]$, where $I(x, y)$ represents intensity at position (x, y)
- A color image would give rise to a vector function with 3 components



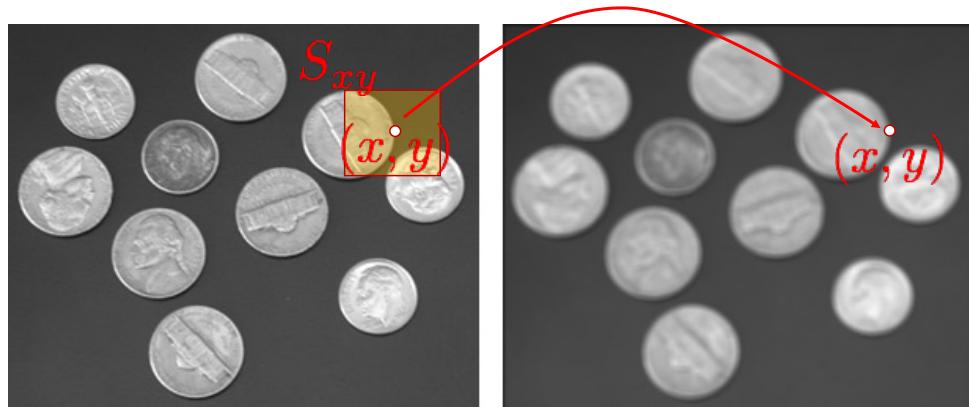
Represented as a matrix

AA 274 | Lecture 9

i	j	88	82	84	88	85	83	80	93	102
	j	88	80	78	80	80	78	73	94	100
i		85	79	80	78	77	74	65	91	99
	j	38	35	40	35	39	74	77	70	65
i		20	25	23	28	37	69	64	60	57
	j	22	26	22	28	40	65	64	59	34
i		24	28	24	30	37	60	58	56	66
	j	21	22	23	27	38	60	67	65	67
i		23	22	22	25	38	59	64	67	66

Spatial filters

- A spatial filter consists of
 1. A neighborhood S_{xy} of pixels around the point (x, y) under examination
 2. A predefined operation F that is performed on the image pixels within S_{xy}



10/25/22

AA 274 | Lecture 9

Linear spatial filters

- Filters can be linear or non-linear
- We will focus on linear spatial filters

$$I'(x, y) = F \circ I = \sum_{i=-N}^N \sum_{j=-M}^M F(i, j) I(x + i, y + j)$$

↑ ↑ ↑
Filtered image Filter mask Original image

- Filter F (of size $(2N + 1) \times (2M + 1)$) is usually called a mask, kernel, or window
- Dealing with boundaries: e.g., pad, crop, extend, or wrap

10/25/22

AA 274 | Lecture 9

Filter example #1: moving average

- The moving average filter returns the average of the pixels in the mask
- Achieves a smoothing effect (removes sharp features)
- E.g., for a *normalized* 3×3 mask

$$F = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Generated with a 5×5 mask

10/25/22

AA 274 | Lecture 9

Filter example #2: Gaussian smoothing

- Gaussian function

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

- To obtain the mask, sample the function about its center
- E.g., for a *normalized* 3×3 mask with $\sigma = 0.85$

$$G = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Convolution

- Still a linear filter, defined as

$$I'(x, y) = F * I = \sum_{i=-N}^N \sum_{j=-M}^M F(i, j) I(x-i, y-j)$$

- Same as correlation, but with negative signs for the filter indices
- Correlation and convolution are identical when the filter is symmetric
- Convolution enjoys the associativity property

$$F * (G * I) = (F * G) * I$$

- Example: smooth image & take derivative = convolve derivative filter with Gaussian filter & convolve the resulting filter with the image

Separability of masks

- A mask is separable if it can be broken down into the convolution of two kernels

$$F = F_1 * F_2$$

- If a mask is separable into “smaller” masks, then it is often cheaper to apply F_1 followed by F_2 , rather than F directly
- Special case: mask representable as outer product of two vectors (equivalent to two-dimensional convolution of those two vectors)
- If mask is $M \times M$, and image has size $w \times h$, then complexity is
 - $O(M^2wh)$ with no separability
 - $O(2Mwh)$ with separability into outer product of two vectors

Example of separable masks

- Moving average

$$F = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} [1 \quad 1 \quad 1]$$

- Gaussian smoothing

$$\begin{aligned} G_\sigma(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= g_\sigma(x) \cdot g_\sigma(y) \end{aligned}$$

Differentiation

- Derivative of discrete function (centered difference)

$$\frac{\partial I}{\partial x} = I(x+1, y) - I(x-1, y) \quad F_x = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$
$$\frac{\partial I}{\partial y} = I(x, y+1) - I(x, y-1) \quad F_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

- Derivative as a convolution operation; e.g., Sobel masks:

Along x direction

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

10/25/22

Along y direction

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

AA 274 | Lecture 9

Note: masks are **mirrored**
In convolution

Similarity measures

- Filtering can also be used to determine similarity across images (e.g., to detect correspondences)

$$SAD = \sum_{i=-n}^n \sum_{j=-m}^m |I_1(x+i, y+j) - I_2(x'+i, y'+j)| \quad \text{Sum of absolute differences}$$

$$SSD = \sum_{i=-n}^n \sum_{j=-m}^m [I_1(x+i, y+j) - I_2(x'+i, y'+j)]^2 \quad \text{Sum of squared differences}$$

10/25/22

AA 274 | Lecture 9

Detectors

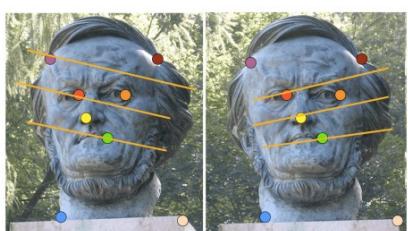
- **Goal:** detect **local features**, i.e., image patterns that differ from immediate neighborhood in terms of intensity, color, or texture
- We will focus on
 - Edge detectors
 - Corner detectors

10/25/22

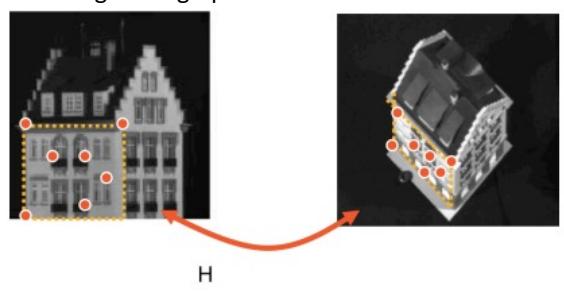
AA 274 | Lecture 9

Use of detectors/descriptors: examples

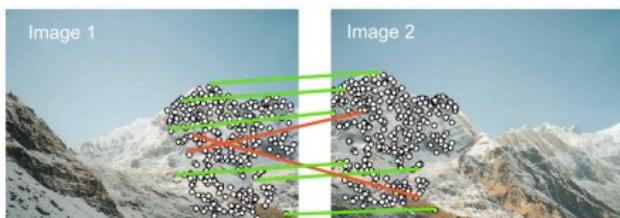
Stereo reconstruction



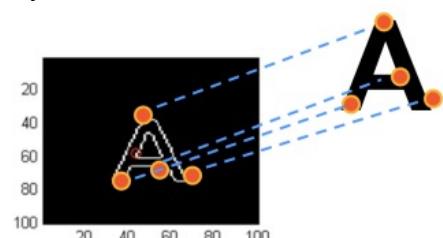
Estimating homographic transformations



Panorama stitching



Object detection



10/25/22

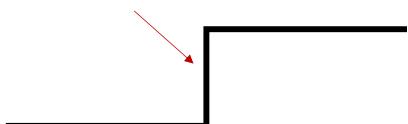
AA 274 | Lecture 9

Edge detectors

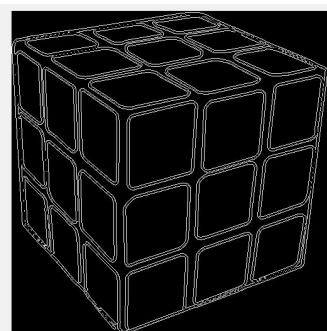
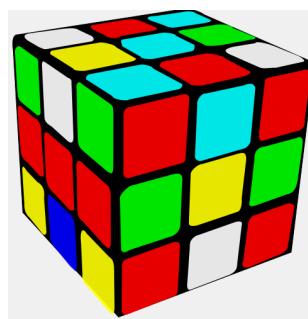
- **Edge:** region in an image where there is a *significant* change in intensity values along one direction, and *negligible* change along the orthogonal direction

In 1D

Magnitude of 1st order derivative is large,
2nd order derivative is equal to zero



In 2D



10/25/22

AA 274 | Lecture 9

Criteria for “good” edge detection

- **Accuracy:** minimize false positives and negatives
- **Localization:** edges must be detected as close as possible to the true edges
- **Single response:** detect one edge per real edge in the image

10/25/22

AA 274 | Lecture 9

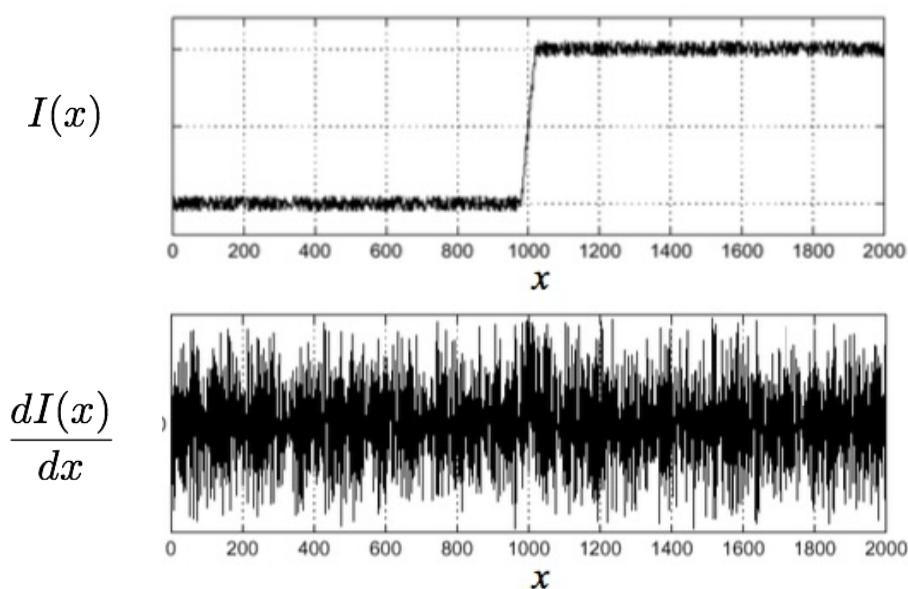
Strategy to design an edge detector

- Two steps:
 1. **Smoothing**: smooth the image to reduce noise prior to differentiation (step 2)
 2. **Differentiation**: take derivatives along x and y directions to find locations with high gradients

10/25/22

AA 274 | Lecture 9

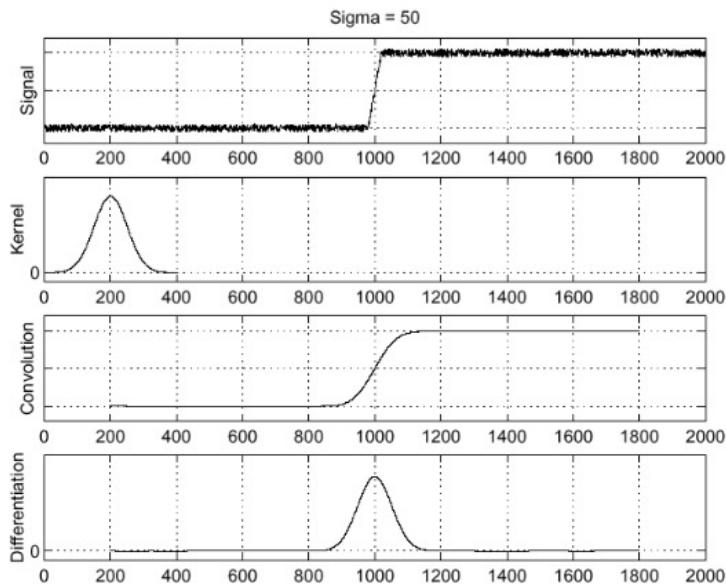
1D case: differentiation without smoothing



10/25/22

AA 274 | Lecture 9

1D case: differentiation with smoothing



$I(x)$

$g_\sigma(x)$

Edges occur at
maxima or
minima of $s'(x)$

$$s(x) = g_\sigma(x) * I(x)$$

$$s'(x) = \frac{d}{dx} * s(x)$$

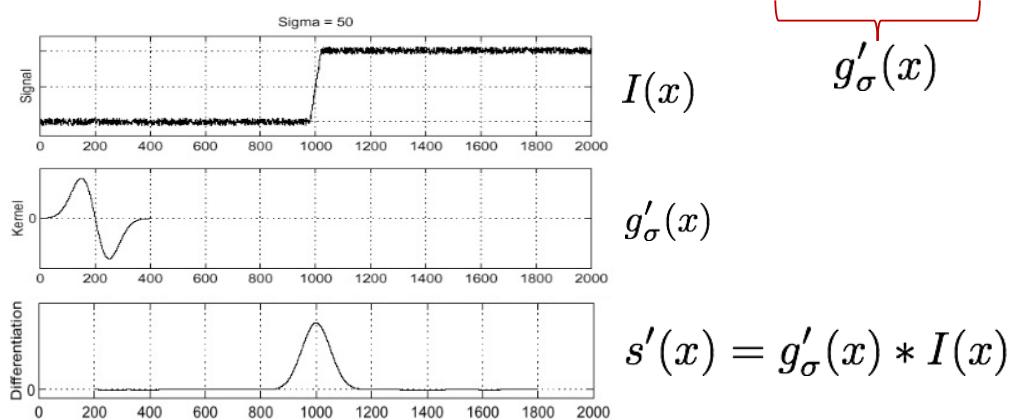
10/25/22

AA 274 | Lecture 9

A better implementation

- Convolution theorem:

$$s'(x) = \frac{d}{dx} * (g_\sigma(x) * I(x)) = \left(\frac{d}{dx} * g_\sigma(x) \right) * I(x)$$



$I(x)$

$g'_\sigma(x)$

$g'_\sigma(x)$

$$s'(x) = g'_\sigma(x) * I(x)$$

10/25/22

AA 274 | Lecture 9

Edge detection in 2D

1. Find the gradient of smoothed image in both directions

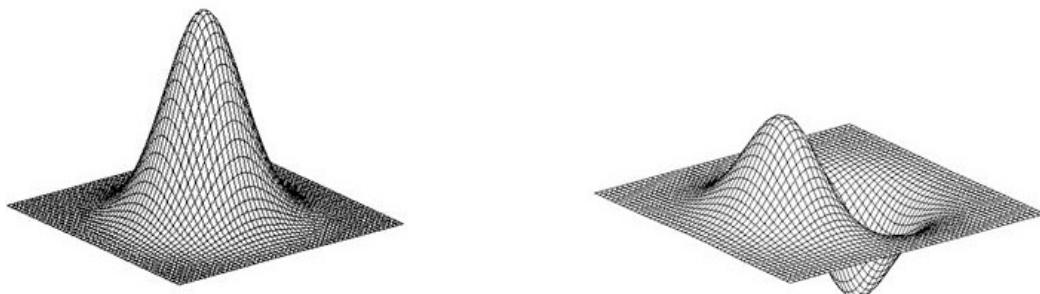
$$\nabla S := \begin{bmatrix} \frac{\partial}{\partial x} * (G_\sigma * I) \\ \frac{\partial}{\partial y} * (G_\sigma * I) \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial}{\partial x} * G_\sigma \right) * I \\ \left(\frac{\partial}{\partial y} * G_\sigma \right) * I \end{bmatrix} = \begin{bmatrix} G_{\sigma,x} * I \\ G_{\sigma,y} * I \end{bmatrix} := \begin{bmatrix} S_x \\ S_y \end{bmatrix}$$

2. Compute the magnitude $|\nabla S| = \sqrt{S_x^2 + S_y^2}$ and discard pixels below a certain threshold
3. Non-maximum suppression: identify local maxima of $|\nabla S|$

10/25/22

AA 274 | Lecture 9

Derivative of Gaussian filter



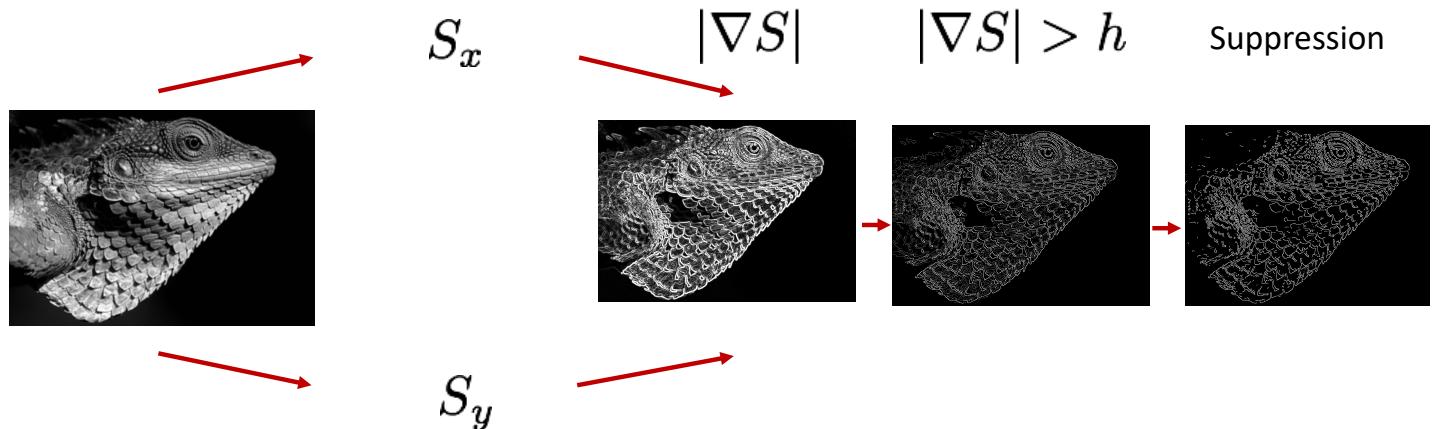
$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

$$\frac{\partial G_\sigma(x, y)}{\partial x}$$

10/25/22

AA 274 | Lecture 9

Canny edge detector



10/25/22

AA 274 | Lecture 9

Corner detectors

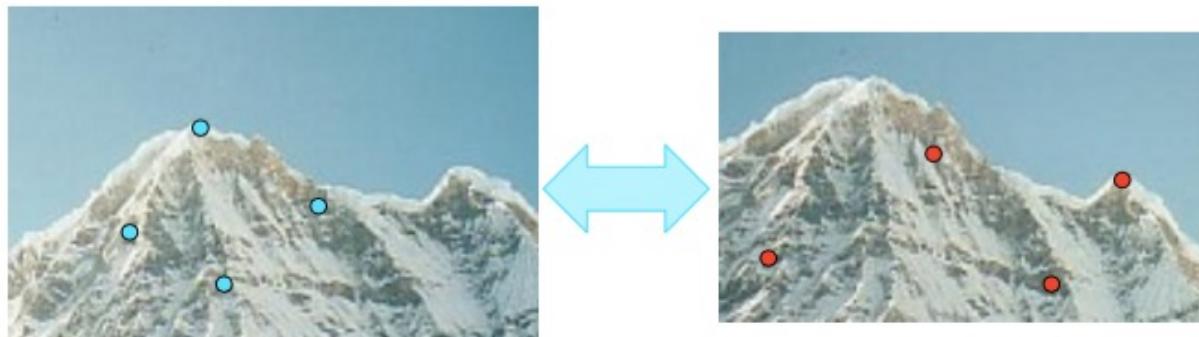
Key criteria for “good” corner detectors

1. **Repeatability**: same feature can be found in multiple images despite geometric and photometric transformations
2. **Distinctiveness**: information carried by the patch surrounding the feature should be as distinctive as possible

10/25/22

AA 274 | Lecture 9

Repeatability

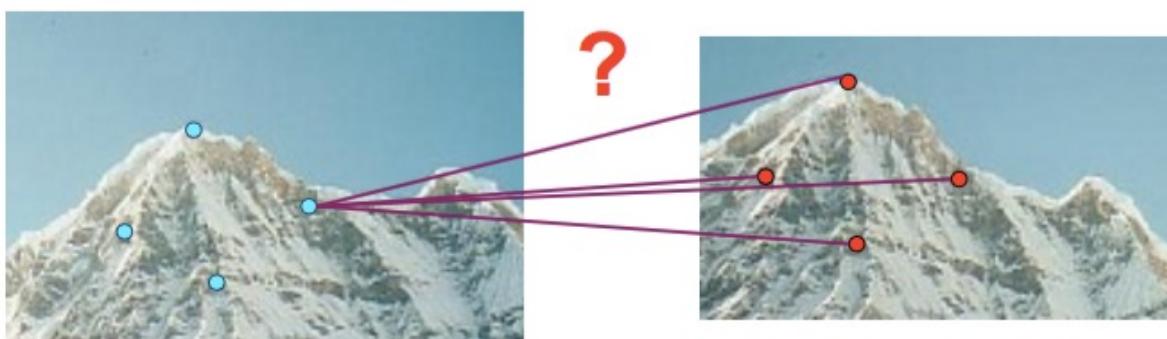


Without repeatability, matching is impossible

10/25/22

AA 274 | Lecture 9

Distinctiveness



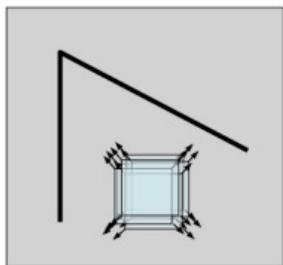
Without distinctiveness, it is not possible to establish reliable correspondences; distinctiveness is key for having a useful descriptor

10/25/22

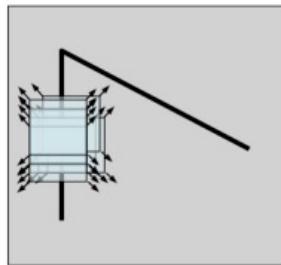
AA 274 | Lecture 9

Finding corners

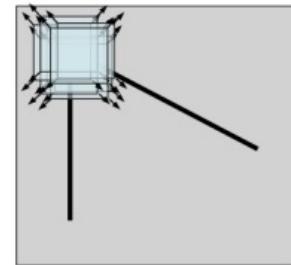
- **Corner:** intersection of two or more edges
- Geometric intuition for corner detection: explore how intensity changes as we shift a window



Flat: no changes in any direction



Edge: no change along the edge direction



Corner: changes in all directions

10/25/22

AA 274 | Lecture 9

Harris detector: example

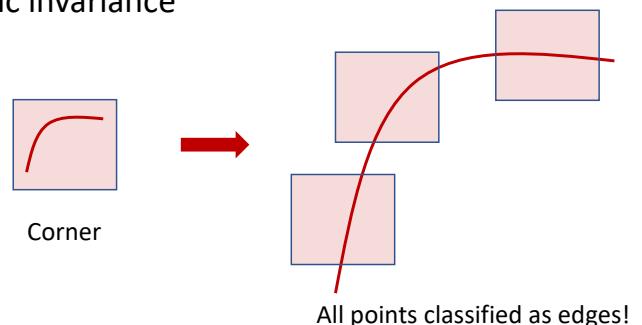


10/25/22

AA 274 | Lecture 9

Properties of Harris detectors

- Widely used
- Detection is invariant to
 - Rotation -> geometric invariance
 - Linear intensity changes -> photometric invariance
- Detection is **not** invariant to
 - Scale changes
 - Geometric affine changes

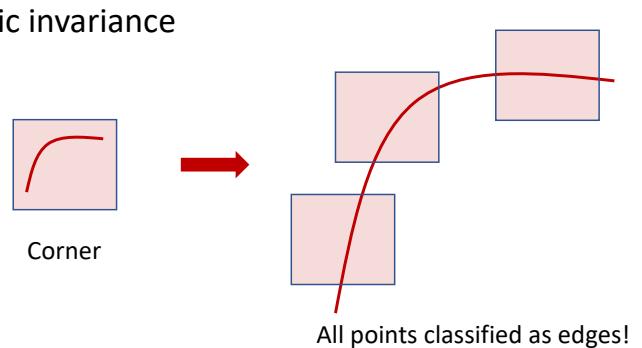


10/25/22

AA 274 | Lecture 9

Properties of Harris detectors

- Widely used
- Detection is invariant to
 - Rotation -> geometric invariance
 - Linear intensity changes -> photometric invariance
- Detection is **not** invariant to
 - Scale changes
 - Geometric affine changes



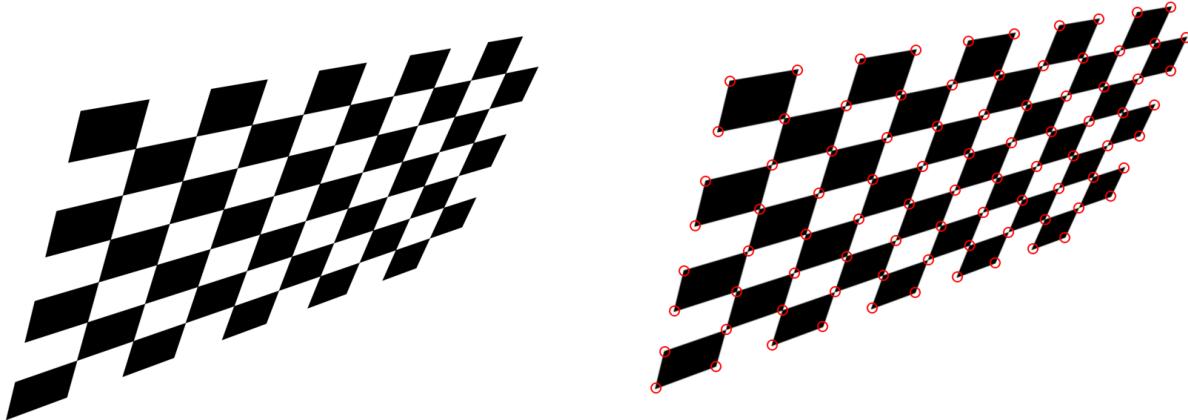
→ Scale-invariant detection, such as

1. Harris-Laplacian
2. in SIFT (specifically, Difference of Gaussians (DoG))

10/25/22

AA 274 | Lecture 9

Example Application of Corner Detector



10/19/21

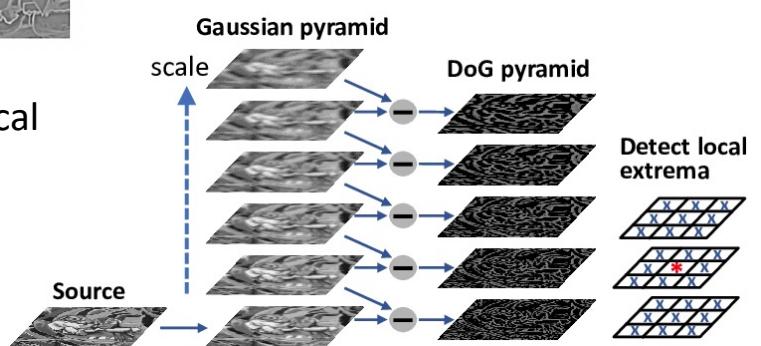
AA 274 | Lecture 9

Difference of Gaussians (DoG)

$$g_{(1)} - g_{(2)} = \text{Laplacian of Gaussian (LOG)}$$

$$f * g_{(1)} - f * g_{(2)} = f * (g_{(1)} - g_{(2)})$$

- Features are detected as local extrema in scale and space

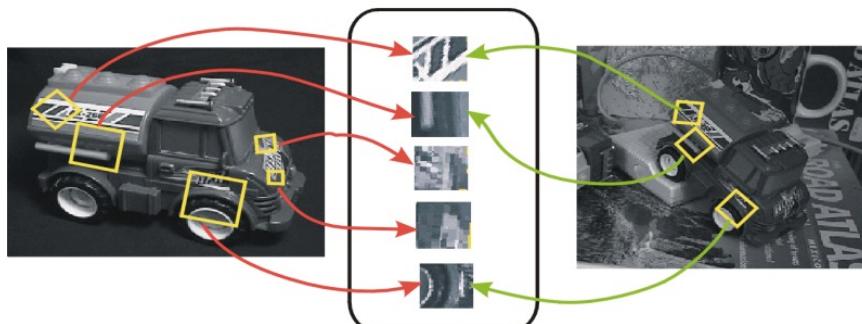


10/25/22

AA 274 | Lecture 9

Descriptors

- **Goal:** describe keypoints so that we can compare them across images or use them for object detection or matching
- Desired properties:
 - Invariance with respect to pose, scale, illumination, etc.
 - Distinctiveness

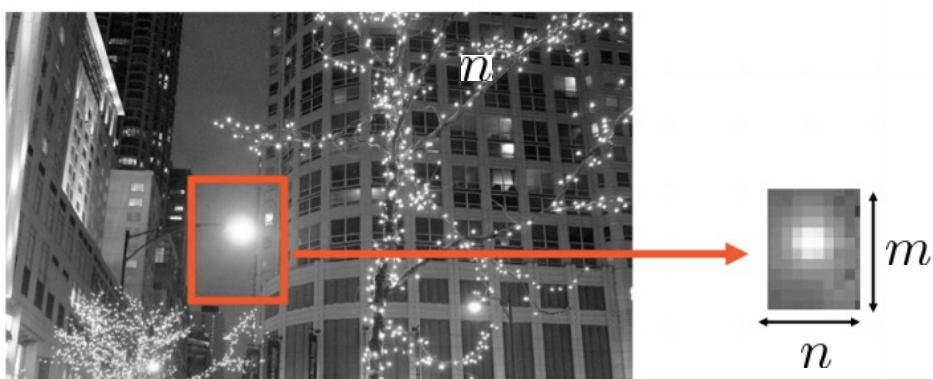


10/25/22

AA 274 | Lecture 9

Simplest descriptor

- Naïve descriptor: associate with a given keypoint an $n \times m$ window of pixel intensities centered at that keypoint
- Window can be normalized to make it invariant to illumination



- Main drawbacks
1. Sensitive to pose
 2. Sensitive to scale
 3. Poorly distinctive

10/25/22

AA 274 | Lecture 9

Popular detectors / descriptors

- SIFT (Scale-Invariant Feature Transformation)
 - Invariant to rotation and scale, but computationally demanding
 - SIFT descriptor is a 128-dimensional vector!
- SURF
- FAST
- BRIEF
- ORB
- BRISK
- LIFT

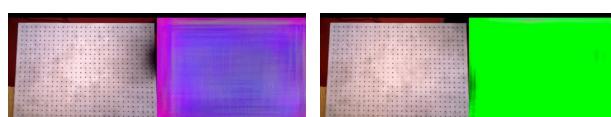
10/25/22

AA 274 | Lecture 9

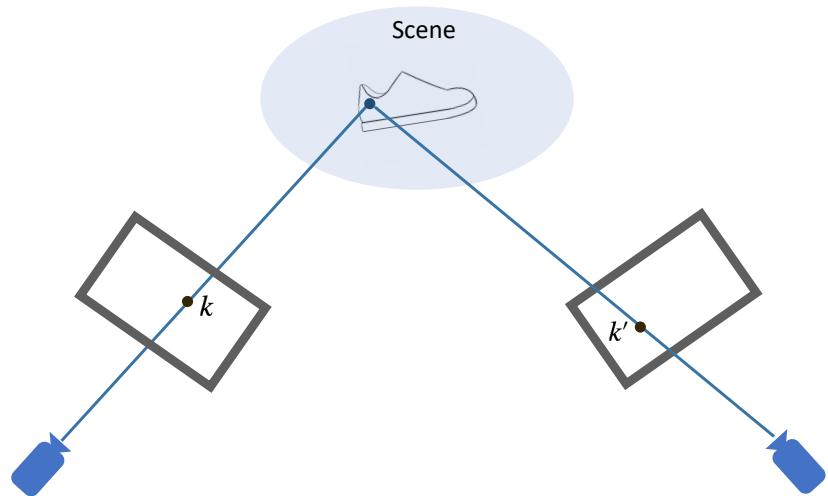
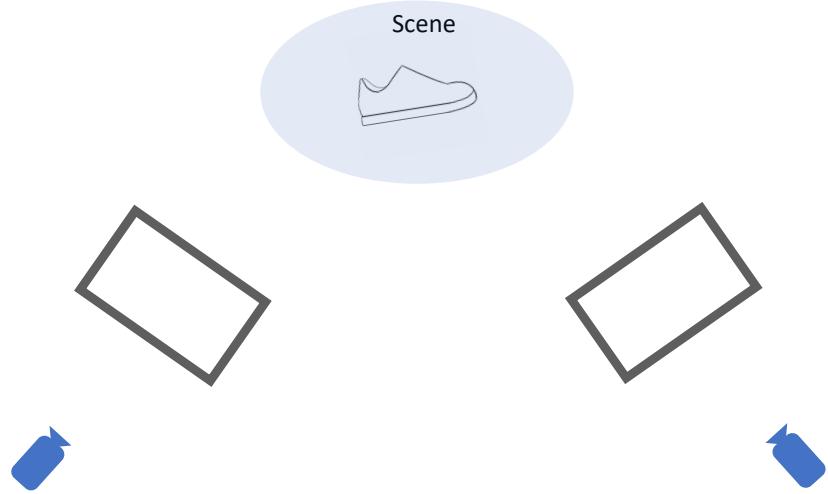
A case study for learning-based Descriptors Dense Object Nets

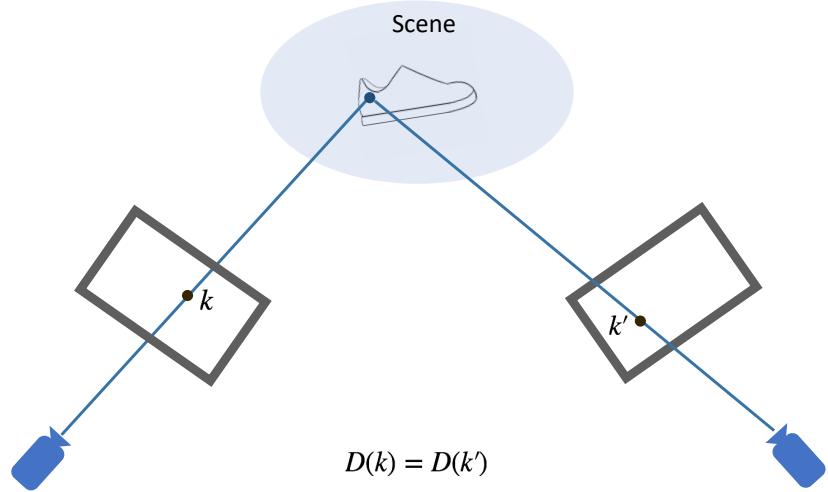
Learning Dense Visual Object Descriptors
By and For Robotic Manipulation. CORL 2018

Peter R. Florence, Lucas Manuelli, Russ Tedrake

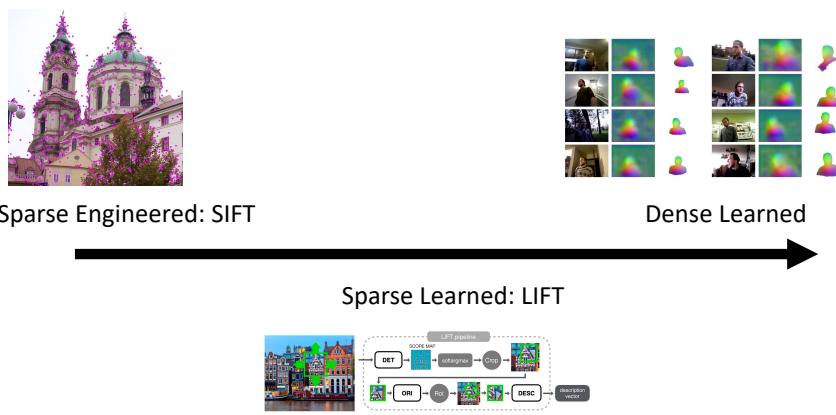


Slides adapted from CS326 by Kevin Zakka and Sriram Somasundaram





A Brief History

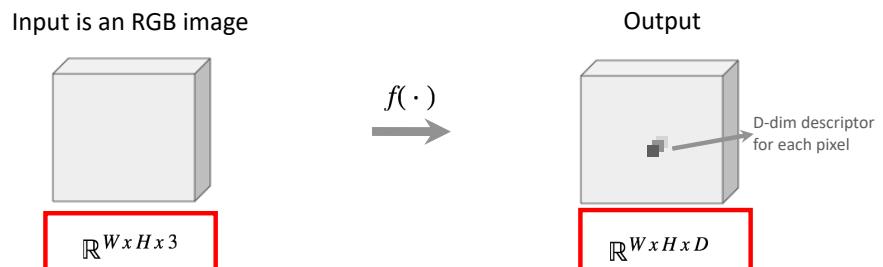


Why Dense?



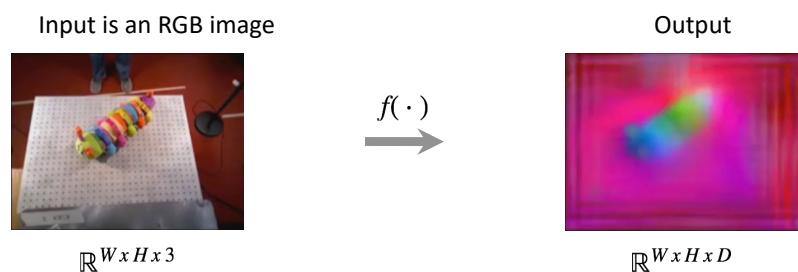
Bachrach et. al.

Dense Descriptors

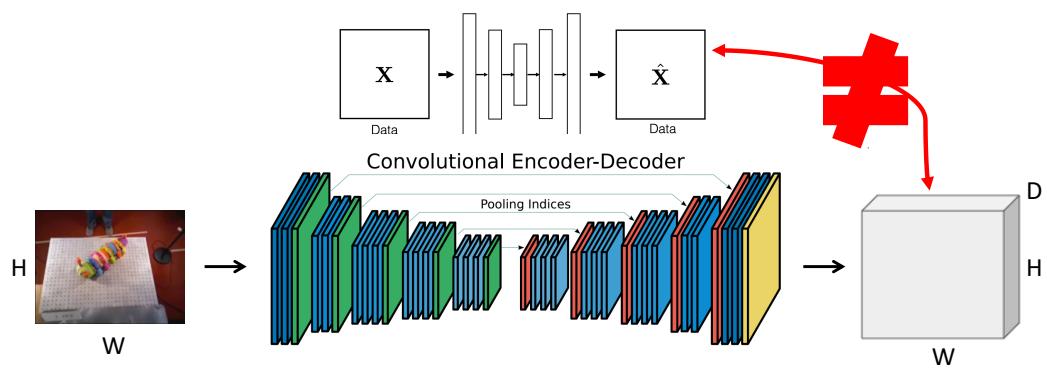


Pay attention to the difference in Dimensionality

Dense Descriptors



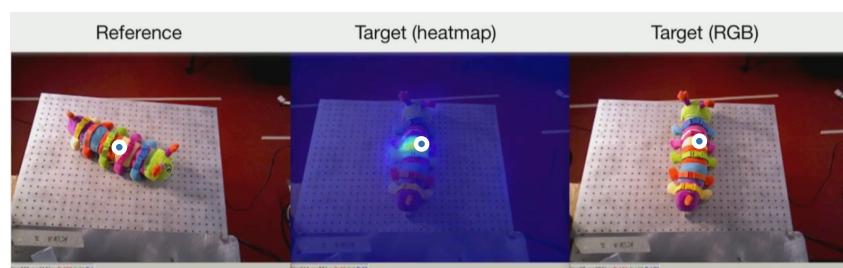
Network Architecture



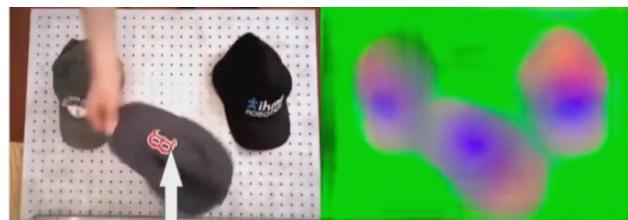
Single Object



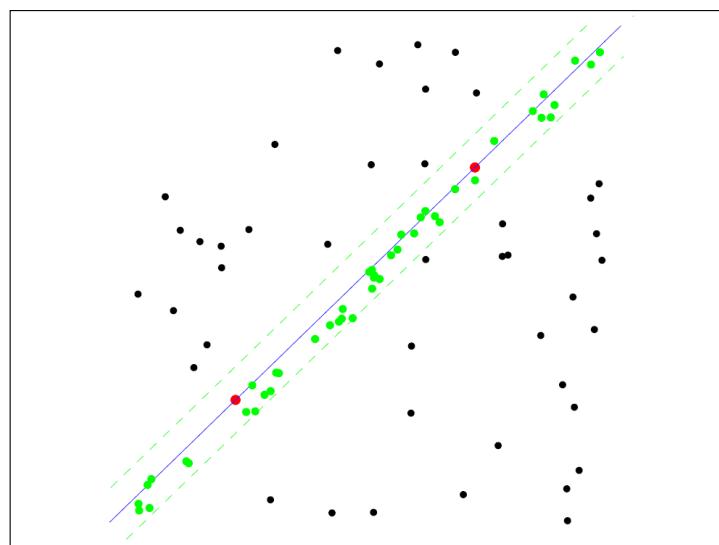
Learned Dense Correspondences



Class consistent descriptors



Next time



Principles of Robot Autonomy I

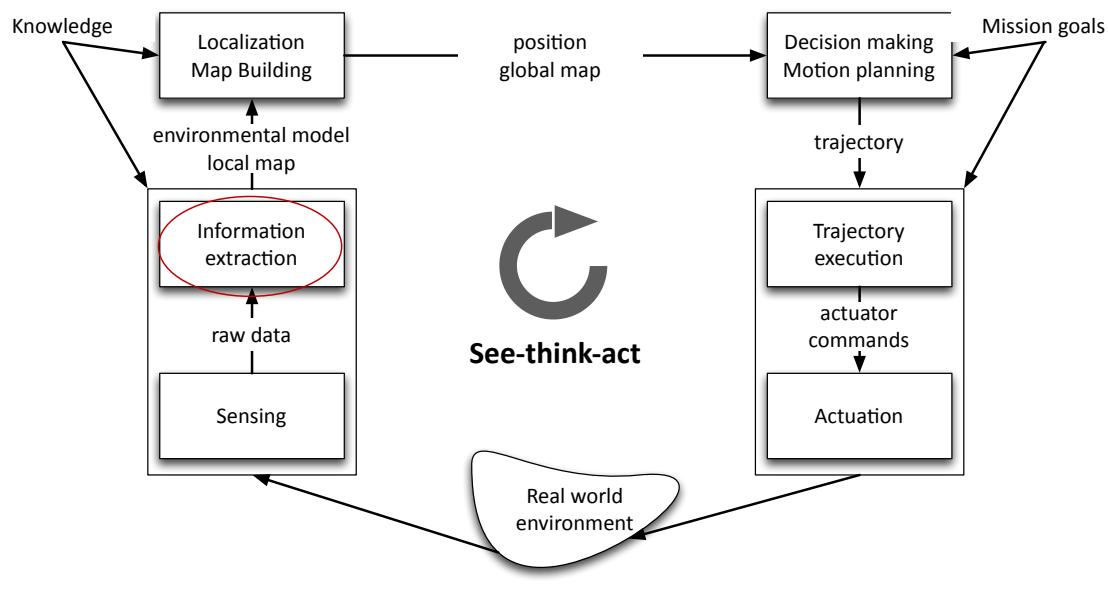
Information extraction



Techniques for information extraction

- Aim
 - Learn how to extract information from sensor measurements
- Readings
 - Siegwart, Nourbakhsh, Scaramuzza. Introduction to Autonomous Mobile Robots. Sections: 4.1.3, 4.6.1 - 4.6.5, 4.7.1 - 4.7.4

The see-think-act cycle



10/27/22

AA 274A | Lecture 10

7

Information extraction

- Next step is to extract *information* from images, such as
 - Geometric primitives (e.g., lines and circles): useful, for example, for robot localization and mapping
 - Object recognition and scene understanding: useful, for example, for localization within a topological map and for high-level reasoning

10/27/22

AA 274A | Lecture 10

8

Geometric feature extraction

- **Geometric feature extraction:** extract geometric primitives from sensor data (e.g., range data)
- Examples: line, circles, corners, planes, etc.
- We focus on *line extraction* from range data (a quite common task); other geometric feature extraction tasks are conceptually analogous
- The two main problems of line extraction from range data
 1. Which points belong to which line? → *segmentation*
 2. Given an association of points to a line, how to estimate line parameters? → *fitting*

Step #2: line fitting

- **Goal:** fit a line to a set of sensor measurements

- It is useful to work in polar coordinates:

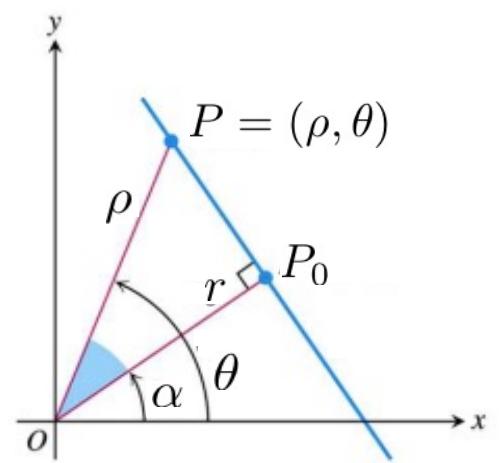
$$x = \rho \cos \theta, \quad y = \rho \sin \theta$$

- Equation of a line in polar coordinates

- Let $P = (\rho, \theta)$ be an arbitrary point on the line
 - Since P, P_0, O determine a right triangle

$$\boxed{\rho \cos(\theta - \alpha) = r} \quad \text{or} \quad x \cos \alpha + y \sin \alpha = r$$

- (r, α) are the parameters of the line



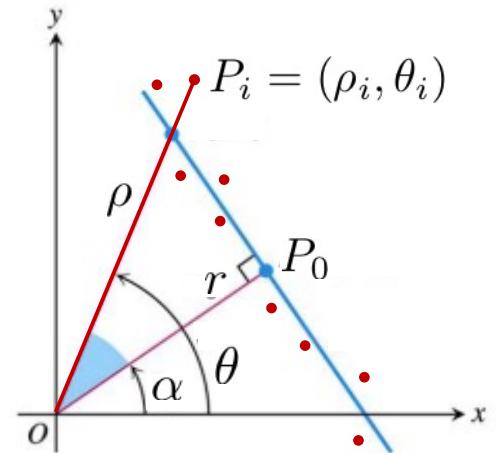
Step #2: line fitting

- Since there is measurement error, the equation of the line is only *approximately* satisfied

$$\rho_i \cos(\theta_i - \alpha) = r + d_i$$

Error

- Assume n ranging measurement points represented in polar coordinates as (ρ_i, θ_i)
- We want to find a line that best “fits” all the measurement points



10/27/22

AA 274A | Lecture 10

Step #2: line fitting

- Consider, first, that all measurements are equally uncertain
- Find line parameters (r, α) that minimize squared error

$$S(r, \alpha) := \sum_{i=1}^n d_i^2 = \sum_{i=1}^n (\rho_i \cos(\theta_i - \alpha) - r)^2$$

- Unweighted least squares

10/27/22

AA 274A | Lecture 10

12

Step #2: line fitting

- Consider, now, the case where each measurement has its own, unique uncertainty
- For example, assume that the variance for each range measurement ρ_i is σ_i
- Associate with each measurement a weight, e.g., $w_i = 1/\sigma_i^2$
- Then, one minimizes

$$S(r, \alpha) := \sum_{i=1}^n w_i d_i^2 = \sum_{i=1}^n w_i (\rho_i \cos(\theta_i - \alpha) - r)^2$$

- Weighted least squares

Step #2: line fitting solution

- Assume that the n ranging measurements are **independent**
- Solution:

$$\alpha = \frac{1}{2} \text{atan2} \left(\frac{\sum_i w_i \rho_i^2 \sin 2\theta_i - \frac{2}{\sum_i w_i} \sum_i \sum_j w_i w_j \rho_i \rho_j \cos \theta_i \sin \theta_j}{\sum_i w_i \rho_i^2 \cos 2\theta_i - \frac{1}{\sum_i w_i} \sum_i \sum_j w_i w_j \rho_i \rho_j \cos(\theta_i + \theta_j)} \right) + \frac{\pi}{2}$$

$$r = \frac{\sum_i w_i \rho_i \cos(\theta_i - \alpha)}{\sum_i w_i}$$

Step #1: line segmentation

- Several algorithms are available
- We will consider three popular algorithms
 1. Split-and-merge
 2. RANSAC
 3. Hough-Transform

Split-and-merge algorithm

- Most popular line extraction algorithm

Data: Set S consisting of all N points, a distance threshold $d > 0$

Result: L , a list of sets of points each resembling a line

$L \leftarrow (S), i \leftarrow 1;$

while $i \leq \text{len}(L)$ **do**

fit a line (r, α) to the set L_i ;

detect the point $P \in L_i$ with the maximum distance D to the line (r, α) ;

if $D < d$ **then**

$| i \leftarrow i + 1$

else

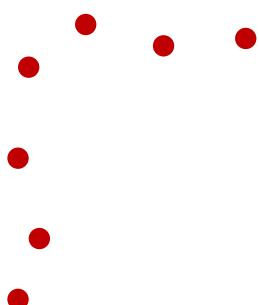
$| \text{split } L_i \text{ at } P \text{ into } S_1 \text{ and } S_2;$

$| L_i \leftarrow S_1; L_{i+1} \leftarrow S_2;$

end

end

Merge collinear sets in L ;



Split-and-merge algorithm

- Most popular line extraction algorithm

Data: Set S consisting of all N points, a distance threshold $d > 0$

Result: L , a list of sets of points each resembling a line

$L \leftarrow (S), i \leftarrow 1;$

while $i \leq \text{len}(L)$ **do**

 fit a line (r, α) to the set L_i ;

 detect the point $P \in L_i$ with the maximum distance D to the line (r, α) ;

if $D < d$ **then**

$i \leftarrow i + 1$

else

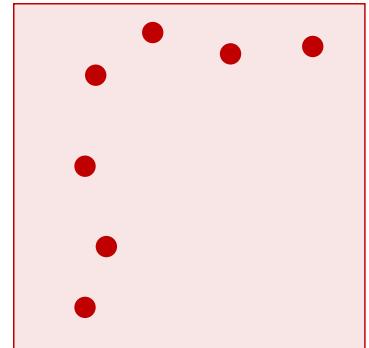
 split L_i at P into S_1 and S_2 ;

$L_i \leftarrow S_1; L_{i+1} \leftarrow S_2;$

end

end

Merge collinear sets in L ;



10/27/22

AA 274A | Lecture 10

16

Split-and-merge algorithm

- Most popular line extraction algorithm

Data: Set S consisting of all N points, a distance threshold $d > 0$

Result: L , a list of sets of points each resembling a line

$L \leftarrow (S), i \leftarrow 1;$

while $i \leq \text{len}(L)$ **do**

 fit a line (r, α) to the set L_i ;

 detect the point $P \in L_i$ with the maximum distance D to the line (r, α) ;

if $D < d$ **then**

$i \leftarrow i + 1$

else

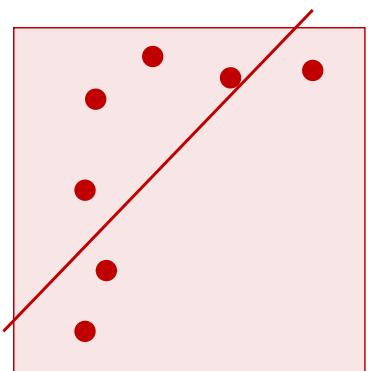
 split L_i at P into S_1 and S_2 ;

$L_i \leftarrow S_1; L_{i+1} \leftarrow S_2;$

end

end

Merge collinear sets in L ;



10/27/22

AA 274A | Lecture 10

16

Split-and-merge algorithm

- Most popular line extraction algorithm

Data: Set S consisting of all N points, a distance threshold $d > 0$

Result: L , a list of sets of points each resembling a line

$L \leftarrow (S)$, $i \leftarrow 1$;

while $i \leq \text{len}(L)$ **do**

 fit a line (r, α) to the set L_i ;

 detect the point $P \in L_i$ with the maximum distance D to the line (r, α) ;

if $D < d$ **then**

$i \leftarrow i + 1$

else

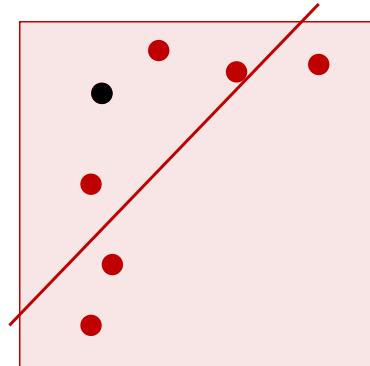
 split L_i at P into S_1 and S_2 ;

$L_i \leftarrow S_1$; $L_{i+1} \leftarrow S_2$;

end

end

Merge collinear sets in L ;



Split-and-merge algorithm

- Most popular line extraction algorithm

Data: Set S consisting of all N points, a distance threshold $d > 0$

Result: L , a list of sets of points each resembling a line

$L \leftarrow (S)$, $i \leftarrow 1$;

while $i \leq \text{len}(L)$ **do**

 fit a line (r, α) to the set L_i ;

 detect the point $P \in L_i$ with the maximum distance D to the line (r, α) ;

if $D < d$ **then**

$i \leftarrow i + 1$

else

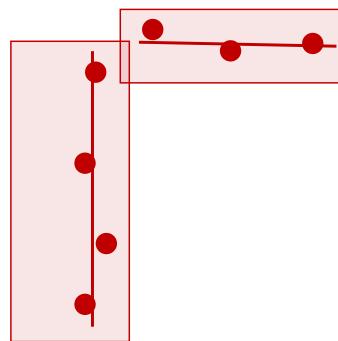
 split L_i at P into S_1 and S_2 ;

$L_i \leftarrow S_1$; $L_{i+1} \leftarrow S_2$;

end

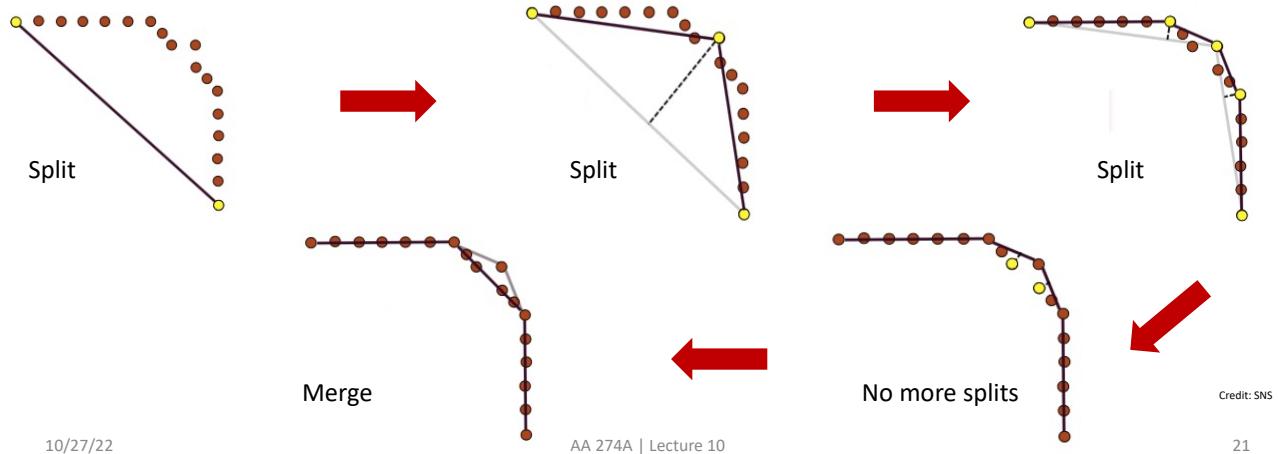
end

Merge collinear sets in L ;



Split-and-merge: iterative-end-point-fit variant

- Iterative-end-point-fit: split-and-merge where the line is constructed by simply connecting the first and last points (as opposed to least squares fit)



10/27/22

AA 274A | Lecture 10

21

RANSAC

- RANSAC: **R**andom **S**ample **C**onsensus
- General method to estimate parameters of a model from a set of observed data in the presence of outliers, where outliers should have no influence on the estimates of the values
- Typical applications in robotics: line extraction from 2D range data, plane extraction from 3D point clouds, feature matching for structure from motion, etc.
- RANSAC is *iterative* and *non-deterministic*: the probability of finding a set free of outliers increases as more iterations are used

10/27/22

AA 274A | Lecture 10

22

RANSAC

Data: Set S consisting of all N points
Result: Set with maximum number of inliers
(and corresponding fitting line)

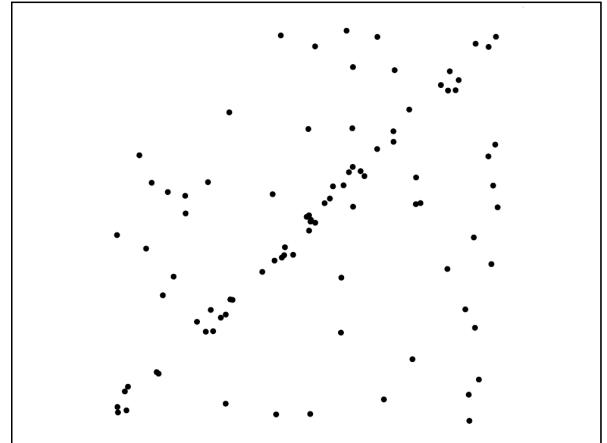
while $i \leq k$ **do**

randomly select 2 points from S ;
fit line l_i through the 2 points;
compute distance of all other points to line l_i ;
construct *inlier* set, i.e., count number of
points with distance to the line less than γ ;
store line l_i and associated set of inliers;

$i \leftarrow i + 1$

end

Choose set with maximum number of inliers



10/27/22

AA 274A | Lecture 10

23

RANSAC

Data: Set S consisting of all N points
Result: Set with maximum number of inliers
(and corresponding fitting line)

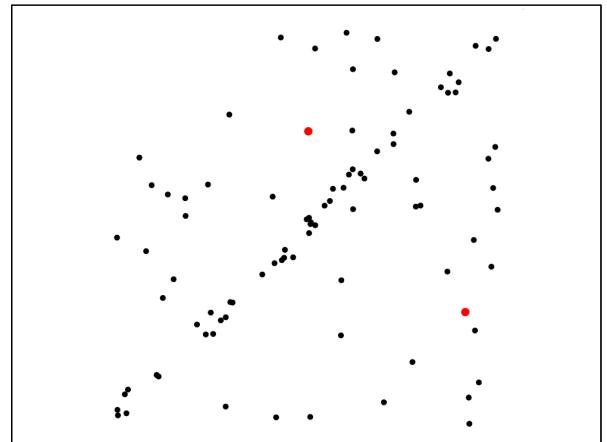
while $i \leq k$ **do**

randomly select 2 points from S ;
fit line l_i through the 2 points;
compute distance of all other points to line l_i ;
construct *inlier* set, i.e., count number of
points with distance to the line less than γ ;
store line l_i and associated set of inliers;

$i \leftarrow i + 1$

end

Choose set with maximum number of inliers



10/27/22

AA 274A | Lecture 10

23

RANSAC

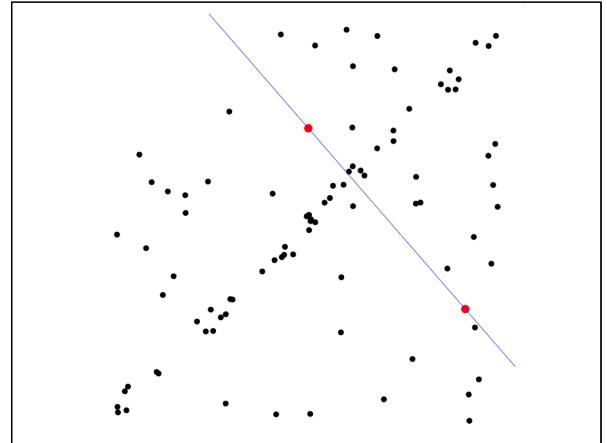
Data: Set S consisting of all N points
Result: Set with maximum number of inliers
(and corresponding fitting line)

while $i \leq k$ **do**

randomly select 2 points from S ;
fit line l_i through the 2 points;
compute distance of all other points to line l_i ;
construct *inlier* set, i.e., count number of
points with distance to the line less than γ ;
store line l_i and associated set of inliers;
 $i \leftarrow i + 1$

end

Choose set with maximum number of inliers



RANSAC

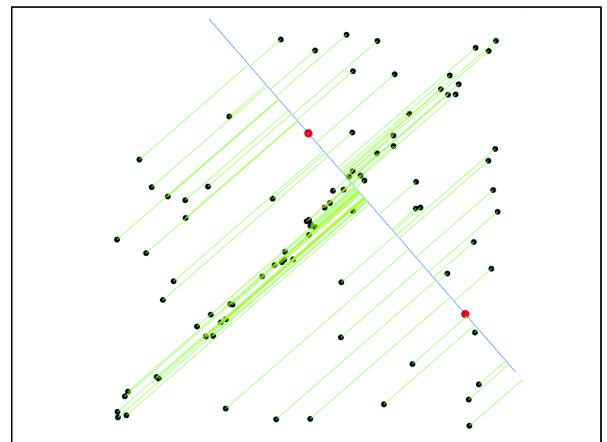
Data: Set S consisting of all N points
Result: Set with maximum number of inliers
(and corresponding fitting line)

while $i \leq k$ **do**

randomly select 2 points from S ;
fit line l_i through the 2 points;
compute distance of all other points to line l_i ;
construct *inlier* set, i.e., count number of
points with distance to the line less than γ ;
store line l_i and associated set of inliers;
 $i \leftarrow i + 1$

end

Choose set with maximum number of inliers

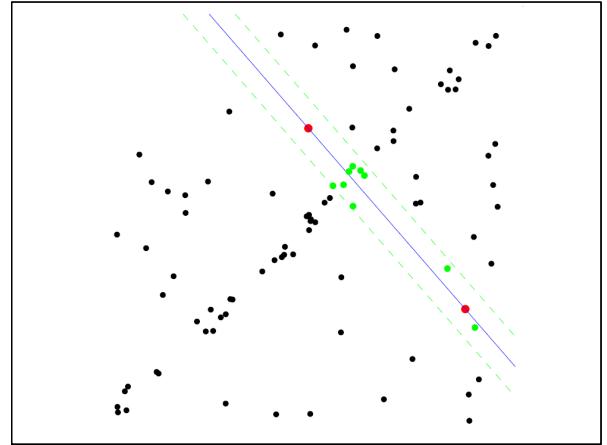


RANSAC

Data: Set S consisting of all N points
Result: Set with maximum number of inliers
(and corresponding fitting line)

```
while  $i \leq k$  do
    randomly select 2 points from  $S$ ;
    fit line  $l_i$  through the 2 points;
    compute distance of all other points to line  $l_i$  ;
    construct inlier set, i.e., count number of
        points with distance to the line less than  $\gamma$ ;
    store line  $l_i$  and associated set of inliers;
     $i \leftarrow i + 1$ 
end
```

Choose set with maximum number of inliers



10/27/22

AA 274A | Lecture 10

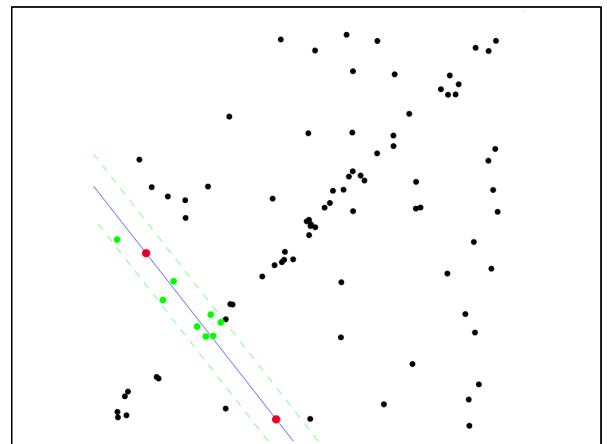
23

RANSAC

Data: Set S consisting of all N points
Result: Set with maximum number of inliers
(and corresponding fitting line)

```
while  $i \leq k$  do
    randomly select 2 points from  $S$ ;
    fit line  $l_i$  through the 2 points;
    compute distance of all other points to line  $l_i$  ;
    construct inlier set, i.e., count number of
        points with distance to the line less than  $\gamma$ ;
    store line  $l_i$  and associated set of inliers;
     $i \leftarrow i + 1$ 
end
```

Choose set with maximum number of inliers



10/27/22

AA 274A | Lecture 10

23

RANSAC

Data: Set S consisting of all N points
Result: Set with maximum number of inliers
(and corresponding fitting line)

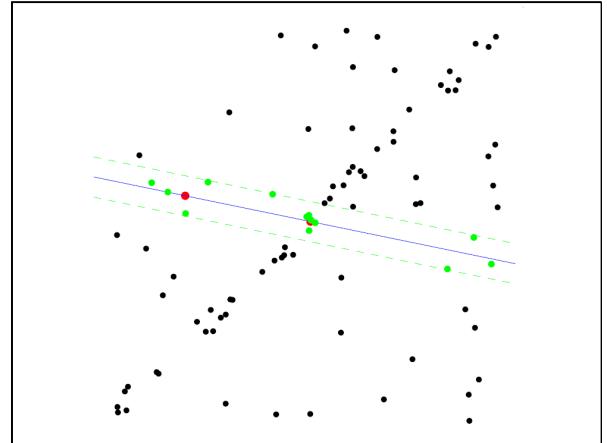
while $i \leq k$ **do**

randomly select 2 points from S ;
fit line l_i through the 2 points;
compute distance of all other points to line l_i ;
construct *inlier* set, i.e., count number of
points with distance to the line less than γ ;
store line l_i and associated set of inliers;

$i \leftarrow i + 1$

end

Choose set with maximum number of inliers



10/27/22

AA 274A | Lecture 10

23

RANSAC

Data: Set S consisting of all N points
Result: Set with maximum number of inliers
(and corresponding fitting line)

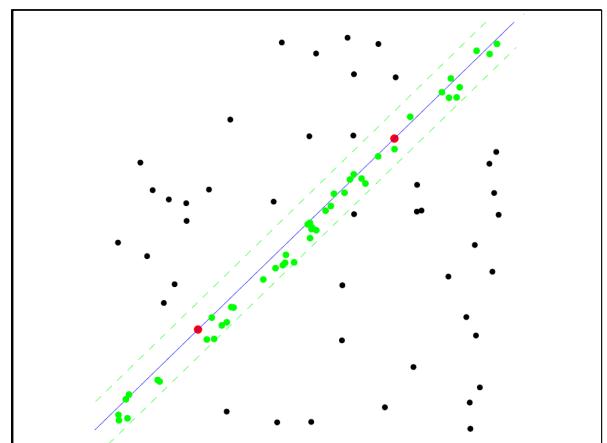
while $i \leq k$ **do**

randomly select 2 points from S ;
fit line l_i through the 2 points;
compute distance of all other points to line l_i ;
construct *inlier* set, i.e., count number of
points with distance to the line less than γ ;
store line l_i and associated set of inliers;

$i \leftarrow i + 1$

end

Choose set with maximum number of inliers



10/27/22

AA 274A | Lecture 10

23

RANSAC iterations

- In principle, one would need to check all possible combinations of 2 points in dataset
- If $|S| = N$, number of combinations is $\frac{N(N-1)}{2} \rightarrow$ too many
- However, if we have a rough estimate of the percentage of inliers, we do not need to check all combinations...

RANSAC iterations: statistical characterization

- Let w be the percentage of inliers in the dataset, i.e.,

$$w = \frac{\text{number of inliers}}{N}$$

- Let p be the desired probability of finding a set of points free of outliers (typically, $p = 0.99$)
- Assumption: 2 points chosen for line estimation are selected independently
 - $P(\text{both points selected are inliers}) = w^2$
 - $P(\text{at least one of the selected points is an outlier}) = 1 - w^2$
 - $P(\text{RANSAC never selects two points that are both inliers}) = (1 - w^2)^k$

RANSAC iterations: statistical characterization

- Then minimum number of iterations \bar{k} to find an outlier-free set with probability at least p is:

$$1 - p = (1 - w^2)^{\bar{k}} \Rightarrow \bar{k} = \frac{\log(1 - p)}{\log(1 - w^2)}$$

- Thus if we know w (at least approximately), after \bar{k} iterations RANSAC will find a set free of outliers with probability p
- Note:
 - \bar{k} depends only on w , not on N !
 - More advanced versions of RANSAC estimate w adaptively

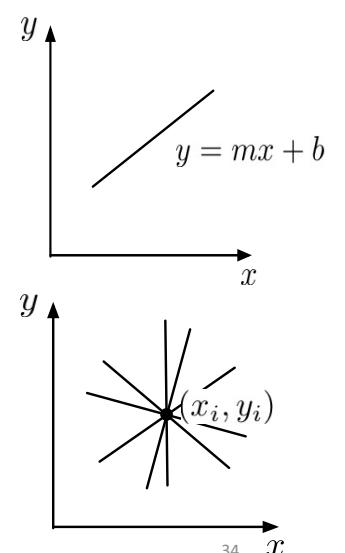
Hough transform

- **Key idea:** each point votes for a set of plausible line parameters

- A line has two parameters: (m, b)

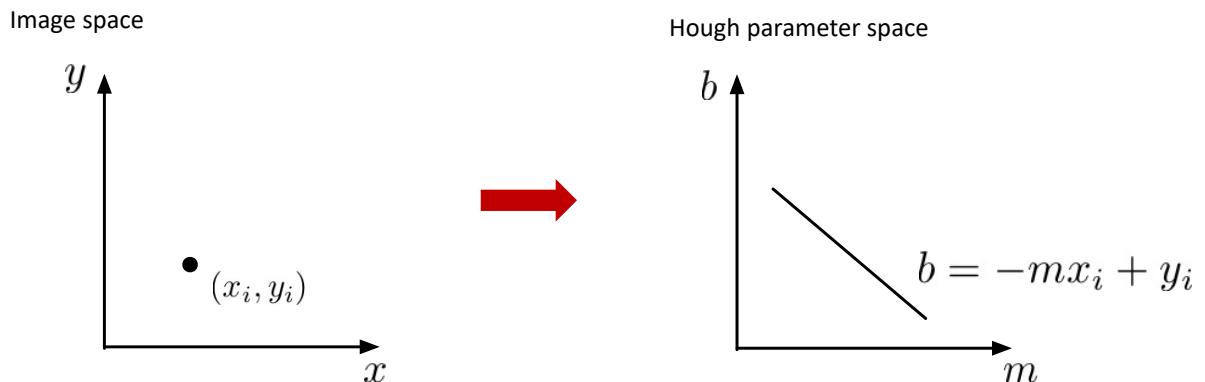
- Given a point (x_i, y_i) , the lines that could pass through this point are all (m, b) satisfying

$$y_i = mx_i + b, \quad \text{or } b = -mx_i + y_i$$



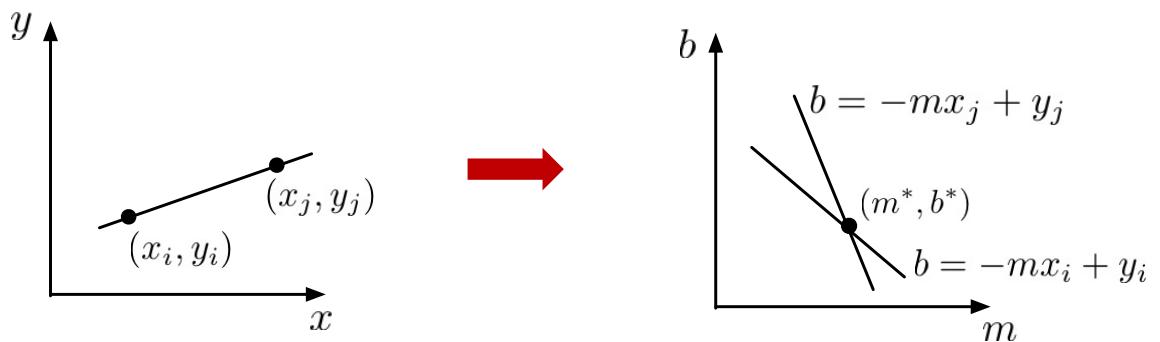
Hough transform

- A point in image space maps into a line in *Hough space*



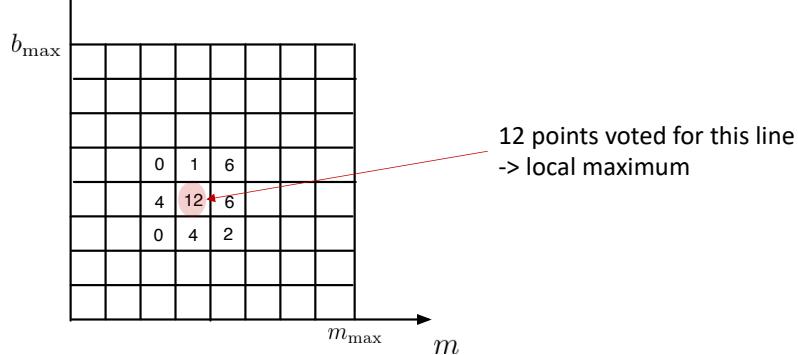
Hough transform

- **Key fact:** all points on a line in image space yield lines in parameter space which intersect at a *common point*, (m^*, b^*)



Hough transform algorithm

1. Initialize an accumulator array $H(m, b)$ to zero
2. For each point (x_i, y_i) , increment all cells that satisfy $b = -x_i m + y_i$
3. Local maxima in array $H(m, b)$ corresponds to lines



10/27/22

AA 274A | Lecture 10

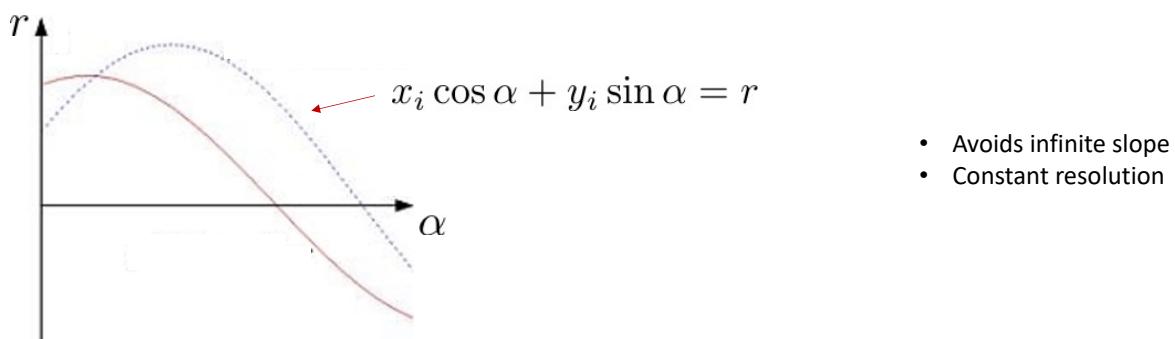
37

Hough transform algorithm: polar coordinate representation

- Equation of a line in polar coordinates

$$x \cos \alpha + y \sin \alpha = r$$

- The parameter space transform of a point is a sinusoidal curve



10/27/22

AA 274A | Lecture 10

38

Hough transform algorithm, revised

Data: Set S containing N points

Result: Line fitting the points in S

Initialize $n_\alpha \times n_r$ accumulator H with zeros;

foreach $(x_i, y_i) \in S$ **do**

foreach $\alpha \in \{\alpha_1, \dots, \alpha_{n_\alpha}\}$ **do**

 compute $r = x_i \cos \alpha + y_i \sin \alpha$;

$H[\alpha, r] \leftarrow H[\alpha, r] + 1$;

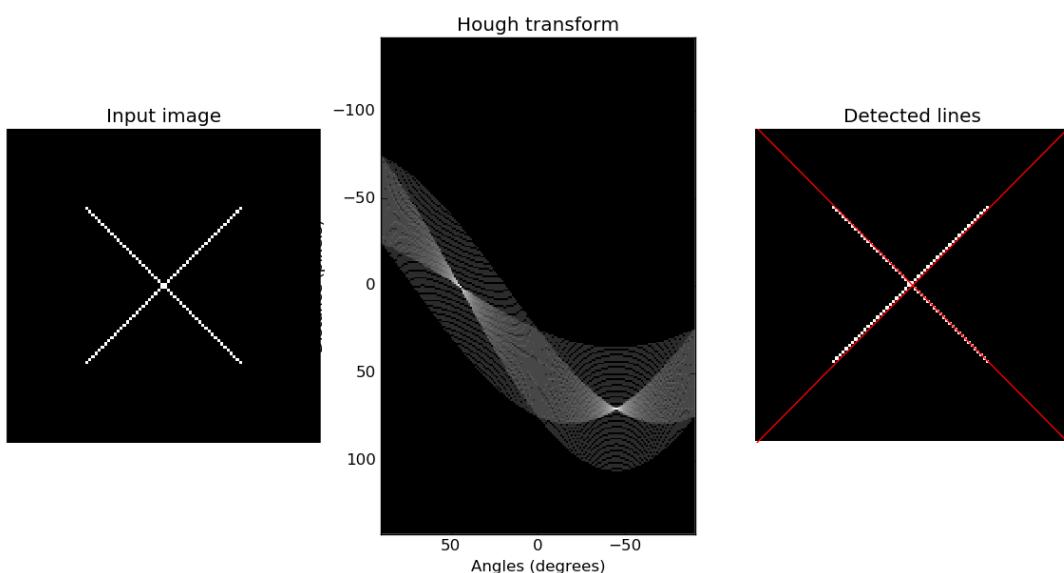
end

end

Choose (α^*, r^*) that corresponds to largest count in H ;

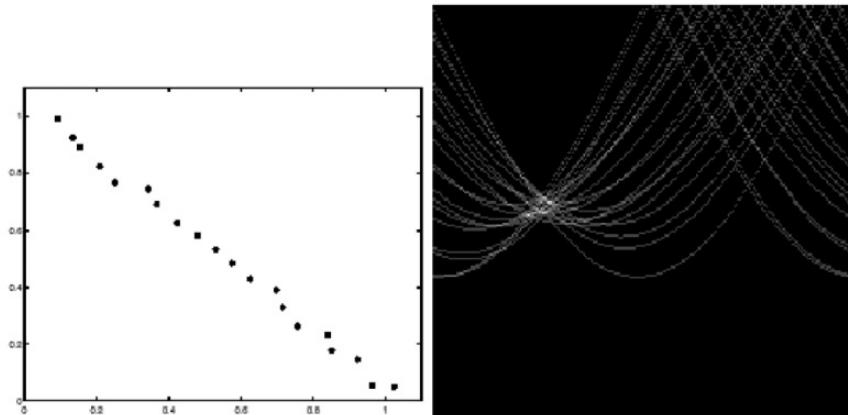
Return line defined by (α^*, r^*)

Hough transform: example



Hough transform: example

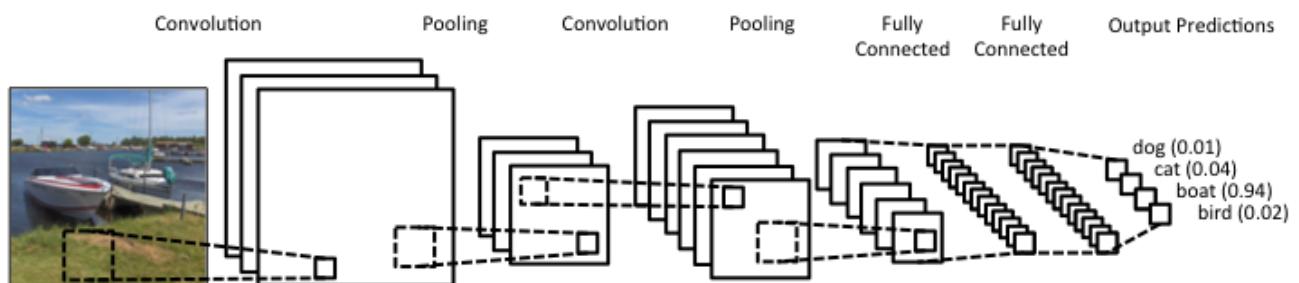
- With noise, peaks may be hard to detect



Object recognition

- Object recognition: capability of naming discrete objects in the world
- Why is it hard? Many reasons, including:
 - Real world is made of a jumble of objects, which all occlude one another and appear in different poses
 - There is a lot of variability intrinsic within each class (e.g., dogs)
- In this class, we will look at three methods:
 - Template matching
 - Bag of visual words
 - Neural network methods (treated as a black box, take AA274B for details)

Standard paradigm - CNNs for recognition



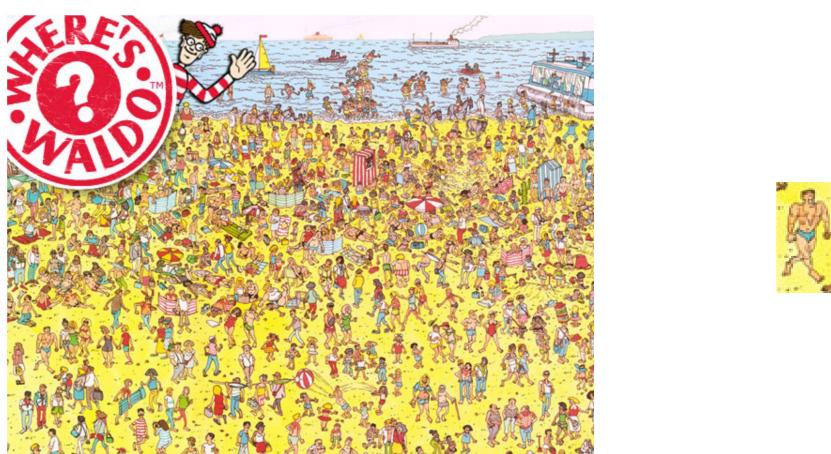
10/21/21

AA 274A | Lecture 10

55

Template matching

- How can we find Waldo?



Source: Sanja Fidler

10/21/21

AA 274A | Lecture 10

43

Template matching

- Slide and compare!

Image I



Filter F

Source: Sanja Fidler

10/21/21

AA 274A | Lecture 10

44

Template matching

- In practice, remember correlation:

$$I'(x, y) = F \circ I = \sum_{i=-N}^N \sum_{j=-M}^M F(i, j) I(x + i, y + j)$$

Vector representation of filter

- One can equivalently write: $I'(x, y) = \mathbf{f}^T \cdot \mathbf{t}_{ij}$

Vector representation of neighborhood patch

- To ensure that perfect matching yields one, we consider *normalized* correlation, that is

$$I'(x, y) = \frac{\mathbf{f}^T \cdot \mathbf{t}_{ij}}{\|\mathbf{f}\| \|\mathbf{t}_{ij}\|}$$

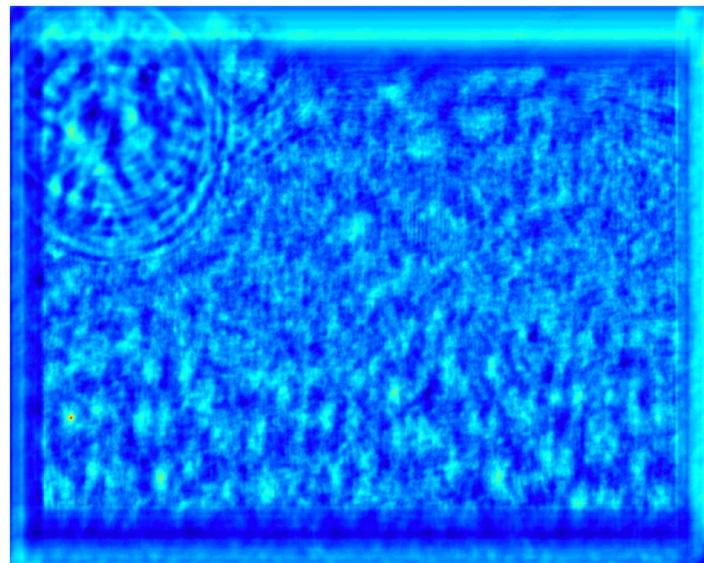
10/21/21

AA 274A | Lecture 10

45

Template matching

Result:



Source: Sanja Fidler

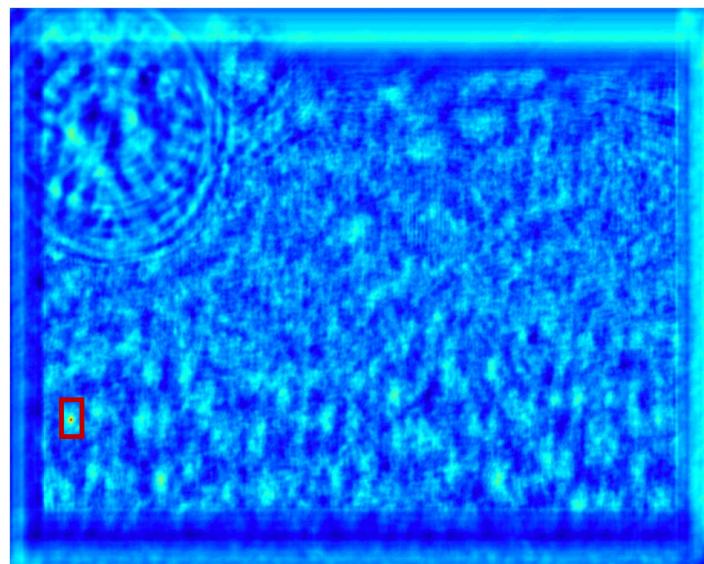
10/21/21

AA 274A | Lecture 10

46

Template matching

Result:



Source: Sanja Fidler

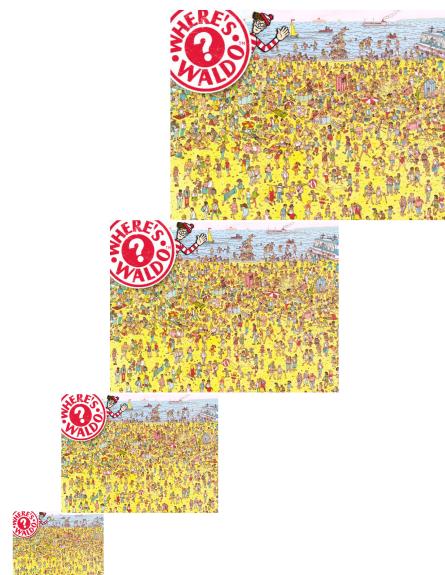
10/21/21

AA 274A | Lecture 10

46

Template matching

- Problem: what if the object in the image is much larger or much smaller than our template?
- Solution: re-scale the image multiple times and do correlation on every size!
- This leads to the idea of *image pyramids*



10/21/21

AA 274A | Lecture 10

47

Image pyramids: scaling down

- Naïve solution: keep only some rows and columns
- E.g.: keep every other column to reduce image by 1/2 in width direction



Source:
Sanja Fidler

10/21/21

AA 274A | Lecture 10

48

Image pyramids: scaling down

- Naïve solution: keep only some rows and columns
- E.g.: keep every other column to reduce image by 1/2 in width direction



Source:
Sanja Fidler

10/21/21

AA 274A | Lecture 10

49

Image pyramids: scaling down

- Solution: blur the image via Gaussian, *then* subsample
- Intuition: remove high frequency content in the image



Source:
Sanja Fidler

10/21/21

AA 274A | Lecture 10

50

Image pyramids: scaling down

- Solution: blur the image via Gaussian, *then* subsample
- Intuition: remove high frequency content in the image



Source:
Sanja Fidler

10/21/21

AA 274A | Lecture 10

51

Image pyramids: scaling down

- Solution: blur the image via Gaussian, *then* subsample
- Intuition: remove high frequency content in the image



Source:
Sanja Fidler

10/21/21

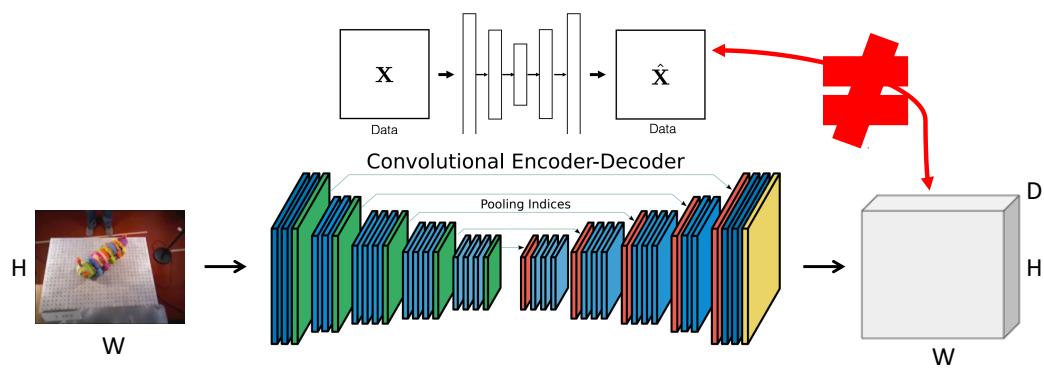
AA 274A | Lecture 10

52

Image pyramids

- A sequence of images created with Gaussian blurring and down-sampling is called a Gaussian pyramid
- The other step is to perform up-sampling (nearest neighbor, bilinear, bicubic, etc.)

Dense ObjectNets - Architecture

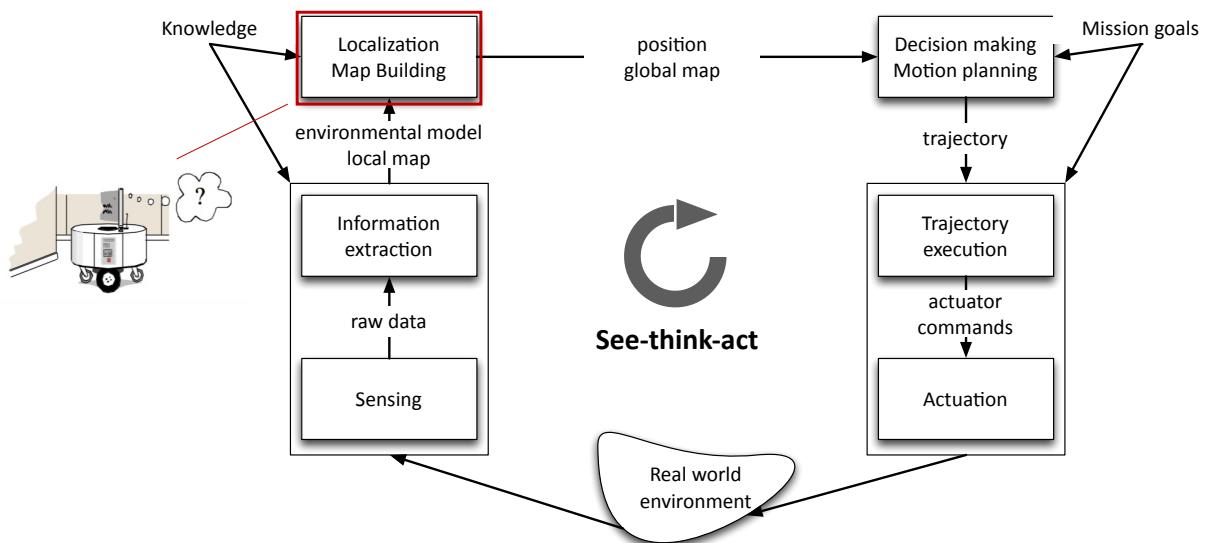


Principles of Robot Autonomy I

Introduction to localization and filtering theory



Module 3



Today's lecture

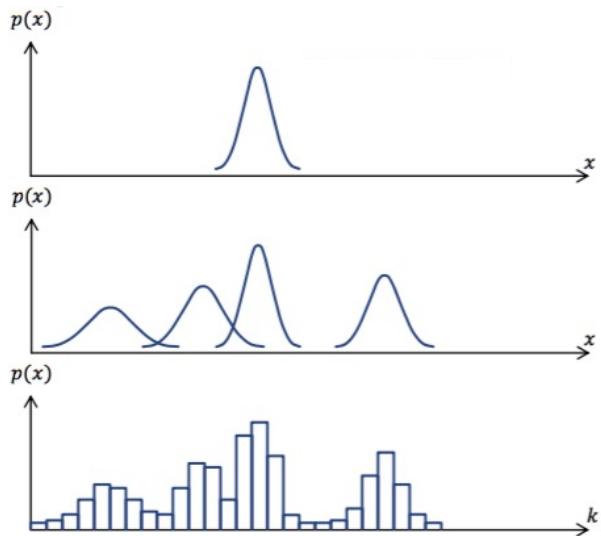
- Aim
 - Learn basic concepts about Bayesian filtering
- Readings
 - S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. MIT press, 2005. Chapter 2

Localization

- Two main approaches:
 1. **Behavioral approach**: design a set of behaviors that together result in the desired robot motion (no need for a map)
 2. **Map-based approaches**: robot *explicitly* attempts to localize by collecting sensor data, then updating belief about its position with respect to a map
- We will focus on **map-based approaches**; two main aspects:
 - Map representation: how to represent the environment?
 - Belief representation: how to model the belief regarding the position within the map?

Probabilistic map-based localization

- Key idea: represent belief as a probability distribution
 1. Encodes sense of position
 2. Maintains notion of robot's uncertainty
- Belief representation:
 1. Single-hypothesis vs. multiple hypothesis
 2. Continuous vs. discretized
- Today we will overview basic concepts in Bayesian filtering



Basic concepts in probability

- Key idea: quantities such as sensor measurements, states of a robot, and its environment are modeled as random variables (RVs)
- Discrete RV: the space of all the values that a random variable X can take on is *discrete*; characterized by probability mass function (pmf)

$$p(X = x) \quad (\text{or } p(x)), \quad \sum_x p(X = x) = 1$$

Random variable Specific value

- Continuous RV: the space of all the values that a random variable X can take on is *continuous*; characterized by probability density function (pdf)

$$P(a \leq X \leq b) = \int_a^b p(x) dx, \quad \int_{-\infty}^{\infty} p(x) dx = 1$$

Joint distribution, independence, and conditioning

- Joint distribution of two random variables X and Y is denoted as

$$p(x, y) := p(X = x \text{ and } Y = y)$$

- If X and Y are independent

$$p(x, y) = p(x)p(y)$$

- Suppose we know that $Y = y$ (with $p(y) > 0$); conditioned on this fact, the probability that the X 's value is x is given by

$$p(x | y) := \frac{p(x, y)}{p(y)}$$

Note: if X and Y are independent

$$p(x | y) := p(x)!$$

Conditional probability

Law of total probability

- For discrete RVs:

$$p(x) = \sum_y p(x, y) = \sum_y p(x | y)p(y)$$

- For continuous RVs:

$$p(x) = \int p(x, y)dy = \int p(x | y)p(y)dy$$

- Note: if $p(y) = 0$, define the product $p(x | y)p(y) = 0$

Bayes' rule

- Key relation between $p(x | y)$ and its “inverse,” $p(y | x)$
- For discrete RVs:

$$p(x | y) = \frac{p(y | x)p(x)}{p(y)} = \frac{p(y | x)p(x)}{\sum_{x'} p(y | x')p(x')}$$

- For continuous RVs:

$$p(x | y) = \frac{p(y | x)p(x)}{p(y)} = \frac{p(y | x)p(x)}{\int p(y | x')p(x') dx'}$$

Bayes' rule and probabilistic inference

- Assume x is a quantity we would like to infer from y
- Bayes rule allows us to do so through the inverse probability, which specifies the probability of data y assuming that x was the cause

Posterior probability distribution

Prior probability distribution

$$p(x | y) = \frac{p(y | x)p(x)}{\int p(y | x')p(x') dx'}$$

Normalizer, does not depend on $x := \eta^{-1}$

- Notational simplification

$$p(x | y) = \eta p(y | x)p(x)$$

More on Bayes' rule and independence

- Extension of Bayes rule: conditioning Bayes rule on $Z=z$ gives

$$p(x | y, z) = \frac{p(y | x, z)p(x | z)}{p(y | z)}$$

- Extension of independence: *conditional independence*

$$p(x, y | z) = p(x | z)p(y | z), \quad \text{equivalent to} \quad \begin{cases} p(x | z) = p(x | z, y) \\ p(y | z) = p(y | z, x) \end{cases}$$

- Note: in general

$$p(x, y | z) = p(x | z)p(y | z) \Rightarrow p(x, y) = p(x)p(y)$$

$$p(x, y) = p(x)p(y) \not\Rightarrow p(x, y | z) = p(x | z)p(y | z)$$

Expectation of a RV

- Expectation for discrete RVs: $E[X] = \sum_x x p(x)$
- Expectation for continuous RVs: $E[X] = \int x p(x) dx$
- Expectation is a linear operator: $E[aX + b] = a E[X] + b$
- Expectation of a vector of RVs is simply the vector of expectations
- Covariance

$$\text{cov}(X, Y) = E[(X - E[X])(Y - E[Y])^T] = E[XY^T] - E[X]E[Y]^T$$

Model for robot-environment interaction

- Two fundamental types of robot-environment interactions: the robot can influence **the state** of its environment through **control actions**, and gather information about the **state** through **measurements**
- **State x_t** : collection at time t of all aspects of the robot and its environment that can impact the future
 - Robot pose (e.g., robot location and orientation)
 - Robot velocity
 - Locations and features of surrounding objects in the environment, etc.
- Useful notation: $x_{t_1:t_2} := x_{t_1}, x_{t_1+1}, x_{t_1+2}, \dots, x_{t_2}$
- A state x_t is called *complete* if no variables prior to x_t can influence the evolution of future states → **Markov property**

Measurement and control data

- **Measurement data z_t** : information about state of the environment at time t ; useful notation

$$z_{t_1:t_2} := z_{t_1}, z_{t_1+1}, z_{t_1+2}, \dots, z_{t_2}$$

- **Control data u_t** : information about the change of state at time t ; useful notation

$$u_{t_1:t_2} := u_{t_1}, u_{t_1+1}, u_{t_1+2}, \dots, u_{t_2}$$

- Key difference: measurement data tends to increase robot's knowledge, while control actions tend to induce a loss of knowledge

State equation

- General probabilistic generative model

$$p(x_t | x_{0:t-1}, z_{1:t-1}, u_{1:t})$$

Convention: first take control action and then take measurement

- Key assumption: state is complete (i.e., the Markov property holds)

$$p(x_t | x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t | x_{t-1}, u_t)$$

State transition probability

- In other words, we assume *conditional independence*, with respect to conditioning on x_{t-1}

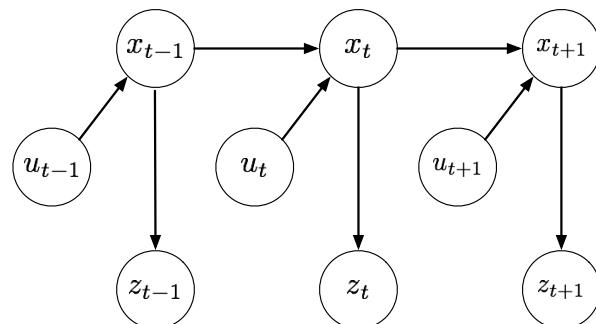
Measurement equation and overall stochastic model

- Assuming x_t is complete

$$p(z_t | x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t | x_t)$$

Measurement probability

- Overall dynamic Bayes network model (also referred to as hidden Markov model)



Belief distribution

- **Belief distribution:** reflects internal knowledge about the state
 - A belief distribution assigns a probability to each possible hypothesis about the true state
 - Formally, belief distributions are posterior probabilities over state variables conditioned on the available data

$$bel(x_t) := p(x_t \mid z_{1:t}, u_{1:t})$$

- Similarly, the *prediction* distribution is defined as

$$\overline{bel}(x_t) := p(x_t \mid \textcolor{red}{z_{1:t-1}}, u_{1:t})$$

- Calculating $bel(x_t)$ from $\overline{bel}(x_t)$ is called correction or measurement update

Bayes filter algorithm

- Bayes' filter algorithm: most general algorithm for calculating beliefs
 - Key assumption: state is complete

- Recursive algorithm
 - Step 1 (prediction):
compute $bel(x_t)$
 - Step 2 (measurement update):
compute $bel(x_t)$
 - Algorithm initialized with $bel(x_0)$
(e.g., uniform or points mass)

```

Data:  $bel(x_{t-1}), u_t, z_t$  ↑
Result:  $bel(x_t)$ 
foreach  $x_t$  do
     $\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1};$ 
     $bel(x_t) = \eta p(z_t | x_t) \overline{bel}(x_t);$ 
end
Return  $bel(x_t)$ 

```

Derivation: measurement update

$$\begin{aligned}
 bel(x_t) &= p(x_t | z_{1:t}, u_{1:t}) \\
 &= \frac{p(z_t | x_t, z_{1:t-1}, u_{1:t}) p(x_t | z_{1:t-1}, u_{1:t})}{\underbrace{p(z_t | z_{1:t-1}, u_{1:t})}_{:=\eta^{-1}}} && \text{Bayes rule} \\
 &= \eta p(z_t | x_t) \underbrace{p(x_t | z_{1:t-1}, u_{1:t})}_{=\overline{bel}(x_t)} && \text{Markov property}
 \end{aligned}$$

Derivation: correction update

$$\begin{aligned}
 \overline{bel}(x_t) &= p(x_t | z_{1:t-1}, u_{1:t}) \\
 &= \int p(x_t | x_{t-1}, z_{1:t-1}, u_{1:t}) p(x_{t-1} | z_{1:t-1}, u_{1:t}) dx_{t-1} && \text{Total probability} \\
 &= \int p(x_t | x_{t-1}, u_t) p(x_{t-1} | z_{1:t-1}, u_{1:t}) dx_{t-1} && \text{Markov} \\
 &= \int p(x_t | x_{t-1}, u_t) p(x_{t-1} | z_{1:t-1}, \textcolor{red}{u_{1:t-1}}) dx_{t-1} && \text{For general output feedback policies, } u_t \text{ does not provide additional information on } x_{t-1} \\
 &= \int p(x_t | x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1}
 \end{aligned}$$

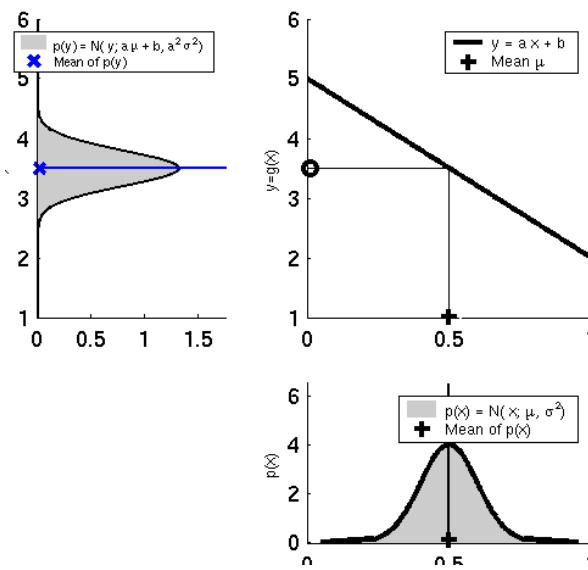
Discrete Bayes' filter

- **Discrete Bayes' filter algorithm:** applies to problems with *finite state spaces*

- Belief $bel(x_t)$ represented as pmf $\{p_{k,t}\}$

Data: $\{p_{k,t-1}\}, u_t, z_t$
Result: $\{p_{k,t}\}$
foreach k **do**
 $\bar{p}_{k,t} = \sum_i p(X_t = x_k | u_t, X_{t-1} = x_i) p_{i,t-1};$
 $p_{k,t} = \eta p(z_t | X_t = x_k) \bar{p}_{k,t};$
end
Return $\{p_{k,t}\}$

Next time



Principles of Robot Autonomy I

Parametric filtering

(Kalman Filter, Extended Kalman Filter, Unscented Kalman Filter)



Today's lecture

- Aim
 - Learn about parametric filters
- Readings
 - S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. MIT press, 2005.
Sections 3.1 – 3.4, 4.1, 4.3, 7.1

Instantiating the Bayes' filter

- Tractable implementations of Bayes' filter exploit structure and / or approximations; two main classes
 - Parametric filters: e.g., **KF**, **EKF**, UKF, etc.
 - Non parametric filters: e.g., **histogram filter**, **particle filter**, etc.

11/03/22

AA 274 | Lecture 12

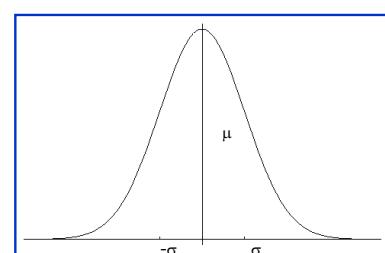
3

Gaussian distributions

- **Key idea:** belief represented as multivariate normal distribution

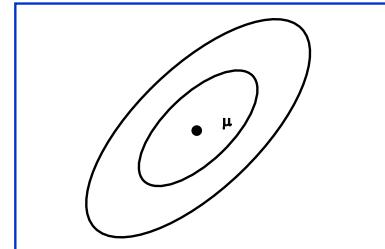
Univariate

$$p(x) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$$
$$\sim \mathcal{N}(x; \mu, \sigma^2)$$



Multivariate

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right)$$
$$\sim \mathcal{N}(\mu, \Sigma)$$



11/03/22

AA 274 | Lecture 12

4

Key properties of Gaussian random variables

- If $X \sim \mathcal{N}(\mu, \Sigma)$ then

$$Y = AX + b \sim \mathcal{N}(A\mu + b, A\Sigma A^T)$$

- The sum of two independent Gaussian RVs

$$X_i \sim \mathcal{N}(\mu_i, \Sigma_i), \quad i = 1, 2$$

is Gaussian, specifically

$$X_1 + X_2 \sim \mathcal{N}(\mu_1 + \mu_2, \Sigma_1 + \Sigma_2)$$

- The product of Gaussian pdf is also Gaussian

Kalman filter (KF)

- Assumption #1: linear dynamics

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- Independent process noise ϵ_t is $\mathcal{N}(0, R_t)$
- Assumption #1 implies that the probabilistic generative model is Gaussian

$$p(x_t | u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}} \exp \left(-\frac{1}{2} (x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1} (x_t - A_t x_{t-1} - B_t u_t) \right)$$

Kalman filter (KF)

- Assumption #2: linear measurement model

$$z_t = C_t x_t + \delta_t$$

- Independent measurement noise δ_t is $\mathcal{N}(0, Q_t)$
- Assumption #2 implies that the measurement probability is Gaussian

$$p(z_t | x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1} (z_t - C_t x_t)\right)$$

Kalman filter (KF)

- Assumption #3: the initial belief is Gaussian

$$bel(x_0) = p(x_0) = \det(2\pi \Sigma_0)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1} (x_0 - \mu_0)\right)$$

- **Key fact:** These three assumptions ensure that the posterior $bel(x_t)$ is Gaussian for all t , i.e., $bel(x_t) = \mathcal{N}(\mu_t, \Sigma_t)$
- Note:
 - KF implements belief computation for continuous states
 - Gaussians are unimodal -> commitment to single-hypothesis filtering

Recap – Bayes Filter

Data: $bel(x_{t-1}), u_t, z_t$

Result: $bel(x_t)$

foreach x_t **do**

$$\begin{cases} \bar{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}; \\ bel(x_t) = \eta p(z_t | x_t) \bar{bel}(x_t); \end{cases}$$

end

Return $bel(x_t)$

Kalman filter: algorithm

Prediction

Project state ahead

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

Project covariance ahead

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

Correction

Compute Kalman gain

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

Update estimate with new measurement

$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$

Update covariance

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

Data: $\overbrace{bel(x_{t-1})}^{bel(x_{t-1})}, (\mu_{t-1}, \Sigma_{t-1}), u_t, z_t$

Result: (μ_t, Σ_t)

Prediction:
 $bel(x_t)$

$$\begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t; \\ \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t; \end{cases}$$

Correction:
 $bel(x_t)$

$$\begin{cases} K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}; \\ \mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t); \\ \Sigma_t = (I - K_t C_t) \bar{\Sigma}_t; \end{cases}$$

Return $\overbrace{(\mu_t, \Sigma_t)}^{bel(x_t)}$

Kalman filter: derivation (sketch)

- Prediction

$$\overline{bel}(x_t) = \int p(x_t | x_{t-1}, u_t) \cdot bel(x_{t-1}) dx_{t-1}$$

$$\mathcal{N}(A_t x_{t-1} + B_t u_t, R_t) \quad \downarrow \quad \mathcal{N}(\mu_{t-1}, \Sigma_{t-1})$$

- Recalling that $x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$

$$\overline{bel}(x_t) = \mathcal{N}(\bar{\mu}_t, \bar{\Sigma}_t) \quad \text{with} \quad \begin{aligned} \bar{\mu}_t &= A_t \mu_{t-1} + B_t u_t \\ \bar{\Sigma}_t &= A_t \Sigma_{t-1} A_t^T + R_t \end{aligned}$$

Kalman filter: derivation (sketch)

- Correction

$$bel(x_t) = \eta p(z_t | x_t) \cdot \overline{bel}(x_t)$$

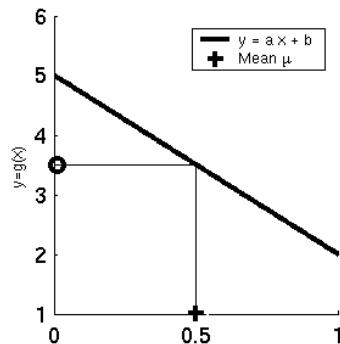
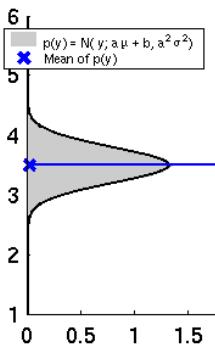
$$\mathcal{N}(C_t x_t, Q_t) \quad \downarrow \quad \mathcal{N}(\bar{\mu}_t, \bar{\Sigma}_t)$$

- After some algebraic manipulations

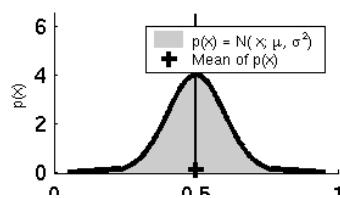
$$bel(x_t) = \mathcal{N}(\mu_t, \Sigma_t) \quad \text{with} \quad \begin{aligned} K_t &= \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1} \\ \mu_t &= \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t) \\ \Sigma_t &= (I - K_t C_t) \bar{\Sigma}_t \end{aligned}$$

- Other derivations are possible; see, e.g., R. E. Kalman, A new approach to linear filtering and prediction problems. Journal of Basic Engineering, 82(1), 35-45, 1960.

Revisiting linearity assumption



- KF crucially exploits the property that a linear transformation of a Gaussian RV results in a Gaussian RV
- However, linearity assumptions are severely restrictive for robotics applications



11/03/22

AA 274 | Lecture 12

13

Extended Kalman filter (EKF)

- **Goal:** relax the linearity assumption
- The dynamics are now given by

$$x_t = g(u_t, x_{t-1}) + \epsilon_t$$

- And the measurement model is now given by

$$z_t = h(x_t) + \delta_t$$

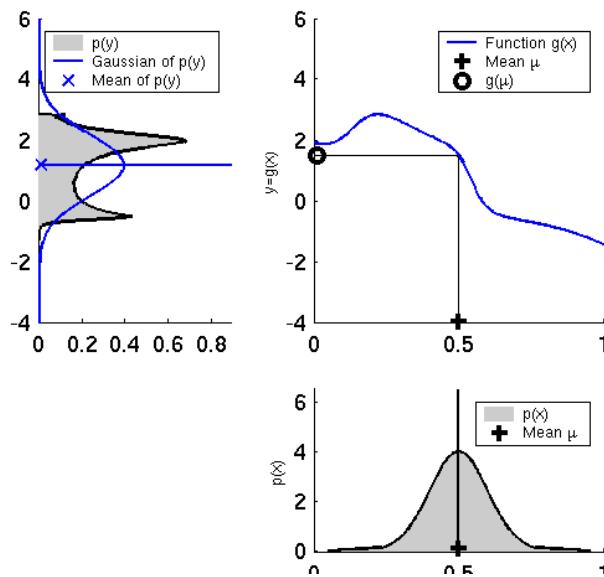
- Key idea: shift focus from computing exact posterior to efficiently compute a Gaussian approximation

11/03/22

AA 274 | Lecture 12

14

Goal of EKF



11/03/22

AA 274 | Lecture 12

15

EKF: key idea

- **Key idea:** linearize g and h around the most likely state and transform beliefs according to such linear approximations
- For the dynamics equation

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \underbrace{J_g(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1})}_{\text{Jacobian of } g} := G_t$$

- Accordingly

$$p(x_t | u_t, x_{t-1}) = \det(2\pi R_t)^{-1/2} \exp\left(-\frac{1}{2}[x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T R_t^{-1} [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]\right)$$

11/03/22

AA 274 | Lecture 12

16

EKF: key idea

- For the measurement model

$$h(x_t) \approx h(\bar{\mu}_t) + \underbrace{J_h(\bar{\mu}_t)}_{:=H_t}(x_t - \bar{\mu}_t)$$

- Accordingly,

$$p(z_t | x_t) = \det(2\pi Q_t)^{-1/2} \exp\left(-\frac{1}{2}[z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t)]Q_t^{-1}[z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t)]\right)$$

EKF: algorithm

- Main differences:

1. Linear predictions are replaced by their nonlinear generalizations
2. EKF uses Jacobians instead of linear system matrices
3. Mathematical derivation of EKF parallels that of KF

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

Result: (μ_t, Σ_t)

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

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

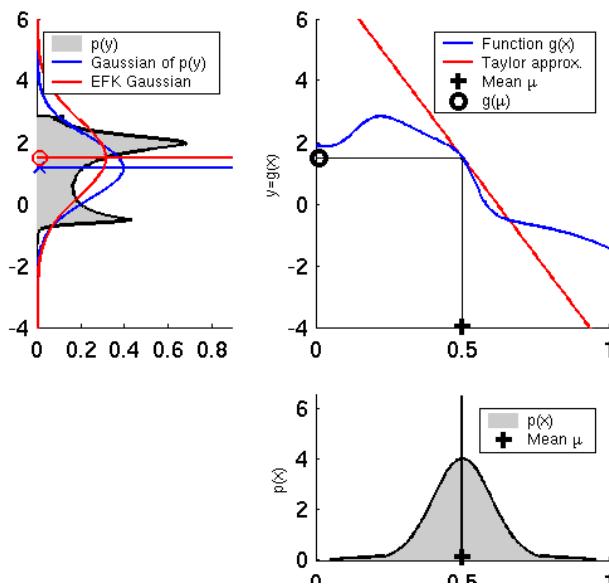
$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$;

$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$;

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

Return (μ_t, Σ_t)

EKF: examples

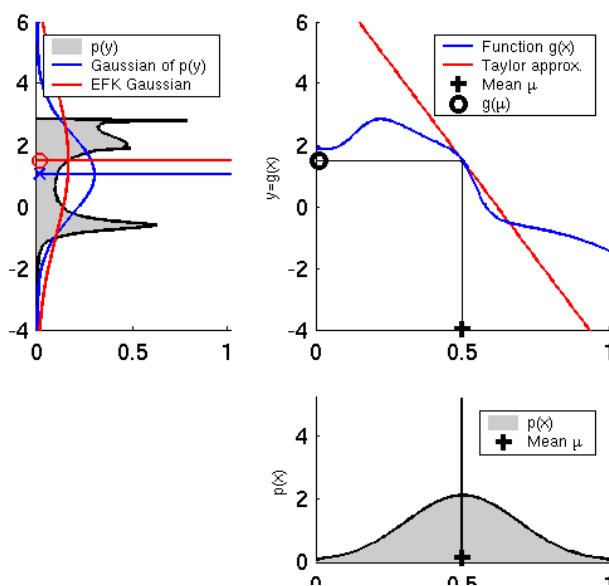


11/03/22

AA 274 | Lecture 12

19

EKF: examples

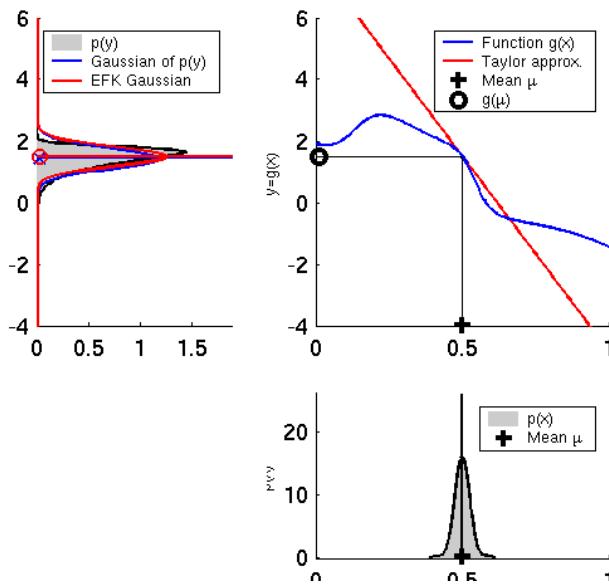


11/03/22

AA 274 | Lecture 12

20

EKF: examples



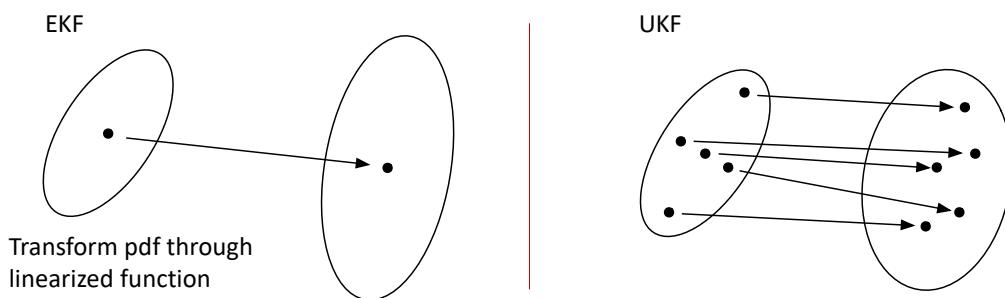
11/03/22

AA 274 | Lecture 12

21

Unscented Kalman filter (UKF) – basic idea

- Taylor series expansion applied by EKF is not the only way to approximate the transformation of a Gaussian; other approaches
 - Assumed density filter
 - **Unscented Kalman filter (UKF)**



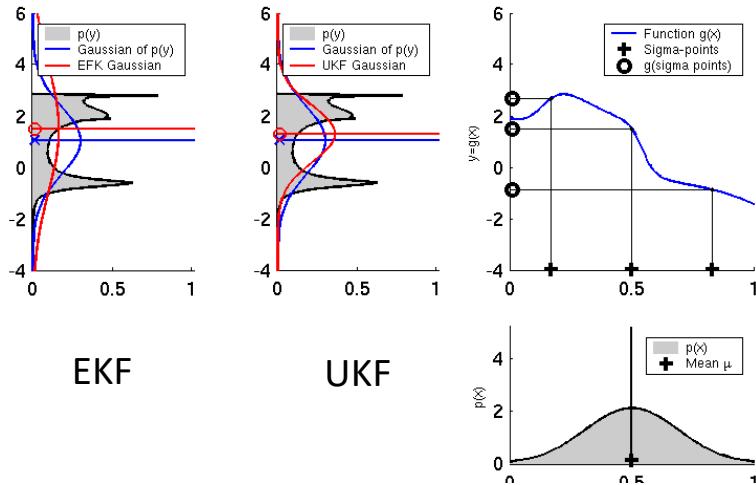
1. Compute sigma-points
2. Transform each sigma point through nonlinear function
3. Compute Gaussian from the transformed and weighted sigma-points

11/03/22

AA 274 | Lecture 12

22

UKF: example

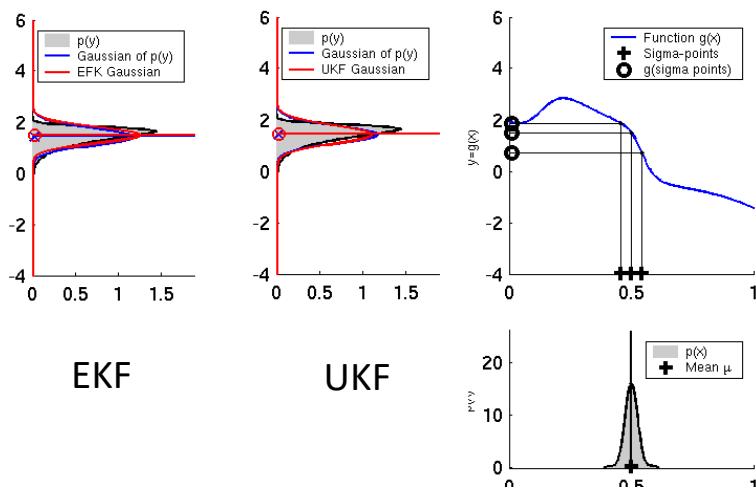


11/03/22

AA 274 | Lecture 12

23

UKF: example



11/03/22

AA 274 | Lecture 12

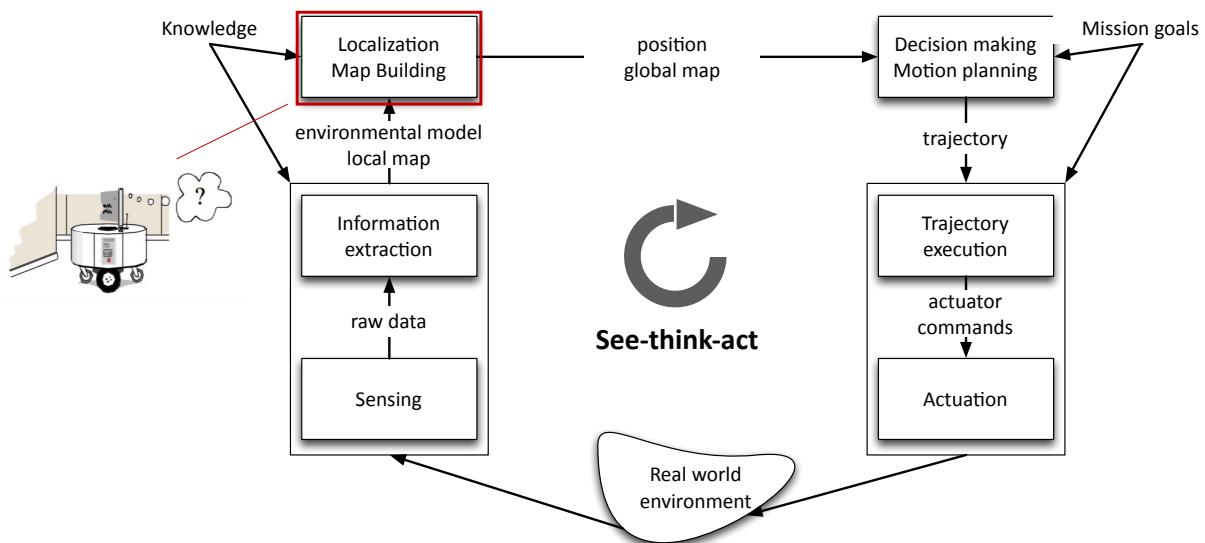
24

Principles of Robot Autonomy I

Non-parametric filtering



Module 3



Today's lecture

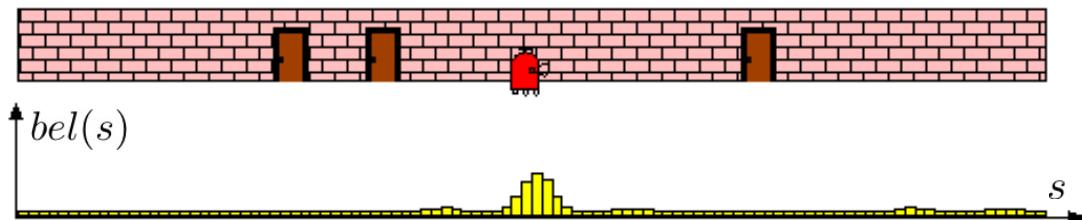
- Aim
 - Learn about non-parametric filters
- Readings
 - S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. MIT press, 2005.
Sections 3.1 – 3.4, 4.1, 4.3, 7.1

Instantiating the Bayes' filter

- Tractable implementations of Bayes' filter exploit structure and / or approximations; two main classes
 - Parametric filters: e.g., KF, EKF, UKF, etc.
 - Non parametric filters: e.g., histogram filter, particle filter, etc.

Histogram filter

- **Key idea:** use *discrete* Bayes' filter as an approximate inference tool for *continuous* state spaces



- Step #1: histogram filters decompose a continuous space into finitely many bins

$$\text{dom}(X_t) = x_{1,t} \cup x_{2,t} \cup \dots x_{K,t}$$

State space

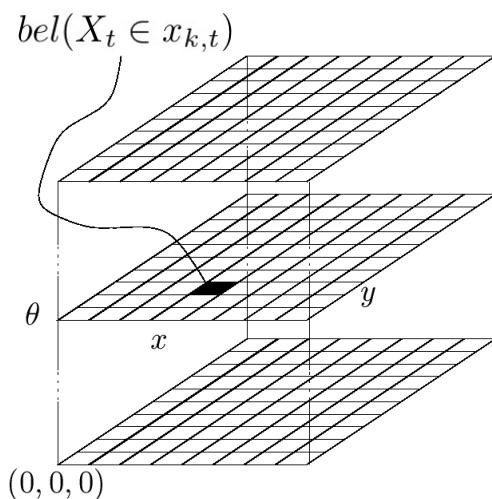
$\{x_{k,t}\}$: convex regions forming a partition of state space (e.g., grid cell)

11/10/22

AA 274 | Lecture 13

5

Example



11/10/22

AA 274 | Lecture 13

6

Histogram filter

- Step #2: assign to each region $x_{k,t}$ a probability $p_{k,t}$; probabilities are then approximated according to a piecewise scheme

$$p(x_t) \equiv \frac{p_{k,t}}{|x_{k,t}|}, \quad \text{for all } x_t \in x_{k,t} \quad \Rightarrow \quad p(X_t \in x_{k,t}) = \int_{x_{k,t}} \frac{p_{k,t}}{|x_{k,t}|} dx_t = p_{k,t}$$

Histogram filter

- Step #3: discretize motion and measurements models, i.e.,

$$p(x_t | u_t, x_{t-1}) \quad \text{and} \quad p(z_t | x_t)$$

1. Select mean state as representative state

$$\hat{x}_{k,t} = |x_{k,t}|^{-1} \int_{x_{k,t}} x_t dx_t$$

2. Approximate measurement model

$$p(z_t | x_{k,t}) \approx p(z_t | \hat{x}_{k,t})$$

3. Approximate transition model

$$p(x_{k,t} | u_t, x_{i,t-1}) \approx \eta |x_{k,t}| p(\hat{x}_{k,t} | u_t, \hat{x}_{i,t-1})$$

- Step #4: execute discrete Bayes' filter with discretized probabilities

Histogram filter

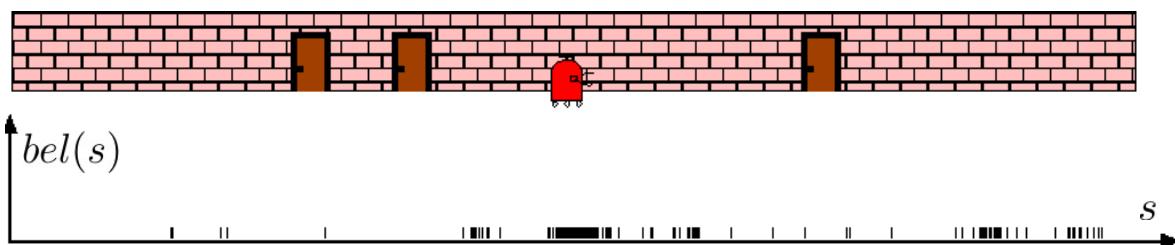
- Then one can run the usual discrete Bayes' filter

- Belief $bel(x_t)$ represented as pmf $\{p_{k,t}\}$

```
Data:  $\{p_{k,t-1}\}, u_t, z_t$ 
Result:  $\{p_{k,t}\}$ 
foreach  $k$  do
     $\bar{p}_{k,t} = \sum_i p(X_t = x_k | u_t, X_{t-1} = x_i) p_{i,t-1};$ 
     $p_{k,t} = \eta p(z_t | X_t = x_k) \bar{p}_{k,t};$ 
end
Return  $\{p_{k,t}\}$ 
```

Particle filter

- Key idea: represent posterior $bel(x_t)$ by a set of random samples



- Allows one to represent non-Gaussian distributions and handle nonlinear transformations in a direct way

Particle filter

- Samples of posterior distribution are called *particles*, denoted as

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]}$$

- A particle represents a hypothesis about what the true world state might be at time t

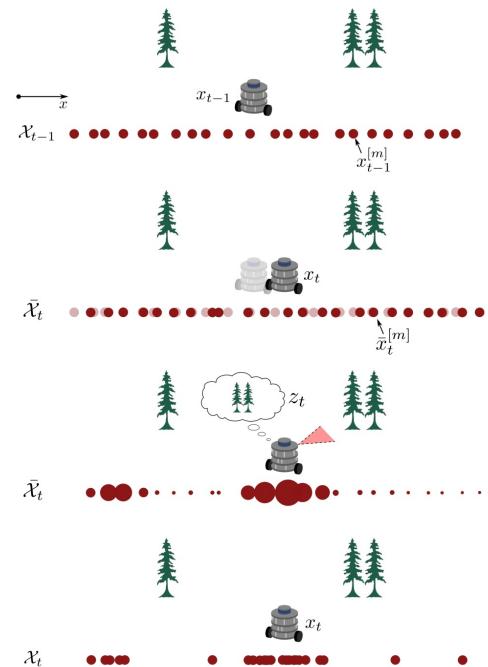
- Ideally, particles should be distributed according to

$$x_t^{[m]} \sim p(x_t | z_{1:t}, u_{1:t}) = bel(x_t)$$

- Matching exact only as $M \rightarrow \infty$, but $M \approx 1000$ usually good enough
- A particle filter constructs the particle set \mathcal{X}_t from the particle set \mathcal{X}_{t-1} recursively, with the goal of matching the distribution $bel(x_t)$

Particle filter: example

- Resampling can be a high variance process (e.g., “weight collapse” can be a problem) motivating the development of lower variance schemes and/or recovery processes
- Many extensions/variants (e.g., Gaussian Sum Particle Filtering in which belief is represented as a Gaussian Mixture Model)



Particle filter: algorithm

- The temporary particle set $\bar{\mathcal{X}}_t$ represents the belief $\overline{bel}(x_t)$
- The particle set \mathcal{X}_t represents the belief $bel(x_t)$
- Importance factors are used to incorporate measurement z_t in the particle set
- After resampling, particles are (as $M \rightarrow \infty$) distributed as

$$bel(x_t) = \eta p(z_t | x_t^{[m]}) \overline{bel}(x_t)$$

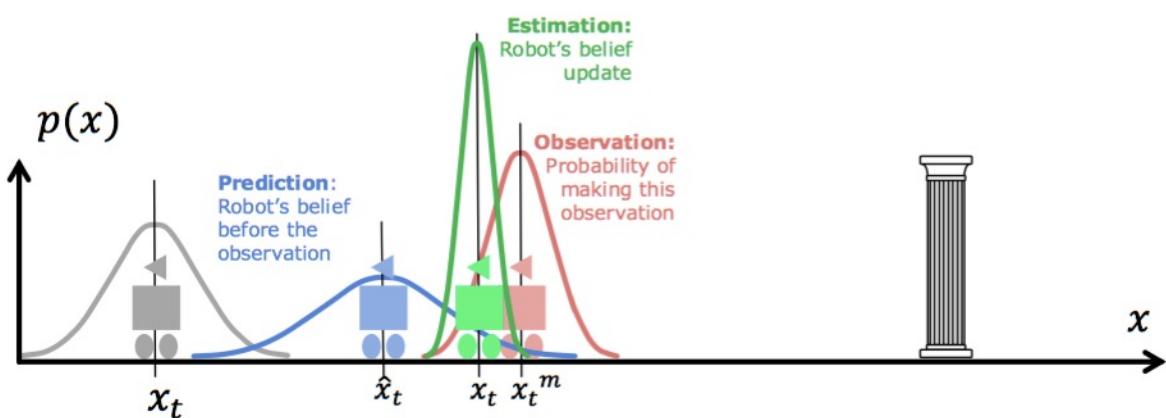
bel(x_{t-1})
 Data: $\mathcal{X}_{t-1}, u_t, z_t$
 Result: \mathcal{X}_t
 $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset;$
 for $i = 1$ to M do
 Prediction: $\overline{bel}(x_t)$ Sample $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$;
 Importance factor $w_t^{[m]} = p(z_t | x_t^{[m]})$;
 $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t \cup (x_t^{[m]}, w_t^{[m]})$;
 end
 for $m = 1$ to M do
 Correction: $bel(x_t)$ Draw i with probability $\propto w_t^{[i]}$;
 Add $x_t^{[i]}$ to \mathcal{X}_t ;
 end
 Return \mathcal{X}_t
 bel(x_t)

11/10/22

AA 274 | Lecture 13

13

Next time



11/10/22

AA 274 | Lecture 13

14

Principles of Robot Autonomy I

Markov localization and EKF-localization

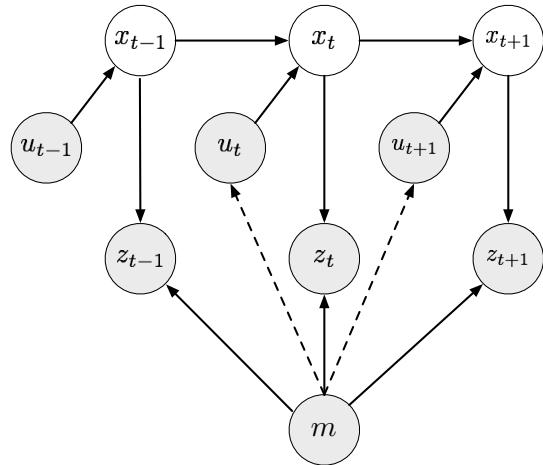


Today's lecture

- Aim
 - Learn about Markov localization, with an emphasis on EKF and non-parametric localization
- Readings
 - S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. MIT press, 2005. Sections 7.2 – 7.6, 8.3

Mobile robot localization

- **Problem:** determine pose of a robot relative to a *given* map
- Localization can be interpreted as the problem of establishing correspondence between the map coordinate system and the robot's local coordinate frame
- This process requires integration of data over time



11/15/22

AA 274 | Lecture 14

3

Local versus global localization

- **Position tracking** assumes that the initial pose is known -> *local* problem well-addressed via Gaussian filters
- In **global localization**, the initial pose is unknown -> *global* problem best addressed via non-parametric, multi-hypothesis filters
- In **kidnapped robot** localization, initial pose is unknown and during operation robot can be "kidnapped" and "teleported" to some other location -> *global* problem best addressed via non-parametric, multi-hypothesis filters

11/15/22

AA 274 | Lecture 14

4

Static versus dynamic environments

- **Static environments** are environments where the only variable quantity is the pose of the robot
- **Dynamic environments** possess objects (e.g., people) other than the robot whose locations change over time -> addressed via either state augmentation or outlier rejection

Passive versus active localization

- In **passive localization**, localization module only *observes* the robot; i.e., robot's motion is not aimed at facilitating localization
- In **active localization**, robot's actions are aimed at minimizing the localization error
- Hybrid approaches are possible

Single-robot versus multi-robot

- In **single-robot localization**, a single, individual robot is involved in the localization process
- In **multi-robot localization**, a team of robots is engaged with localization, possibly cooperatively (or even adversarially!)

In this class we will focus on **local & global, static** (or quasi-static), **passive, single-robot** localization problems

Casting the localization problem within a Bayesian filtering framework

- State x_t , control u_t and measurements z_t have the same meaning as in the general filtering context
- For a differential drive robot equipped with a laser range-finder (returning a set of range r_i and bearing ϕ_i measurements)

$$x_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \quad u_t = \begin{pmatrix} v \\ \omega \end{pmatrix} \quad z_t = \left\{ \begin{pmatrix} r_i \\ \phi_i \end{pmatrix} \right\}_i$$

Casting the localization problem within a Bayesian filtering framework

- A map m is a list of objects in the environment along with their properties

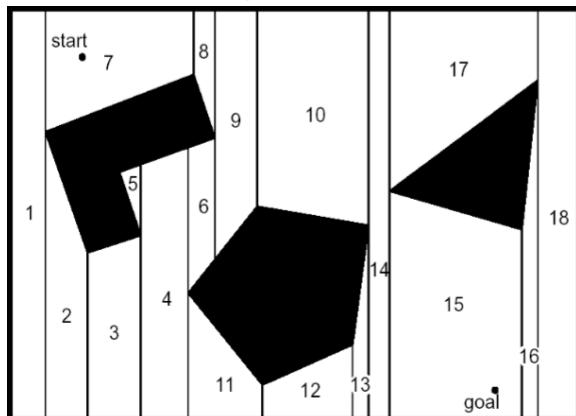
$$m = \{m_1, m_2, \dots, m_N\}$$

- Maps can be

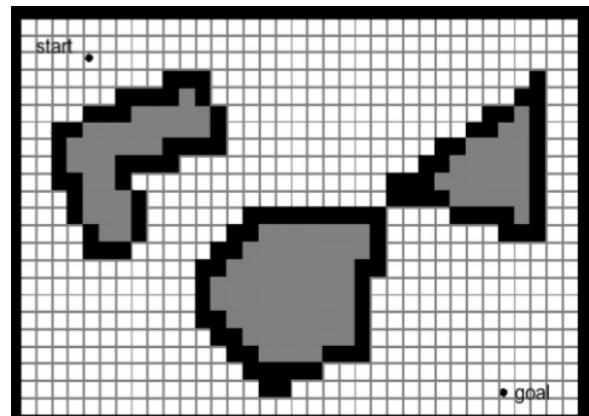
- *Location-based*: index i corresponds to a specific location (hence, they are volumetric)
- *Feature-based*: index i is a feature index, and m_i contains, next to the properties of a feature, the Cartesian location of that feature

Location-based maps

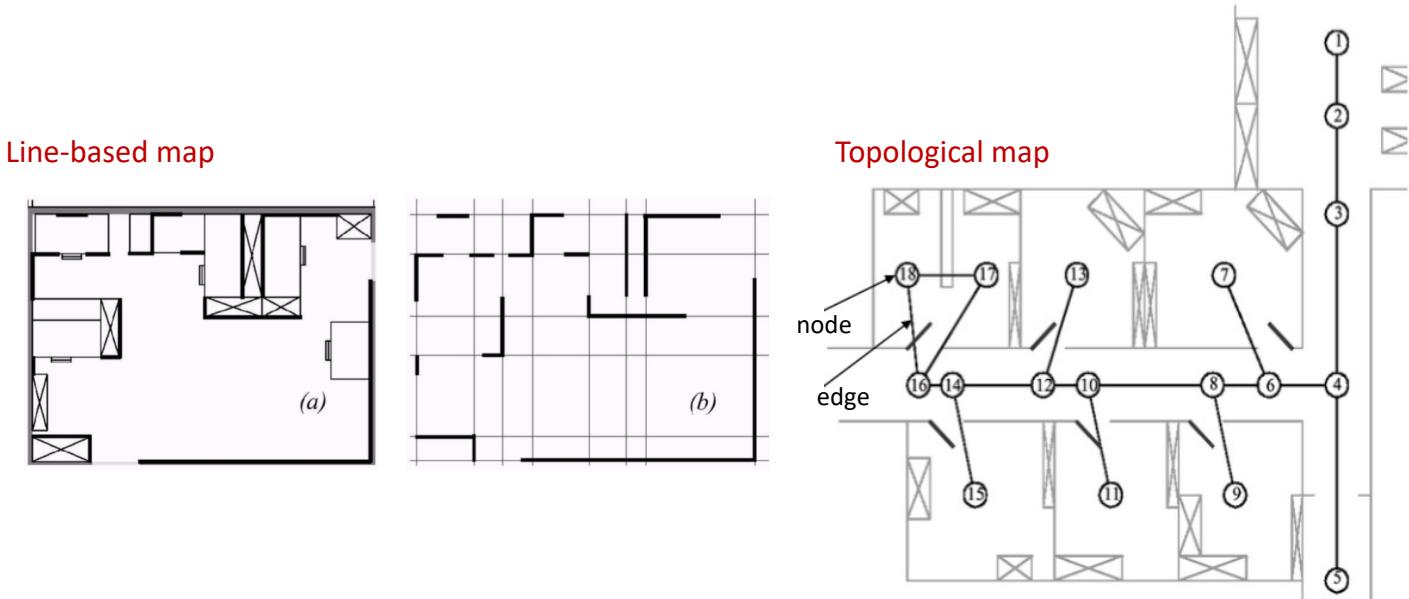
Vertical cell decomposition



Fixed cell decomposition (occupancy grid)



Feature-based maps



11/15/22

AA 274 | Lecture 14

11

Casting the localization problem within a Bayesian filtering framework

- Motion model is probabilistic

$$p(x_t \mid u_t, x_{t-1})$$

$$x_{t-1} \xrightarrow{u_t}$$

- Key fact: $p(x_t | u_t, x_{t-1}) \neq p(x_t | u_t, x_{t-1}, m)$
 - Useful approximation (tight at high update rates)

$$p(x_t | u_t, x_{t-1}, m) \approx \eta \frac{p(x_t | u_t, x_{t-1}) p(x_t | m)}{n(x_t)}$$

Consistency of state
 x_t with map m

Uses approximation

Casting the localization problem within a Bayesian filtering framework

- Measurement model is probabilistic

$$p(z_t | x_t, m)$$

- Sensors usually generate more than one measurement when queried

$$z_t = \{z_t^1, \dots, z_t^K\}$$

- Typically, independence assumption is made

$$p(z_t | x_t, m) = \prod_{k=1}^K p(z_t^k | x_t, m)$$

Markov localization

- Straightforward application of Bayes filter
- Requires a map m as input
- Addresses:
 - Global localization
 - Position tracking
 - Kidnapped robot problem

Data: $bel(x_{t-1}), u_t, z_t, m$

Result: $bel(x_t)$

foreach x_t **do**

$$\left| \begin{array}{l} \overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}, m) bel(x_{t-1}) dx_{t-1}; \\ bel(x_t) = \eta p(z_t | x_t, m) \overline{bel}(x_t); \end{array} \right.$$

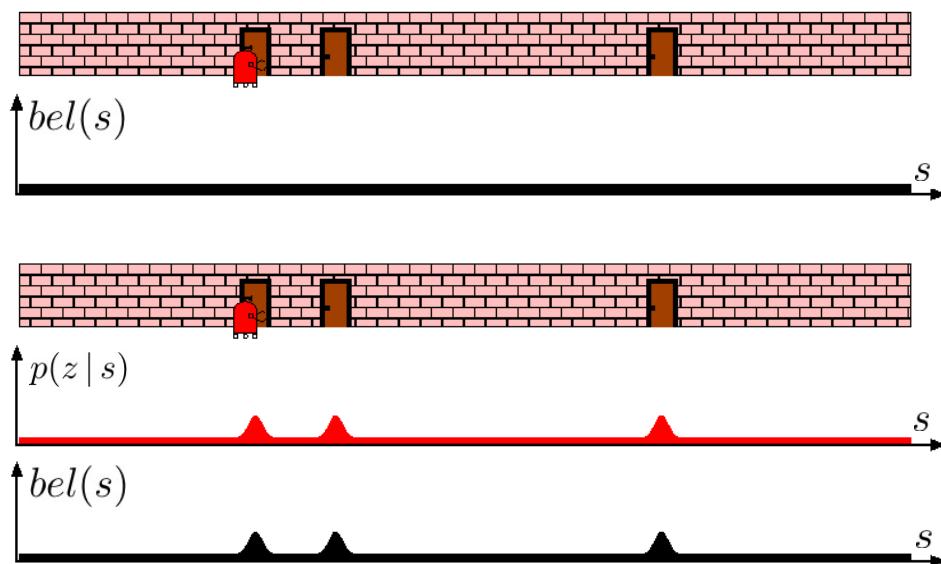
end

Return $bel(x_t)$

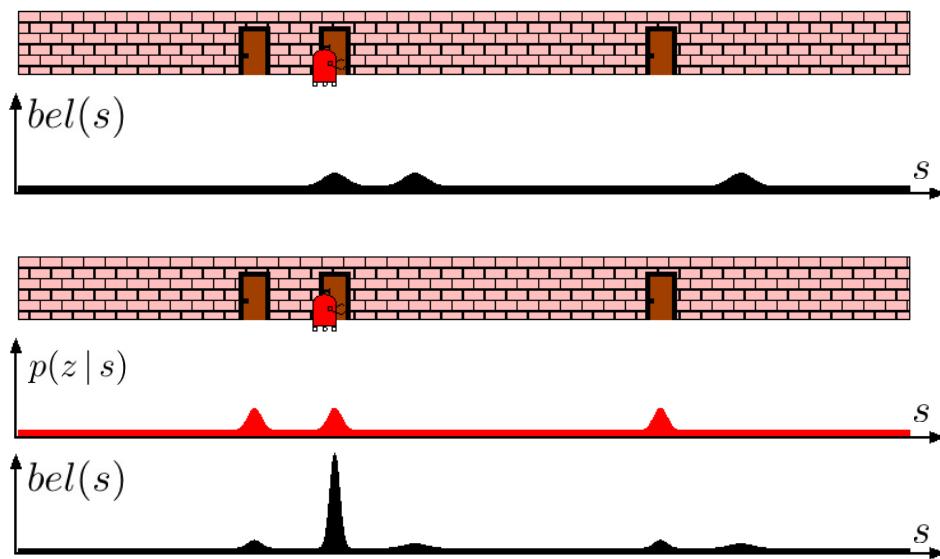
Markov localization: typical choices for initial belief

- Initial belief, $bel(x_0)$ reflects initial knowledge of robot pose
 - If initial pose is known, $bel(x_0) = \begin{cases} 1 & \text{if } x_0 = \bar{x}_0 \\ 0 & \text{otherwise} \end{cases}$
 - If partially known, $bel(x_0) \sim \mathcal{N}(\bar{x}_0, \Sigma_0)$
- For global localization
 - If initial pose is unknown, $bel(x_0) = 1/|X|$

Markov localization: example



Markov localization: example

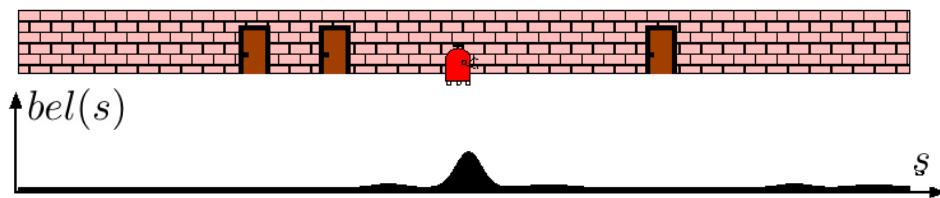


11/15/22

AA 274 | Lecture 14

17

Markov localization: example



11/15/22

AA 274 | Lecture 14

18

Instantiation of Markov localization

- To make algorithm tractable, we need to add some structure to the representation of $bel(x_t)$
 1. Gaussian representation
 2. Particle filter representation

Extended Kalman filter (EKF) localization

- **Key idea:** represent belief $bel(x_t)$ by its first and second moment, i.e., μ_t and Σ_t
- We will develop the EKF localization algorithm under the assumptions that:
 1. A **feature-based map** is available, consisting of point landmarks
$$m = \{m_1, m_2, \dots\}, \quad m_i = (m_{i,x}, m_{i,y})$$

Location of the landmark in the global coordinate frame
 2. There is a sensor that can measure **the range r and the bearing ϕ** of the landmarks relative to the robot's local coordinate frame
- Key concepts carry forward to other map / sensing models

Range and bearing sensors

- Range & bearing sensors are common: features extracted from range scans and stereo vision come with range r and bearing ϕ information
- At time t , a **set** of features is measured (assumed independent)

$$z_t = \{z_t^1, z_t^2, \dots\} = \{(r_t^1, \phi_t^1), (r_t^2, \phi_t^2), \dots\}$$

- Assuming that the i -th measurement at time t corresponds to the j -th landmark in the map, the measurement model is

$$\begin{pmatrix} r_t^i \\ \phi_t^i \end{pmatrix} = \underbrace{\begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \text{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \end{pmatrix}}_{=h(x_t, j, m)} + \mathcal{N}(0, Q_t)$$

Gaussian noise 

The issue of data association

- **Data association problem:** uncertainty may exists regarding the identity of a landmark
- Formally, we define a *correspondence variable* between measurement z_t^i and landmark m_j in the map as (assume N landmarks)
 - $c_t^i \in \{1, \dots, N+1\}$
 - $c_t^i = j \leq N$ if i -th measurement at time t corresponds to j -th landmark
 - $c_t^i = N+1$ if a measurement does not correspond to any landmark
- Two versions of the localization problem
 1. Correspondence variables are known
 2. Correspondence variables are not known (usual case)

EKF localization with known correspondences

- Algorithm is derived from EKF filter
- Assume motion model (in our case, differential drive robot)

$$x_t = g(u_t, x_{t-1}) + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, R_t), \quad G_t := J_g(u_t, \mu_{t-1})$$

- Assume range and bearing measurement model

$$z_t^i = h(x_t, j, m) + \delta_t, \quad \delta_t \sim \mathcal{N}(0, Q_t), \quad H_t^i := \frac{\partial h(\bar{\mu}_t, j, m)}{\partial x_t}$$

$$\frac{\partial h(\bar{\mu}_t, j, m)}{\partial x_t} = \begin{pmatrix} \frac{\partial r_t^i}{\partial \bar{\mu}_{t,x}} & \frac{\partial r_t^i}{\partial \bar{\mu}_{t,y}} & \frac{\partial r_t^i}{\partial \bar{\mu}_{t,\theta}} \\ \frac{\partial \phi_t^i}{\partial \bar{\mu}_{t,x}} & \frac{\partial \phi_t^i}{\partial \bar{\mu}_{t,y}} & \frac{\partial \phi_t^i}{\partial \bar{\mu}_{t,\theta}} \end{pmatrix} = \begin{pmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2}} & 0 \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2} & -1 \end{pmatrix}$$

$$Q_t = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix}$$

11/15/22

AA 274 | Lecture 14

23

EKF localization with known correspondences

- Main difference with EKF filter: multiple measurements are processed at the same time
- We exploit conditional independence assumption

$$p(z_t | x_t, c_t, m) = \prod_i p(z_t^i | x_t, c_t^i, m)$$

- Such assumption allows us to incrementally add the information, as if there was zero motion in between measurements

Data: $(\mu_{t-1}, \Sigma_{t-1}), u_t, z_t, \mathbf{c}_t, \mathbf{m}$

Result: (μ_t, Σ_t)

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

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

foreach $z_t^i = (r_t^i, \phi_t^i)^T$ **do**

$$j = c_t^i;$$

$$\hat{z}_t^i = \begin{pmatrix} \sqrt{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2} \\ \text{atan2}(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{pmatrix};$$

$$S_t^i = H_t^i \bar{\Sigma}_t [H_t^i]^T + Q_t;$$

$$K_t^i = \bar{\Sigma}_t [H_t^i]^T [S_t^i]^{-1};$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i);$$

$$\bar{\Sigma}_t = (I - K_t^i H_t^i) \bar{\Sigma}_t;$$

end

$$\mu_t = \bar{\mu}_t \text{ and } \Sigma_t = \bar{\Sigma}_t;$$

Return (μ_t, Σ_t)

Innovation covariance

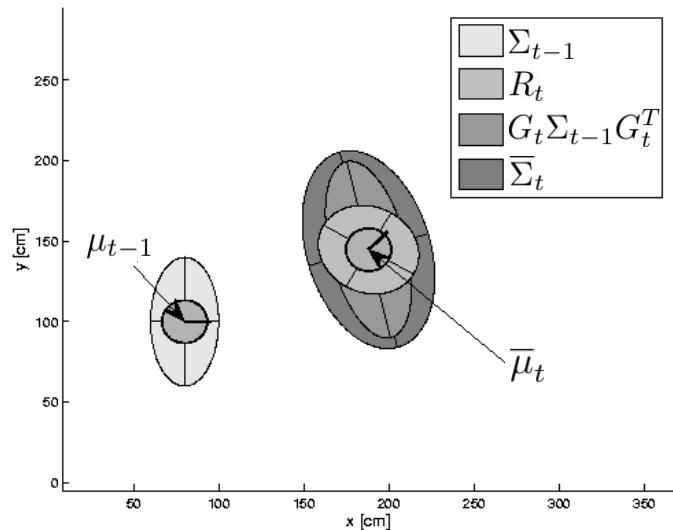
11/15/22

AA 274 | Lecture 14

24

Example of EKF-localization: prediction step

- Observations measure relative distance and bearing to a marker
- For simplicity, we assume that the robot detects only one marker at a time

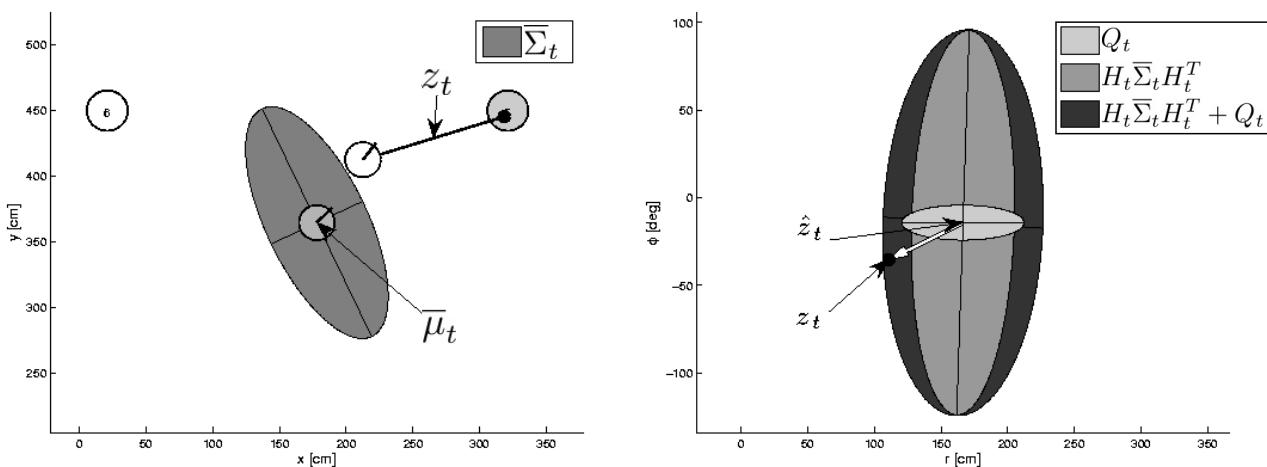


11/15/22

AA 274 | Lecture 14

25

Example of EKF-localization: measurement prediction step

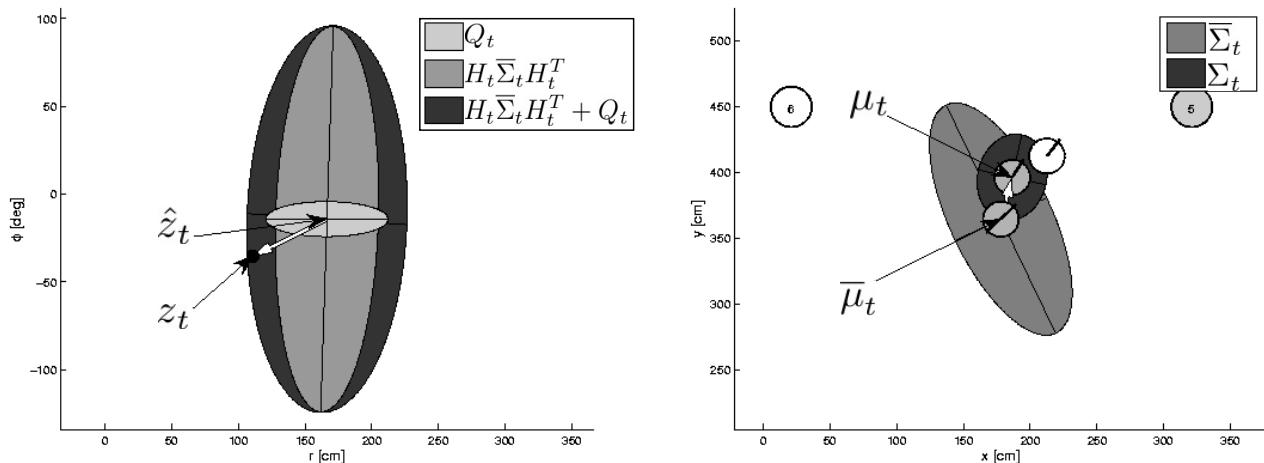


11/15/22

AA 274 | Lecture 14

26

Example of EKF-localization: correction step



11/15/22

AA 274 | Lecture 14

27

EKF localization with unknown correspondences

- **Key idea:** determine the identity of a landmark during localization via maximum likelihood estimation, whereby one first determines the most likely value of c_t , and then takes this value for granted
- Formally, the maximum likelihood estimator determines the correspondence that maximizes the data likelihood

$$\hat{c}_t = \arg \max_{c_t} p(z_t | c_{1:t}, m, z_{1:t-1}, u_{1:t})$$

- Challenge: there are exponentially many terms in the maximization above!
- Solution: perform maximization *separately* for each z_t^i

11/15/22

AA 274 | Lecture 14

28

Estimating the correspondence variables

- Step #1: find

$$p(\mathbf{z}_t^i | c_{1:t}, m, z_{1:t-1}, u_{1:t})$$

- Derivation (sketch)

$$\begin{aligned} p(z_t^i | c_{1:t}, m, z_{1:t-1}, u_{1:t}) &= \int p(z_t^i | x_t, c_{1:t}, m, z_{1:t-1}, u_{1:t}) p(x_t | c_{1:t}, m, z_{1:t-1}, u_{1:t}) dx_t \\ &= \int p(z_t^i | x_t, c_t^i, m) \cdot \overline{bel}(x_t) dx_t \\ &\sim \mathcal{N}(h(x_t, c_t^i, m), Q_t) \quad \text{---} \quad \sim \mathcal{N}(\bar{\mu}_t, \bar{\Sigma}_t) \\ &\approx \mathcal{N}(h(\bar{\mu}_t, c_t^i, m) + H_t^i(x_t - \bar{\mu}_t), Q_t) \end{aligned}$$

Estimating the correspondence variables

- Performing the algebraic calculations

$$p(z_t^i | c_{1:t}, m, z_{1:t-1}, u_{1:t}) \approx \mathcal{N}(h(\bar{\mu}_t, c_t^i, m), H_t^i \bar{\Sigma}_t [H_t^i]^T + Q_t)$$

- Step #2: estimate correspondence as

$$\begin{aligned} \hat{c}_t^i &= \arg \max_{c_t^i} p(z_t^i | c_{1:t}, m, z_{1:t-1}, u_{1:t}) \\ &\approx \arg \max_{c_t^i} \mathcal{N}(z_t^i; h(\bar{\mu}_t, c_t^i, m), H_t \bar{\Sigma}_t H_t^T + Q_t) \end{aligned}$$

EKF localization with unknown correspondences

- Same as before, plus the inclusion of a maximum likelihood estimator for the correspondence variables

```

Data:  $(\mu_{t-1}, \Sigma_{t-1}), u_t, z_t, \textcolor{red}{m}$ 
Result:  $(\mu_t, \Sigma_t)$ 
 $\bar{\mu}_t = g(u_t, \mu_{t-1}) ;$ 
 $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t;$ 
foreach  $z_t^i = (r_t^i, \phi_t^i)^T$  do
    foreach landmark  $k$  in the map do
         $\hat{z}_t^k = \begin{pmatrix} \sqrt{(m_{k,x} - \bar{\mu}_{t,x})^2 + (m_{k,y} - \bar{\mu}_{t,y})^2} \\ \text{atan2}(m_{k,y} - \bar{\mu}_{t,y}, m_{k,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{pmatrix};$ 
         $S_t^k = H_t^k \bar{\Sigma}_t [H_t^k]^T + Q_t;$ 
    end
     $j(i) = \arg \max_k \mathcal{N}(z_t^i; \hat{z}_t^k, S_t^k)$ 
     $K_t^i = \bar{\Sigma}_t [H_t^{j(i)}]^T [S_t^{j(i)}]^{-1};$ 
     $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^{j(i)});$ 
     $\bar{\Sigma}_t = (I - K_t^i H_t^{j(i)}) \bar{\Sigma}_t;$ 
end
 $\mu_t = \bar{\mu}_t$  and  $\Sigma_t = \bar{\Sigma}_t;$ 
Return  $(\mu_t, \Sigma_t)$ 

```

Correspondence estimation

11/15/22

AA 274 | Lecture 14

31

Comments

- Other popular features include lines, corners, distinct patterns
- In the case of lines, an observation would be

$$z_t^i = \begin{bmatrix} r_t^i \\ \alpha_t^i \end{bmatrix}$$

11/15/22

AA 274 | Lecture 14

32

Comments

- To deal with outliers/minor changes in the map, we may also consider a *validation gate*:

Only match landmark j with measurement i if $(z_t^i - \hat{z}_t^j)^T [S_t^k]^{-1} (z_t^i - \hat{z}_t^j) \leq \gamma$

Mahalanobis distance

- A more general approach to deal with data association is the multi-hypothesis tracking filter, where a belief is represented by a mixture of Gaussians (each tracking a sequence of data association decisions)
- UKF localization is another popular approach for feature-based localization

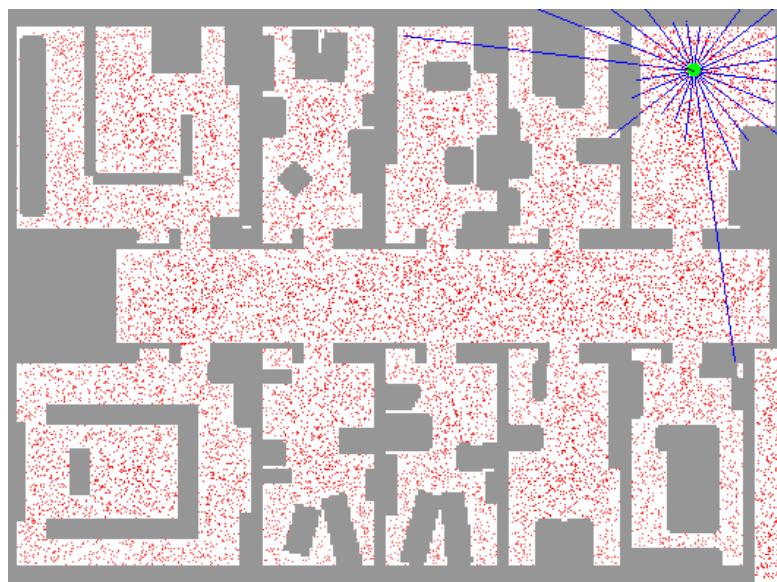
Monte Carlo localization (MCL)

- **Key idea:** represent belief $bel(x_t)$ by a set of M particles
- Requires a map m as input
- Addresses:
 - Global localization
 - Position tracking
 - Kidnapped robot problem (by injecting random particles)
- Can handle dynamic environments via outlier rejection

$$\mathcal{X}_t = \{x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]}\}$$

Data: $\mathcal{X}_{t-1}, u_t, z_t, m$
Result: \mathcal{X}_t
 $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset;$
for $i = 1$ **to** M **do**
 | Sample $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]}, m);$
 | $w_t^{[m]} = p(z_t | x_t^{[m]}, m);$
 | $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t \cup (x_t^{[m]}, w_t^{[m]});$
end
for $i = 1$ **to** M **do**
 | Draw i with probability $\propto w_t^{[i]};$
 | Add $x_t^{[i]}$ to $\mathcal{X}_t;$
end
Return \mathcal{X}_t

MCL: example

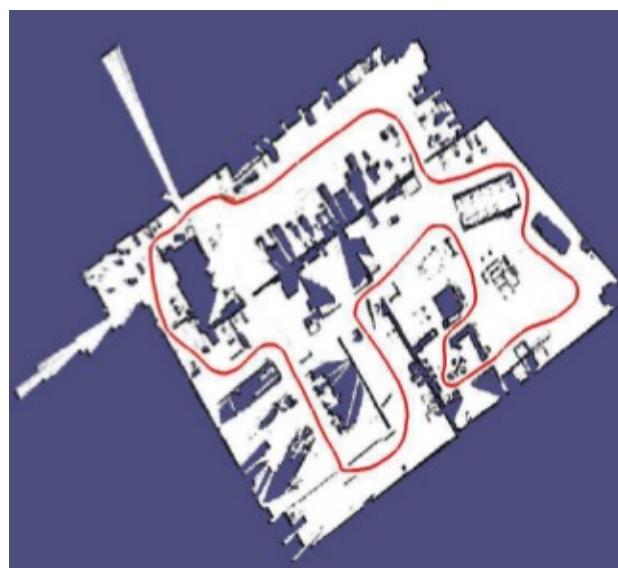


11/15/22

AA 274 | Lecture 14

35

Next time



11/15/22

AA 274 | Lecture 14

36

Principles of Robot Autonomy I

Simultaneous Localization and Mapping (SLAM)

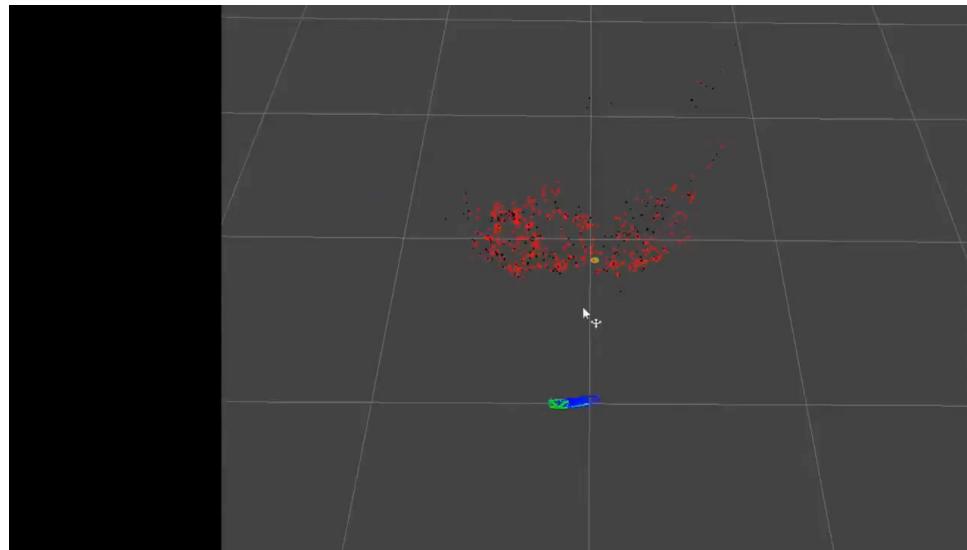


Today's lecture

- Aim
 - Learn about the general SLAM problem
 - Learn about EKF SLAM
 - Introduce particle filter SLAM
- Readings
 - S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. MIT press, 2005.
Sections 8.1 – 8.3, 10.1 – 10.4
 - S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. MIT press, 2005.
Sections 13.1-13.3, 13.5

Simultaneous Localization and Mapping

The SLAM problem:
given measurements
 $z_{1:t}$ and controls $u_{1:t}$,
find the path (or pose)
of the robot and
acquire a map of the
environment



11/17/22

AA 274 | Lecture 15

3

Forms of SLAM

- **Online SLAM problem:** estimate the posterior over the momentary pose along with the map

$$p(x_t, m \mid z_{1:t}, u_{1:t}) \quad \text{or} \quad p(x_t, m, c_t \mid z_{1:t}, u_{1:t})$$

- **Full SLAM problem:** estimate posterior over the entire path along with the map

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \quad \text{or} \quad p(x_{1:t}, m, c_t \mid z_{1:t}, u_{1:t})$$

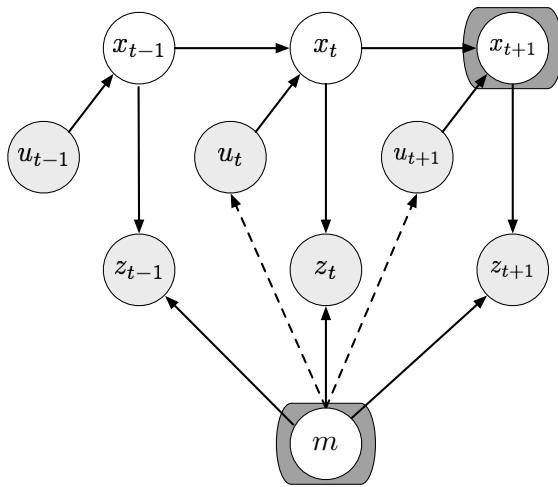
11/17/22

AA 274 | Lecture 15

4

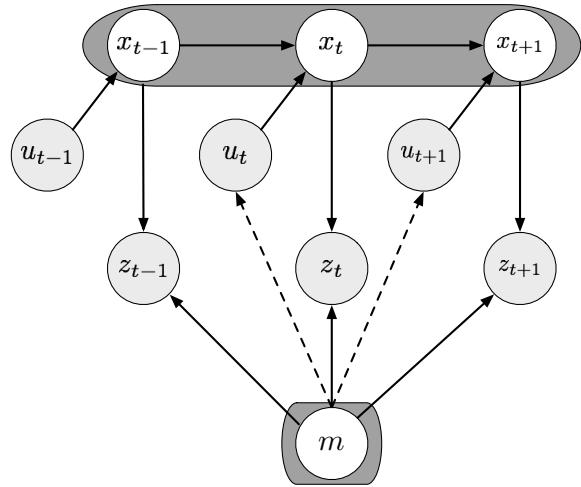
Graphical models of SLAM

Online SLAM



11/17/22

Full SLAM

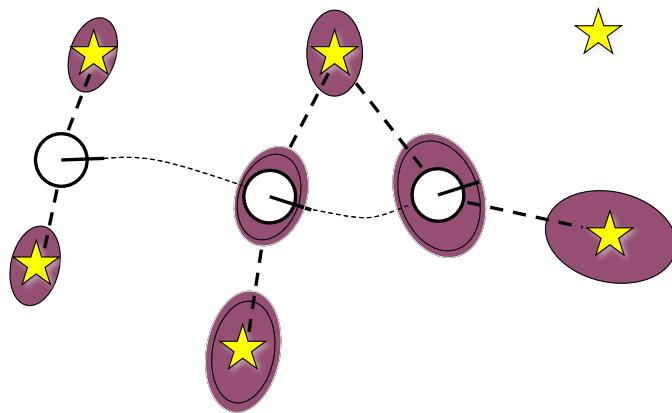


AA 274 | Lecture 15

5

The challenge of SLAM

- Robot path and map are both **unknown**



- Path error is correlated with map error

11/17/22

AA 274 | Lecture 15

6

EKF SLAM

- Historically the earliest SLAM algorithm
- **Key idea:** apply EKF to online SLAM using maximum likelihood data association
- Assumptions:
 1. Gaussian assumption for motion and perception noise, and Gaussian approximation for belief (essential)
 2. Feature-based maps (essential)
- Two versions of the problem
 1. Correspondence variables are known
 2. Correspondence variables are not known (usual case)

EKF SLAM with known correspondences

- Similar to EKF localization algorithm with known correspondences
- **Key difference:** in addition to estimate the robot pose x_t , the EKF SLAM algorithm also estimates the coordinates of **all** landmarks
- Define combined state vector

$$y_t := \begin{pmatrix} x_t \\ m \end{pmatrix} = (x, y, \theta, m_{1,x}, m_{1,y}, m_{2,x}, m_{2,y} \dots m_{N,x}, m_{N,y})^T$$

3 + 2N vector

- **Goal:** calculate the **online posterior**

$$p(y_t | z_{1:t}, u_{1:t})$$

Motion and sensing model

- (Following discussion is for illustration purposes; setup can be generalized to other motion and sensing models)
- Assume motion model with state $x_t = (x, y, \theta)$

$$y_t = g(u_t, y_{t-1}) + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, R_t), \quad G_t := J_g(u_t, \mu_{t-1})$$

where we assume that the landmarks are *static*, that is

1. $g(u_t, y_{t-1})$ is a $3+2N$ vector, whose last $2N$ components are the same as those in y_{t-1}
2. R_t has zero entries, except for the top left 3×3 block

Motion and sensing model

- Assume range and bearing measurement model

$$z_t^i = \underbrace{\left(\sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \right.}_{:= h(y_t, j)} \left. + \text{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \right) + \delta_t, \quad \delta_t \sim \mathcal{N}(0, Q_t), \quad Q_t = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix}$$

- Usual linear approximation for sensing model (with $j = c_t^i$)

$$h(y_t, j) \approx h(\bar{\mu}_t, j) + H_t^i(y_t - \bar{\mu}_t), \quad \text{where } H_t^i := \frac{\partial h(\bar{\mu}_t, j)}{\partial y_t}$$

- Since h depends only on x_t and m_j , H_t^i can be factored as

$$H_t^i = h_t^i F_{x,j}$$

Motion and sensing model

- First term, a 2×5 matrix, is the Jacobian of $h(y_t, j)$ at $\bar{\mu}_t$ w.r.t. x_t and m_j :

$$h_t^i = \frac{\partial h(\bar{\mu}_t, j)}{\partial(x_t, m_j)} = \begin{pmatrix} \frac{\bar{\mu}_{t,x} - \bar{\mu}_{j,x}}{\sqrt{q_{t,j}}} & \frac{\bar{\mu}_{t,y} - \bar{\mu}_{j,y}}{\sqrt{q_{t,j}}} & 0 & \frac{\bar{\mu}_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q_{t,j}}} & \frac{\bar{\mu}_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q_{t,j}}} \\ \frac{\bar{\mu}_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q_{t,j}}} & \frac{\bar{\mu}_{t,x} - \bar{\mu}_{j,x}}{\sqrt{q_{t,j}}} & -1 & \frac{\bar{\mu}_{t,y} - \bar{\mu}_{j,y}}{\sqrt{q_{t,j}}} & \frac{\bar{\mu}_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q_{t,j}}} \end{pmatrix}$$

where $q_{t,j} := (\bar{\mu}_{j,x} - \bar{\mu}_{t,x})^2 + (\bar{\mu}_{j,y} - \bar{\mu}_{t,y})^2$

- Second term, a $5 \times (3+2N)$ matrix, maps h_t^i into H_t^i :

$$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & \underbrace{0 \cdots 0}_{2j-2} & 0 & 1 & \underbrace{0 \cdots 0}_{2N-2j} \end{pmatrix}$$

11/17/22

AA 274 | Lecture 15

11

Initialization

- Initial belief expressed as

$$\mu_0 = (0, 0, 0 \dots 0)^T$$

Initialization for pose variables

$$\Sigma_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \infty & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \infty \end{pmatrix}$$

(3+2N) x (3+2N)

11/17/22

AA 274 | Lecture 15

12

Initialization

- When a landmark is observed for the first time, the landmark estimate $(\bar{\mu}_{j,x}, \bar{\mu}_{j,y})^T$ is initialized with the expected position, that is

$$\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix}$$

- Bearing only SLAM would require multiple sightings

EKF SLAM algorithm

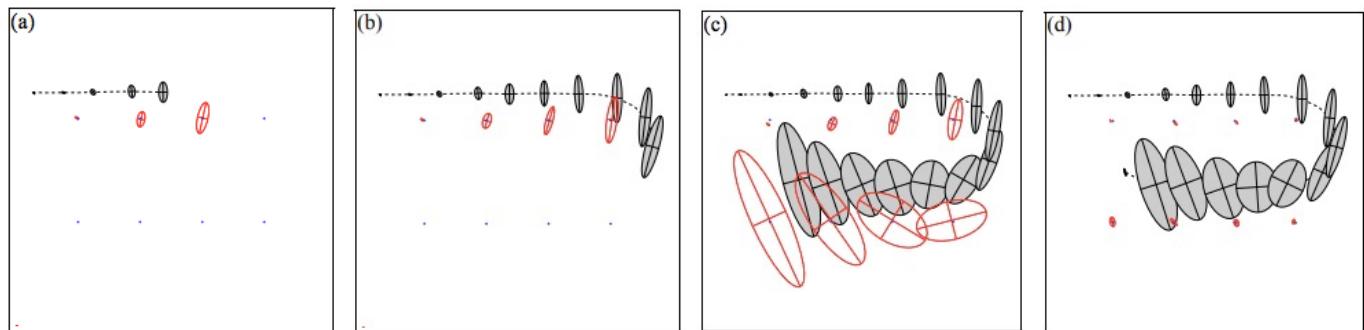
- Similar to EKF localization; main differences:
 - Augmented state vector
 - Augmented dynamics (with trivial dynamics for the landmarks)
 - Initialization of unseen landmarks
 - Augmented measurement Jacobian

```

Data:  $(\mu_{t-1}, \Sigma_{t-1}), u_t, z_t, \textcolor{red}{c_t}$ 
Result:  $(\mu_t, \Sigma_t)$ 
 $\bar{\mu}_t = g(u_t, \mu_{t-1});$ 
 $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t;$ 
foreach  $z_t^i = (r_t^i, \phi_t^i)^T$  do
   $j = c_t^i;$ 
  if landmark  $j$  never seen before then
     $\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix};$ 
  end
   $\hat{z}_t^i = \left( \sqrt{(\bar{\mu}_{j,x} - \bar{\mu}_{t,x})^2 + (\bar{\mu}_{j,y} - \bar{\mu}_{t,y})^2}, \right.$ 
   $\left. \text{atan2}(\bar{\mu}_{j,y} - \bar{\mu}_{t,y}, \bar{\mu}_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \right);$ 
   $H_t^i = \textcolor{red}{h}_t^i F_{x,j};$ 
   $S_t^i = H_t^i \bar{\Sigma}_t [H_t^i]^T + Q_t;$ 
   $K_t^i = \bar{\Sigma}_t [H_t^i]^T [S_t^i]^{-1};$ 
   $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i);$ 
   $\bar{\Sigma}_t = (I - K_t^i H_t^i) \bar{\Sigma}_t;$ 
end
 $\mu_t = \bar{\mu}_t$  and  $\Sigma_t = \bar{\Sigma}_t;$ 
Return  $(\mu_t, \Sigma_t)$ 

```

Example



EKF SLAM with unknown correspondences

- **Key idea:** use an incremental maximum likelihood estimator to determine correspondences
- Similar to EKF localization with unknown correspondences, but now we also need to create hypotheses for new landmarks
- **Caveat:** maximum likelihood data association often makes the algorithm brittle, as it is not possible to revise past data associations

EKF SLAM with unknown correspondences

- In the measurement update loop, we first create the hypothesis of a new landmark
- A new landmark is created if the Mahalanobis distance to all existing landmarks exceeds the value α

Data: $(\mu_{t-1}, \Sigma_{t-1}), u_t, z_t, \textcolor{red}{N_{t-1}}$
Result: (μ_t, Σ_t)

$$N_t = N_{t-1};$$

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

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

Hypothesis
for new
landmark

```

foreach  $z_t^i = (r_t^i, \phi_t^i)^T$  do
     $\begin{pmatrix} \bar{\mu}_{N_t+1,x} \\ \bar{\mu}_{N_t+1,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix};$ 
    for  $k = 1$  to  $N_t + 1$  do
         $\hat{z}_t^k = \begin{pmatrix} \sqrt{(\bar{\mu}_{j,x} - \bar{\mu}_{t,x})^2 + (\bar{\mu}_{j,y} - \bar{\mu}_{t,y})^2} \\ \text{atan2}(\bar{\mu}_{j,y} - \bar{\mu}_{t,y}, \bar{\mu}_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{pmatrix};$ 
         $H_t^k = h_t^k F_{x,k};$ 
         $S_t^k = H_t^k \bar{\Sigma}_t [H_t^k]^T + Q_t;$ 
         $\pi_k = (z_t^i - \hat{z}_t^k)^T [S_t^k]^{-1} (z_t^i - \hat{z}_t^k);$ 
    end
     $\pi_{N_t+1} = \alpha;$ 
     $j(i) = \operatorname{argmin}_k \pi_k;$  Hypothesis test
     $N_t = \max\{N_t, j(i)\};$ 
     $K_t^i = \bar{\Sigma}_t [H_t^{j(i)}]^T [S_t^{j(i)}]^{-1};$ 
     $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^{j(i)});$ 
     $\bar{\Sigma}_t = (I - K_t^i H_t^{j(i)}) \bar{\Sigma}_t;$ 

```

Mahalanobis
distance

end

$\mu_t = \bar{\mu}_t$ and $\Sigma_t = \bar{\Sigma}_t;$
Return (μ_t, Σ_t)

11/17/22

AA 274 | Lecture 15

17

Making EKF SLAM robust

- A key issue is represented by the fact that **fake landmarks** might be created; furthermore, EKF can **diverge** if nonlinearities are large
- Several techniques exist to mitigate such issues
 1. Outlier rejection schemes, for example via provisional landmark lists
 2. Strategies to enhance the distinctiveness of landmarks
 - Spatial arrangement
 - Signatures
 - Enforcing geometric constraints
- **Dilemma of EKF SLAM:** accurate localization typically requires dense maps, but EKF requires sparse maps due to quadratic update complexity

11/17/22

AA 274 | Lecture 15

18

Particle filter SLAM

- **Key idea:** use particles to approximate the belief, and particle filter to simultaneously estimate the robot path and the map
- Goal is to solve **full-scale** SLAM, i.e., estimate

$$p(x_{1:t}, m, c_t \mid z_{1:t}, u_{1:t})$$

- Challenge: naïve implementation of particle filter to SLAM is intractable, due to the excessively large number of particles required
- **Key insight:** knowledge of the robot's true path renders features conditionally independent -> mapping problem can be *factored* into separate problems, one for each feature in the map

Factoring the posterior

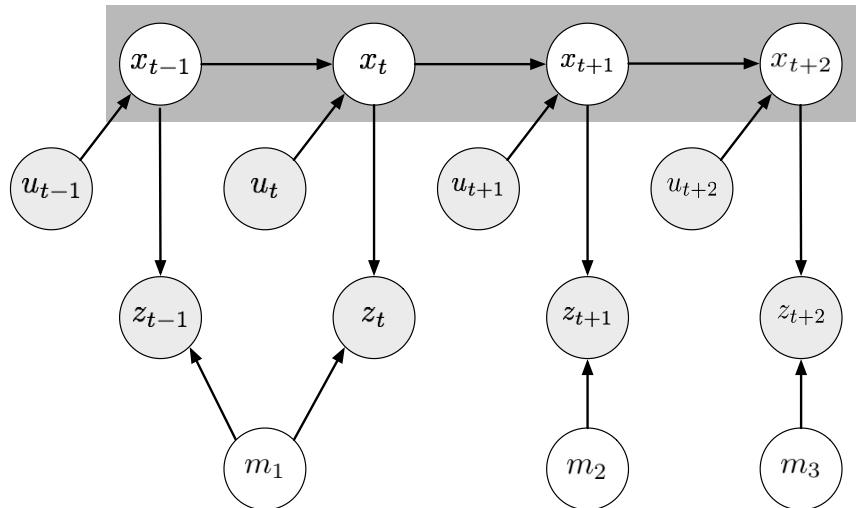
- The key mathematical insight behind particle filter SLAM is the factorization of the posterior

$$p(y_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^N p(m_n \mid x_{1:t}, z_{1:t}, c_{1:t})$$

The diagram shows the factorization of the posterior probability. The main equation is $p(y_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^N p(m_n \mid x_{1:t}, z_{1:t}, c_{1:t})$. Three red arrows point upwards from below, each labeled with a component name: "SLAM posterior" (pointing to $p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t})$), "Path posterior (particles)" (pointing to $\prod_{n=1}^N p(m_n \mid x_{1:t}, z_{1:t}, c_{1:t})$), and "Feature posterior (EKF)" (pointing to $p(m_n \mid x_{1:t}, z_{1:t}, c_{1:t})$).

Factoring the posterior

- Intuition



11/17/22

AA 274 | Lecture 15

21

Factoring the posterior

- Proof follows from Bayes' rule and induction
- Step #1:

$$\begin{aligned} p(y_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) &= p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) p(m \mid x_{1:t}, z_{1:t}, u_{1:t}, c_{1:t}) \\ &= p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) p(m \mid x_{1:t}, z_{1:t}, c_{1:t}) \end{aligned}$$

11/17/22

AA 274 | Lecture 15

22

Factoring the posterior

- Step 2.a: assume $c_t \neq n$

$$p(m_n | x_{1:t}, z_{1:t}, c_{1:t}) = p(m_n | x_{1:t-1}, z_{1:t-1}, c_{1:t-1})$$

- Step 2.b: assume $c_t = n$

$$\begin{aligned} p(m_{c_t} | x_{1:t}, z_{1:t}, c_{1:t}) &= \frac{p(z_t | m_{c_t}, x_{1:t}, z_{1:t-1}, c_{1:t}) p(m_{c_t} | x_{1:t}, z_{1:t-1}, c_{1:t})}{p(z_t | x_{1:t}, z_{1:t-1}, c_{1:t})} \\ &= \frac{p(z_t | m_{c_t}, x_t, c_t) p(m_{c_t} | x_{1:t-1}, z_{1:t-1}, c_{1:t-1})}{p(z_t | x_{1:t}, z_{1:t-1}, c_{1:t})} \end{aligned}$$

Factoring the posterior

- Step 3 (induction): assume at time $t - 1$ (induction hypothesis)

$$p(m | x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) = \prod_{n=1}^N p(m_n | x_{1:t-1}, z_{1:t-1}, c_{1:t-1})$$

Factoring the posterior

- Then at time t

$$\begin{aligned}
 p(m \mid x_{1:t}, z_{1:t}, c_{1:t}) &= \frac{p(z_t \mid m, x_{1:t}, z_{1:t-1}, c_{1:t}) p(m \mid x_{1:t}, z_{1:t-1}, c_{1:t})}{p(z_t \mid x_{1:t}, z_{1:t-1}, c_{1:t})} \\
 &= \frac{p(z_t \mid m, x_t, c_t) p(m \mid x_{1:t-1}, z_{1:t-1}, c_{1:t-1})}{p(z_t \mid x_{1:t}, z_{1:t-1}, c_{1:t})} \\
 &= \frac{p(z_t \mid m, x_t, c_t)}{p(z_t \mid x_{1:t}, z_{1:t-1}, c_{1:t})} \prod_{n=1}^N p(m_n \mid x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) \\
 &= \frac{p(z_t \mid m, x_t, c_t)}{p(z_t \mid x_{1:t}, z_{1:t-1}, c_{1:t})} \underbrace{p(m_{c_t} \mid x_{1:t-1}, z_{1:t-1}, c_{1:t-1})}_{\text{Step 2.b}} \prod_{n \neq c_t} \underbrace{p(m_n \mid x_{1:t-1}, z_{1:t-1}, c_{1:t-1})}_{\text{Step 2.a}} \\
 &= p(m_{c_t} \mid x_{1:t}, z_{1:t}, c_{1:t}) \prod_{n=1}^N p(m_n \mid x_{1:t}, z_{1:t}, c_{1:t}) = \prod_{n=1}^N p(m \mid x_{1:t}, z_{1:t}, c_{1:t})
 \end{aligned}$$

11/17/22

AA 274 | Lecture 15

25

Fast SLAM with known correspondences

- **Key idea:** exploit factorization result to decompose problem into sub-problems
 - Path posterior is estimated using particle filter
 - Map features are estimated via EKF conditioned on the robot path (one EKF for each feature)
- Accordingly, particles in Fast SLAM are represented as

$$Y_t^{[k]} = \langle x_t^{[k]}, \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, \dots, \mu_{N,t}^{[k]}, \Sigma_{N,t}^{[k]} \rangle$$

11/17/22

AA 274 | Lecture 15

26

Fast SLAM with known correspondences

- Each particle possesses its own set of EKFs!
- In total there are NM EKFs
- Filtering involves generating a new particle set Y_t from Y_{t-1} by incorporating a new control u_t and a new measurement z_t with associated correspondence variable c_t
- Update entails three steps
 1. Extend path posterior
 2. Update observed feature estimate
 3. Resample

Step 1: Extending path posterior

- For each particle $Y_t^{[k]}$, sample pose x_t according to motion posterior
$$x_t^k \sim p(x_t | x_{t-1}^k, u_t)$$
- Sample $x_t^{[k]}$ is then concatenated with previous poses $x_{1:t-1}^{[k]}$



Step 2: updating observed feature estimate

- This step entails updating the posterior over the feature estimates
- If $c_t \neq n$

$$\langle \mu_{n,t}^{[k]}, \Sigma_{n,t}^{[k]} \rangle = \langle \mu_{n,t-1}^{[k]}, \Sigma_{n,t-1}^{[k]} \rangle$$

- If $c_t = n$

$$p(m_{c_t} | x_{1:t}, z_{1:t}, c_{1:t}) = \eta p(z_t | m_{c_t}, x_t, c_t) p(m_{c_t} | x_{1:t-1}, z_{1:t-1}, c_{1:t-1})$$


 $\sim \mathcal{N}(\mu_{n,t-1}^{[k]}, \Sigma_{n,t-1}^{[k]})$

Step 2: updating observed feature estimate

- To ensure that the new estimate is Gaussian as well, measurement model is linearized as usual

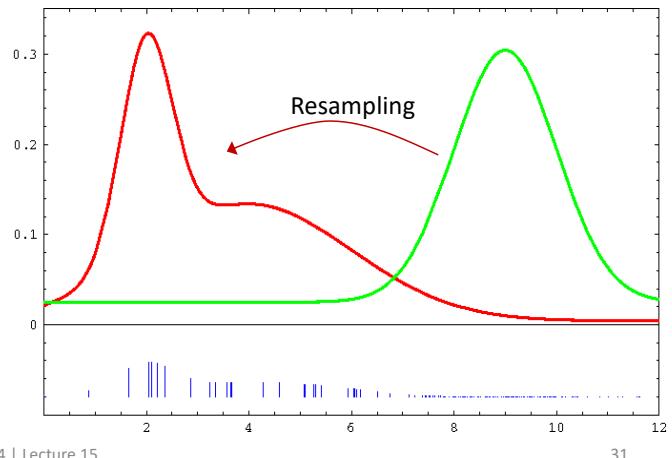
$$h(m_{c_t}, x_t^{[k]}) \approx h(\mu_{c_t,t-1}^{[k]}, x_t^{[k]}) + \underbrace{h'(\mu_{c_t,t-1}^{[k]}, x_t^{[k]})}_{:= H_t^{[k]}} (m_{c_t} - \mu_{c_t,t-1}^{[k]})$$

- Mean and covariance are then obtained as per standard EKF

$$\begin{aligned} K_t^{[k]} &= \Sigma_{c_t,t-1}^{[k]} [H_t^{[k]}]^T (H_t^{[k]} \Sigma_{c_t,t-1}^{[k]} [H_t^{[k]}]^T + Q_t)^{-1} \\ \mu_{c_t,t}^{[k]} &= \mu_{c_t,t-1}^{[k]} + K_t^{[k]} (z_t - \hat{z}_t^{[k]}) \\ \Sigma_{c_t,t}^{[k]} &= (I - K_t^{[k]} H_t^{[k]}) \Sigma_{c_t,t-1}^{[k]} \end{aligned}$$

Step 3: resampling

- Step 1 generates pose x_t only in accordance with the most recent control u_t , paying no attention to the measurement z_t
- Goal: resample particles to correct for this mismatch



11/17/22

AA 274 | Lecture 15

31

Step 3: resampling

- How do we find the weights?
- Path particles at this stage are distributed according to

$$p(x_{1:t}^{[k]} \mid z_{1:t-1}, u_{1:t}, c_{1:t-1}) = p(x_t \mid x_{t-1}^k, u_t) p(x_{1:t-1}^{[k]} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1})$$

↑
Sampling distribution ↑
Distribution of path
particles in $Y_{t-1}^{[k]}$

- The target distribution takes into account z_t , along with c_t

$$p(x_{1:t}^{[k]} \mid z_{1:t}, u_{1:t}, c_{1:t})$$

11/17/22

AA 274 | Lecture 15

32

Step 3: resampling

- Importance factor is then given by

$$\begin{aligned} w_t^{[k]} &= \frac{p(x_{1:t}^{[k]} | z_{1:t}, u_{1:t}, c_{1:t})}{p(x_{1:t}^{[k]} | z_{1:t-1}, u_{1:t}, c_{1:t-1})} \\ &= \frac{\eta p(z_t | x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}, c_{1:t}) p(x_{1:t}^{[k]} | z_{1:t-1}, u_{1:t}, c_{1:t})}{p(x_{1:t}^{[k]} | z_{1:t-1}, u_{1:t}, c_{1:t-1})} \\ &= \frac{\eta p(z_t | x_t^{[k]}, c_t) p(x_{1:t}^{[k]} | z_{1:t-1}, u_{1:t}, c_{1:t-1})}{p(x_{1:t}^{[k]} | z_{1:t-1}, u_{1:t}, c_{1:t-1})} \\ &= \eta p(z_t | x_t^{[k]}, c_t) \end{aligned}$$

Step 3: resampling

- To derive an (approximate) close-form expression for $w_t^{[k]}$, one can then apply the total probability law along with a linearization of the measurement model to obtain

$$w_t^{[k]} = \eta \det(2\pi Q_t^{[k]})^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (z_t - \hat{z}_t^{[k]}) [Q_t^{[k]}]^{-1} (z_t - \hat{z}_t^{[k]}) \right\}$$

$$Q_t^{[k]} = [H_t^{[k]}]^T \Sigma_{n,t-1}^{[k]} H_t^{[k]} + Q_t$$

Fast Slam algorithm

- Key fact: only the most recent pose is used in the process of generating a new particle at time t !

```

Data:  $Y_{t-1}, u_t, z_t, c_t$ 
Result:  $Y_t$ 
for  $k = 1$  to  $M$  do
     $x_t^k \sim p(x_t | x_{t-1}^k, u_t);$ 
     $j = c_t;$ 
    if feature  $j$  never seen before then
        initialize feature
    else
         $\hat{z} = h(\mu_{j,t-1}^{[k]}, x_t^{[k]});$ 
        calculate Jacobian  $H$ ;
         $Q = H\Sigma_{j,t-1}^{[k]}H^T + Q_t;$ 
         $K = \Sigma_{j,t-1}^{[k]}H^T Q^{-1};$ 
         $\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z});$ 
         $\Sigma_{j,t}^{[k]} = (I - KH)\Sigma_{j,t-1}^{[k]};$ 
         $w^{[k]} = \det(2\pi Q)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - \hat{z})Q_t^{-1}(z_t - \hat{z})\right\};$ 
    end
    for all other features  $n \neq j$  do
         $\langle \mu_{n,t}^{[k]}, \Sigma_{n,t}^{[k]} \rangle = \langle \mu_{n,t-1}^{[k]}, \Sigma_{n,t-1}^{[k]} \rangle;$ 
    end
     $Y_t = \emptyset;$ 
end
for  $i = 1$  to  $M$  do
    Draw  $k$  with probability  $\propto w^{[k]}$ ;
    Add  $\langle x_t^{[k]}, \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, \dots, \mu_{N,t}^{[k]}, \Sigma_{N,t}^{[k]} \rangle$  to  $Y_t$ ;
end
Return  $Y_t$ 

```

11/17/22

AA 274 | Lecture 15

35

Fast SLAM with unknown correspondences

- Key advantage of particle filters: each particle can rely on its own, local data association decisions!
- Key idea: per-particle data association generalizes the per-filter data association to individual particles
- Each particle maintains a *local set* of data association variables, $\hat{c}_t^{[k]}$
- Data association is solved, as usual, via maximum likelihood estimation

$$\hat{c}_t^{[k]} = \arg \max_{c_t} p(z_t | c_t, \hat{c}_{1:t-1}^{[k]}, x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t})$$

Computed, as usual, via total probability law + linearization

11/17/22

AA 274 | Lecture 15

36

Summary: Gaussian filtering (EKF, UKF)

- **Key ideas:**
 - Represent a belief with a Gaussian distribution
 - Assume all uncertainty sources are Gaussian
- Pros:
 - Runs online
 - Well understood
 - Works well when uncertainty is low
- Cons:
 - Unimodal estimate
 - States must be well approximated by a Gaussian
 - Works poorly when uncertainty is high

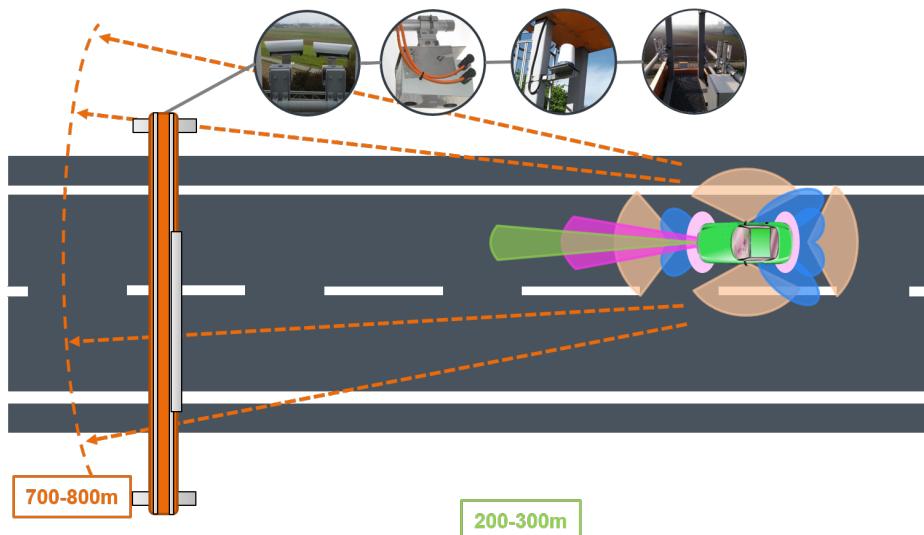
Summary: particle filter approaches

- **Key ideas:**
 - Approximate belief with particles
 - Use particle filters to perform inference
- Pros:
 - Can handle “any” noise distribution
 - Relatively easy to implement
 - Naturally represents multimodal beliefs
 - Robust to data association errors
- Cons:
 - Does not scale well to large dimensional problems
 - Might require many particles for good convergence
 - Might have issues with loop closure

Final considerations

- A recent overview of SLAM (with strong focus on graph SLAM): C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age." *IEEE Transactions on Robotics* 32, no. 6 (2016): 1309-1332.
- Trends: from the classical age, to the algorithmic-analysis age, to the robust perception age
- Popular software packages
 - <https://www.openslam.org/>: comprehensive list of open-source SLAM software
 - <https://github.com/pamela-project/slambench>: popular benchmark framework
 - Commercial SDKs: ARCore/ARKit from Google/Apple, Oculus Insight

Next time



Principles of Robot Autonomy I

Multi-sensor perception and sensor fusion I

Daniel Watzenig

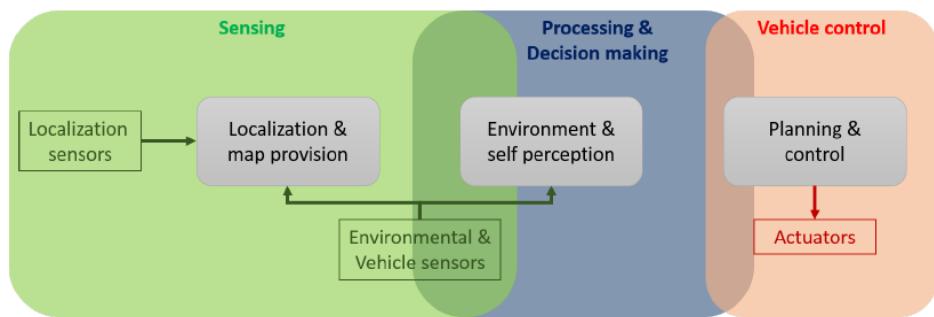


Today's lecture

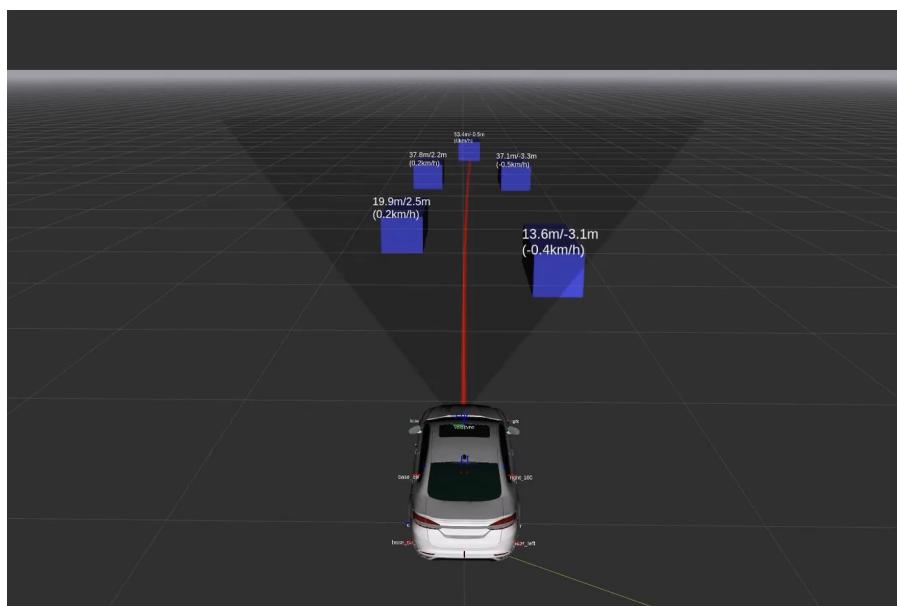
- Aim
 - Introduce the topic of multi-sensor perception and sensor fusion
 - Learn about Kalman filtering applied to sensor fusion
 - Devise a sensor fusion algorithm for position estimation (low-level fusion)
- Readings
 - F. Gustafsson. Statistical Sensor Fusion. 2010.
 - D. Simon. Optimal State Estimation: Kalman, H_∞ , and Nonlinear Approaches. 2006.

Multi-sensor approach

- Localization
- Environment



Multi-sensor perception



Modeling the environment

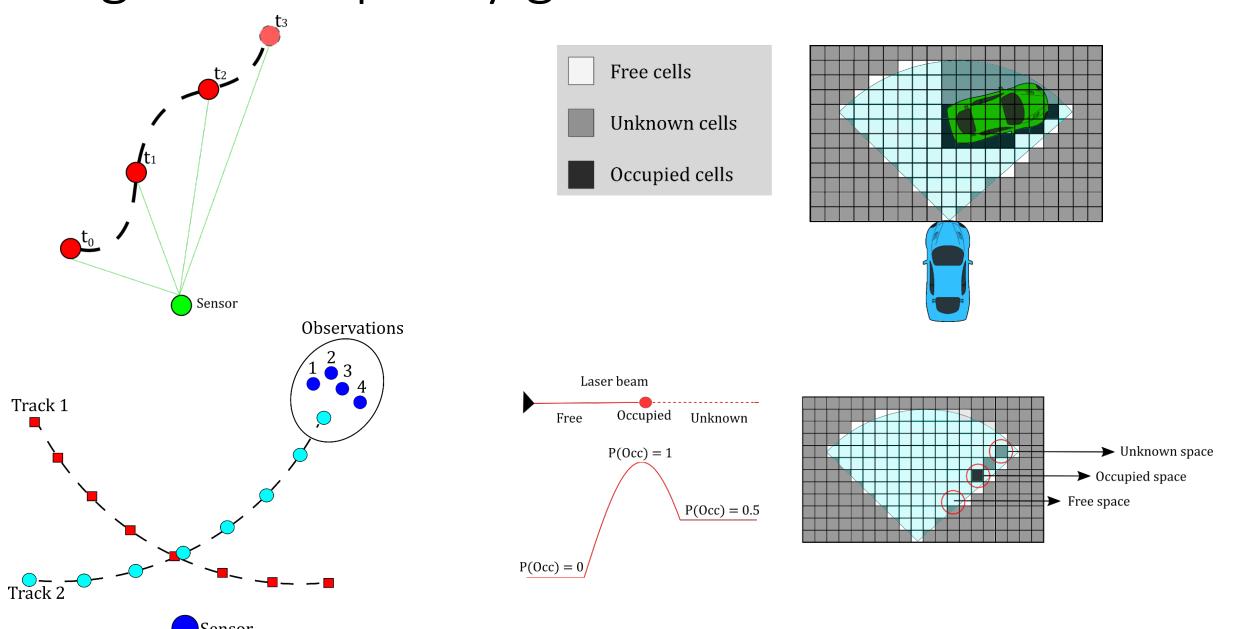
- Two types of algorithms are typically used (**multiple sensors**)
 - Object tracking algorithms
 - Occupancy grid algorithms
- Goal of object tracking algorithms
 - to determine the list of objects, which are currently present in the environment
 - to estimate their state variables
- Occupancy grid approach
 - we describe the environment in a form of a discrete grid with certain height and width of the cells (fixed resolution step size)
 - each cell has a probability that it is occupied (or not), defined by sensor observations

11/24/22

Lecture 16

5

Tracking vs. occupancy grids



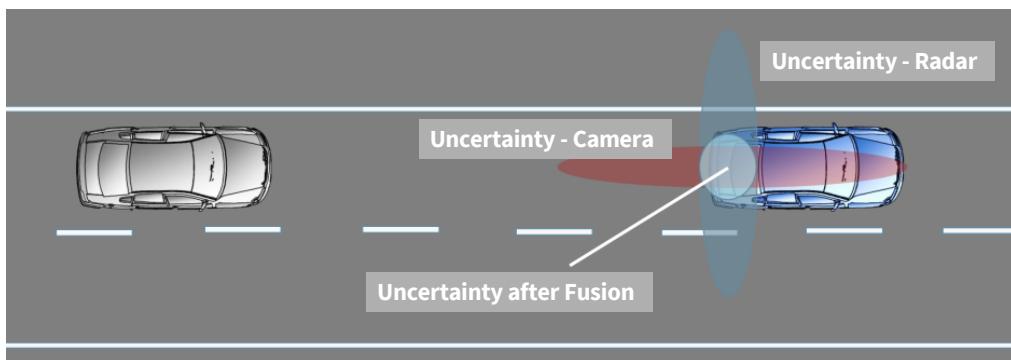
11/24/22

Lecture 16

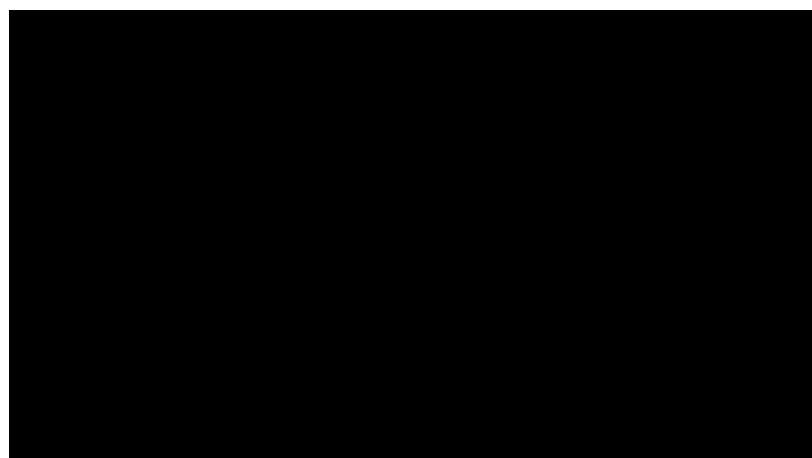
6

Multi-sensor perception

- Uncertainty reduction



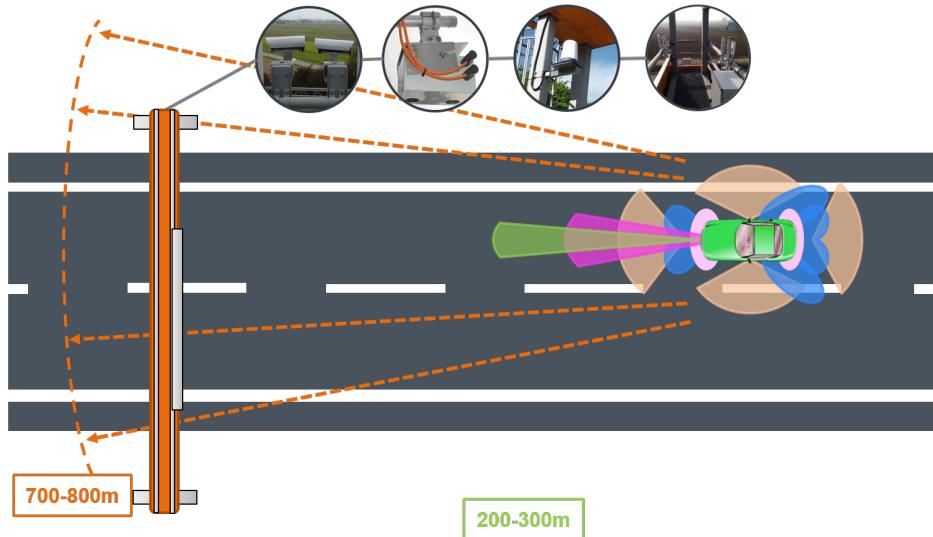
Multi-sensor perception / tracking



Sensor fusion of camera and long-range radar

[Source: Baselabs, 2017]

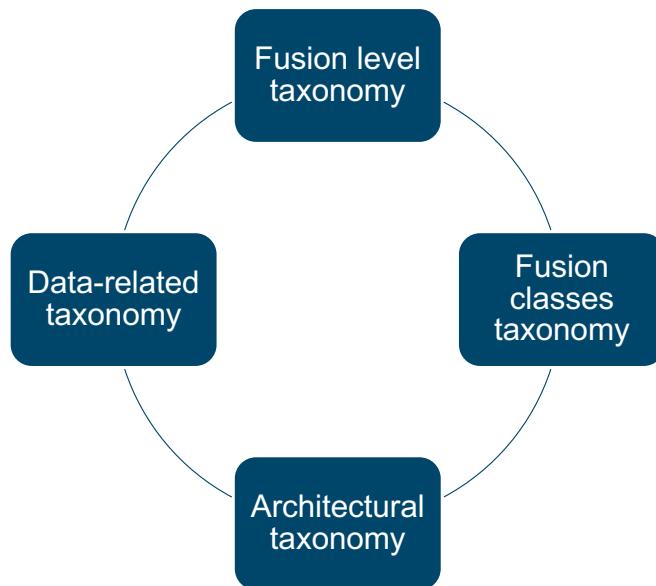
Using stationary sensors



Single-sensor vs multi-sensor perception

- Drawbacks of single-sensor perception
 - Limited range and field of view
 - Performance is susceptible to common environmental conditions
 - Range determination is not as accurate as required
 - Detection of artefacts, so-called false positives
- Multi-sensor perception might compensate these, and provide:
 - Increased classification accuracy of objects
 - Improved state estimation accuracy
 - Improved robustness for instance in adverse weather conditions
 - Increased availability
 - Enlarged field of view

Sensor fusion taxonomies



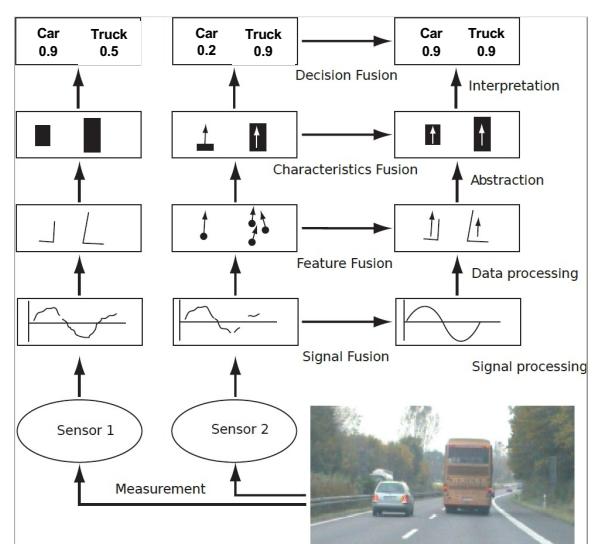
11/24/22

Lecture 16

11

Fusion level taxonomy

- Fusion is typically divided into three levels of abstraction:
 - Low-level fusion
 - Intermediate-level fusion
 - High-level fusion
- They respectively fuse:
 - Signals
 - Features and characteristics
 - Decisions



Schematic depiction of fusion levels (Stüker, Heterogene Sensordatenfusion zur robusten Objektverfolgung im automobilen Straßenverkehr, 2016)

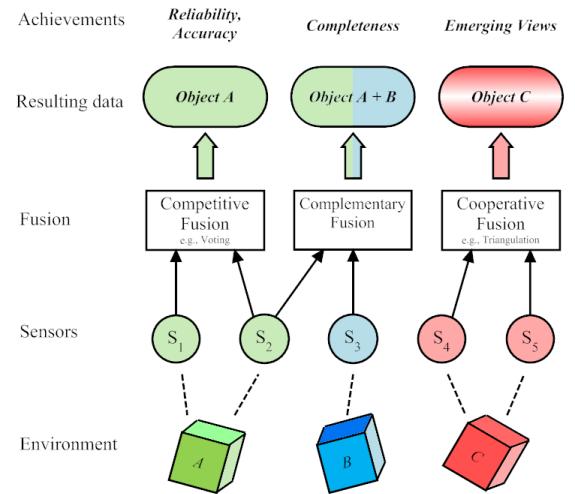
11/24/22

Lecture 16

12

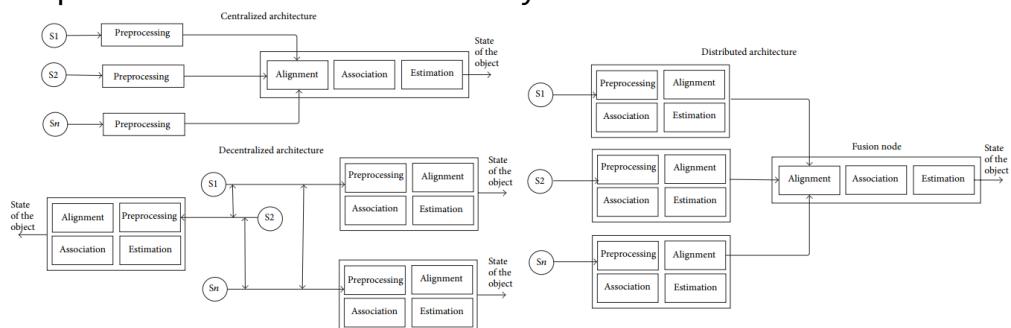
Fusion class taxonomy

- **Competitive fusion**
 - is used when redundant sensors measure the same quantity, in order to reduce the overall uncertainty
- **Complementary fusion**
 - is used when sensors provide a complementary information about the environment, for instance distance sensors with different ranges
- **Cooperative fusion**
 - is used when the required information can not be inferred from a single sensor (e.g. GPS localization and stereo vision)



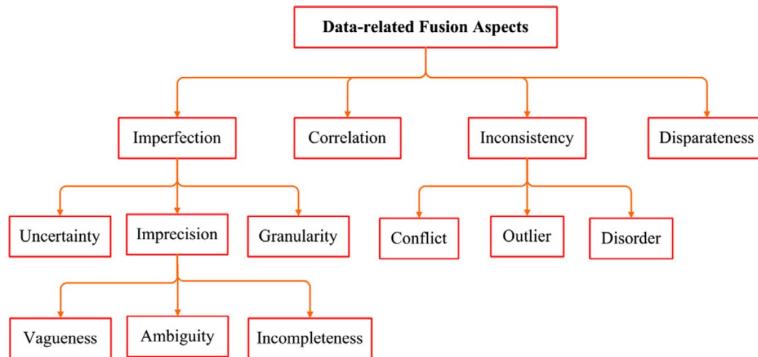
Architectural taxonomy

- The **centralized** architecture is theoretically optimal, but scales badly with respect to communication and processing
- The **decentralized** architecture is a collection of autonomous centralized systems, and has the same scaling issues
- The **distributed** architecture scales better, but can lead to information loss because each sensor processes its information locally



Data-related taxonomy

- The most interesting data-related fusion aspect is the inherent imperfection of the sensory data
- The data-related taxonomy provides us with a checklist of underlying data issues and how to deal with them



Data-related taxonomy

- Sensory data makes a statement about the environment
 - "The distance to the nearest car is 35.12 m"
- Due to the inherent data imprecision, we have to deal with:
 - **Uncertainty:** The distance to the nearest car is more than 20 m with 80% probability
 - **Vagueness:** The distance to the nearest car is more than 20 m with 80% probability, and we are 90% confident in this statement
 - **Ambiguity**
 - **Incompleteness**
- The underlying data can contain multiple imperfections at once

Bayesian statistics in multi-sensor data fusion

- **Basic premise:** all unknowns are treated as random variables and the knowledge of these quantities is summarized via a probability distribution
 - This includes the observed data, any missing data, noise, unknown parameters, and models
- Bayesian statistics provides
 - a framework for **quantifying objective and subjective uncertainties**
 - principled methods for **model estimation and comparison** and the **classification of new observations**
 - a **natural way to combine different sensor observations**
 - principle methods for dealing **with missing information**

Sensor fusion – a simple example

- **Problem:** determine the distance to n objects using measurements from two sensors
- Assumptions:
 - Both sensors have the same field of view
 - First sensor has a higher precision than the second sensor
 - Consider the simplest case ($n=1$)
- How to fuse these measurements properly?

Sensor fusion – a simple example

- Sensors provide redundant measurements of the same physical quantity (distance)
- To incorporate the precision information → measurements are assumed to be **normally distributed random variables**
- Specifically, the univariate Gaussian distributions are:

$$d_1(x) = (2\pi\sigma_1^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \frac{(x - \mu_1)^2}{\sigma_1^2}\right) \sim \mathcal{N}(\mu_1, \sigma_1^2)$$

$$d_2(x) = (2\pi\sigma_2^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \frac{(x - \mu_2)^2}{\sigma_2^2}\right) \sim \mathcal{N}(\mu_2, \sigma_2^2)$$

Sensor fusion – a simple example

- Assumption from before:
 - First sensor has a higher precision than the second sensor
- This can be captured as: $\sigma_1^2 < \sigma_2^2$
- Problem is to find $d(x) \sim \mathcal{N}(\mu, \sigma^2)$
- The idea is to combine the previous Gaussian distributions

$$d(x) = d_1(x) \cdot d_2(x) = (4\pi^2\sigma_1^2\sigma_2^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \left(\frac{(x - \mu_1)^2}{\sigma_1^2} + \frac{(x - \mu_2)^2}{\sigma_2^2} \right)\right)$$

Sensor fusion – a simple example

- Re-arranging the expression in the exponent and dividing the numerator and denominator by $(\sigma_1^2 + \sigma_2^2)$:

$$\begin{aligned} -\frac{1}{2} \left(\frac{(x - \mu_1)^2}{\sigma_1^2} + \frac{(x - \mu_2)^2}{\sigma_2^2} \right) &= -\frac{1}{2} \frac{(\sigma_1^2 + \sigma_2^2)x^2 - 2(\sigma_2^2\mu_1 + \sigma_1^2\mu_2)x + (\sigma_2^2\mu_1^2 + \sigma_1^2\mu_2^2)}{\sigma_1^2\sigma_2^2} \\ &= -\frac{1}{2} \frac{x^2 - 2\frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}x + \frac{\mu_1^2\sigma_2^2 + \mu_2^2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}} \end{aligned}$$

- To obtain an expression of form $x^2 - 2\mu x + \mu^2 = (x - \mu)^2$ in the numerator, it is necessary to add and subtract the square of the second term

Sensor fusion – a simple example

$$-\frac{1}{2} \frac{x^2 - 2\frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}x + \left(\frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}\right)^2 - \left(\frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}\right)^2 + \frac{\mu_1^2\sigma_2^2 + \mu_2^2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}}$$

- The expression in the exponent becomes

$$-\frac{1}{2} \frac{(x - \mu)^2 - \mu^2 + s}{\sigma^2} = -\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2} + \frac{\mu^2 - s}{2\sigma^2}$$

Sensor fusion – a simple example

- Putting everything together leads to the final distribution which represents the fused information

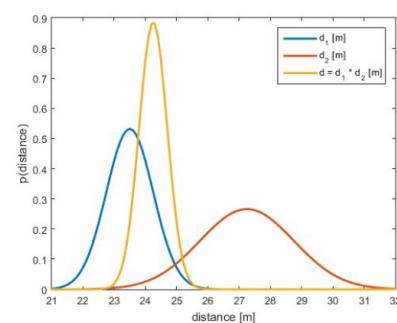
$$\begin{aligned}d(x) &= (2\pi\sigma_1\sigma_2)^{-1} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2} + \frac{\mu^2-s}{2\sigma^2}\right) \\&= (2\pi\sigma_1\sigma_2)^{-1} \exp\left(\frac{\mu^2-s}{2\sigma^2}\right) \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right) \\&= C \cdot \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)\end{aligned}$$

Sensor fusion – a simple example

- Mean value and variance are

$$\mu = \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}$$

$$\sigma^2 = \frac{\sigma_1^2 \cdot \sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$



- The fused value is the **weighted average** of the measurements
- The **weighting favors the sensor with higher precision**
- The overall **uncertainty decreases**

Kalman filter (KF) – again

- Assumption #1: linear dynamics

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

- i.i.d .process noise ϵ_t is $\mathcal{N}(0, R_t)$

- Assumption #1 implies that the probabilistic generative model is Gaussian

$$p(x_t | u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}} \exp \left(-\frac{1}{2} (x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1} (x_t - A_t x_{t-1} - B_t u_t) \right)$$

Kalman filter (KF)

- Assumption #2: linear measurement model

$$z_t = C_t x_t + \delta_t$$

- i.i.d. measurement noise δ_t is $\mathcal{N}(0, Q_t)$

- Assumption #2 implies that the measurement probability is Gaussian

$$p(z_t | x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp \left(-\frac{1}{2} (z_t - C_t x_t)^T Q_t^{-1} (z_t - C_t x_t) \right)$$

Kalman filter (KF)

- Assumption #3: the initial belief is Gaussian

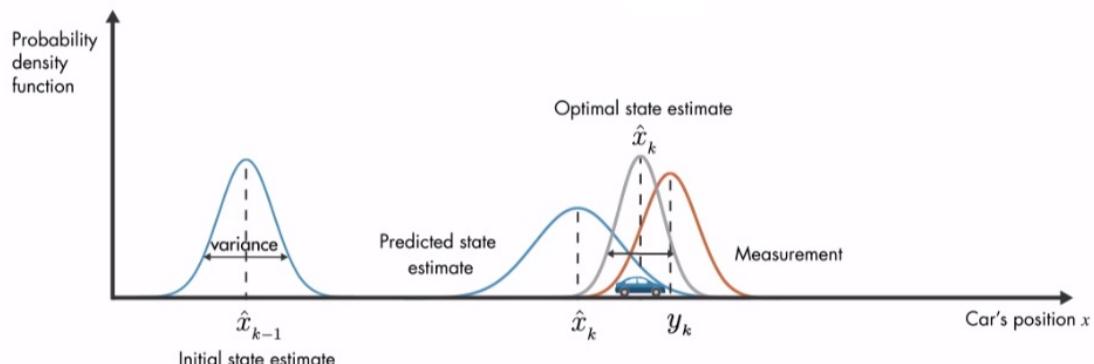
$$bel(x_0) = p(x_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1} (x_0 - \mu_0)\right)$$

- Key fact:** These three assumptions ensure that the posterior $bel(x_t)$ is Gaussian for all t , i.e., $bel(x_t) = \mathcal{N}(\mu_t, \Sigma_t)$

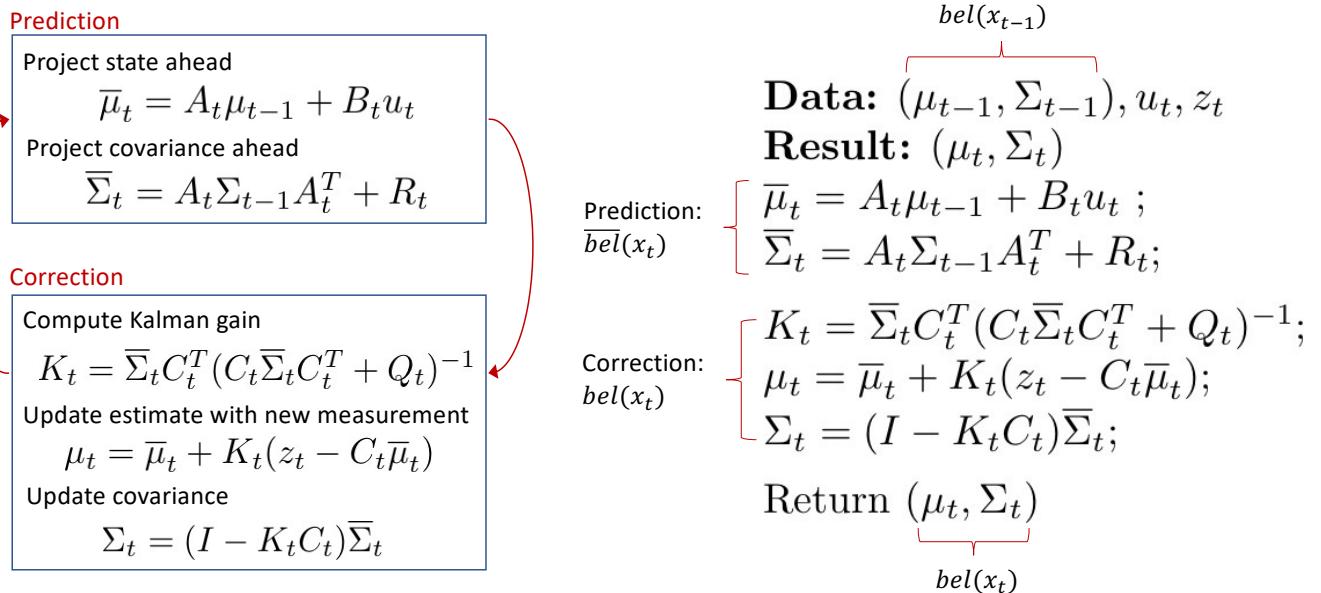
- Note:
 - KF implements a belief computation for continuous states
 - Gaussians are unimodal → commitment to single-hypothesis filtering

Kalman filter (KF) – pose estimation

- The KF is optimal for linear Gaussian systems (minimum variance)
- But: reality is typically nonlinear
- KF is computationally very efficient



Kalman filter: algorithm revisited



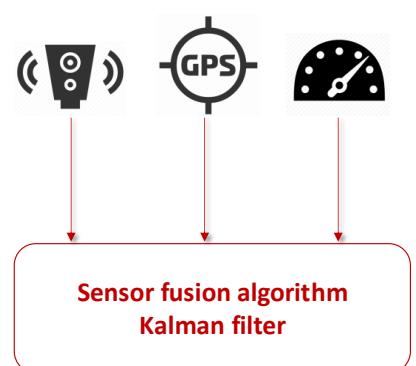
11/24/22

Lecture 16

29

Sensor fusion example

- **Problem:** Estimate position, velocity, and acceleration of a vehicle from noisy position and acceleration measurements
- **Assumptions:**
 - Single track model for the vehicle
 - Lidar provides position measurements with low precision
 - GPS provides position measurements with high precision
 - IMU provides acceleration measurements
- Sensor fusion is done using the **Kalman filter**



11/24/22

Lecture 16

30

Sensor fusion example: Motion model

- **State vector:** $\mu_t = [p \ v \ a]^T$
- Change of the state over time is captured by the **motion model**

$$\begin{aligned} p_t &= p_{t-1} + T_s v_{t-1} + \frac{T_s^2}{2} a_{t-1} + \epsilon_{pt} \\ v_t &= v_{t-1} + T_s a_{t-1} + \epsilon_{vt} \\ a_t &= a_{t-1} + \epsilon_{at} \end{aligned}$$

- T_s represents sampling time

Sensor fusion example: Motion model

- The motion model can be represented in matrix form

$$\underbrace{\begin{bmatrix} p \\ v \\ a \end{bmatrix}}_t = \underbrace{\begin{bmatrix} 1 & T_s & \frac{T_s^2}{2} \\ 0 & 1 & T_s \\ 0 & 0 & 1 \end{bmatrix}}_{\text{State transition matrix}} \underbrace{\begin{bmatrix} p \\ v \\ a \end{bmatrix}}_{t-1} + \underbrace{\begin{bmatrix} \epsilon_p \\ \epsilon_v \\ \epsilon_a \end{bmatrix}}_t$$

$$\mu = A_t \mu_{t-1} + \epsilon_t$$

where ϵ_t is independent process noise distributed as $\mathcal{N}(0, R_t)$

Sensor fusion example: Measurement model

- The **measurement model** defines a mapping from the state space to the measurement space
- For this example, two possible fusion scenarios will be considered:
 1. Lidar + IMU
 2. Lidar + GPS + IMU
- In the first scenario, only measurements from Lidar and IMU are available
 - Assumption: Lidar provides low precision measurements (noisy data)
- In the second scenario, high precision GPS measurements are also available

Sensor fusion example: Measurement model

- First scenario – measurement model is given by

$$\begin{bmatrix} p_{lidar} \\ a_{imu} \end{bmatrix}_t = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Measurement matrix}} \underbrace{\begin{bmatrix} p \\ v \\ a \end{bmatrix}}_t + \underbrace{\begin{bmatrix} \delta_{lidar} \\ \delta_{imu} \end{bmatrix}}_{\text{Measurement noise}}$$

$$z_t = C_t \mu_t + \delta_t$$

where δ_t is independent measurement noise distributed as $\mathcal{N}(0, Q_t)$

Sensor fusion example: Initialization

- Choosing the **initial state vector μ_0** - depends on available information

- If there is *a-priori* knowledge – initialization is done with known values
- If there is a lack of information – initial state is chosen to be zero
- For this example the initial state vector is set to zero

$$\mu_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- Choosing the **initial covariance matrix Σ_0** - should be defined based on the initialization error

- If the initial state is not very close to the correct state - Σ_0 will have large values
- If the initial state is close to the correct state - Σ_0 will have small values

$$\Sigma_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

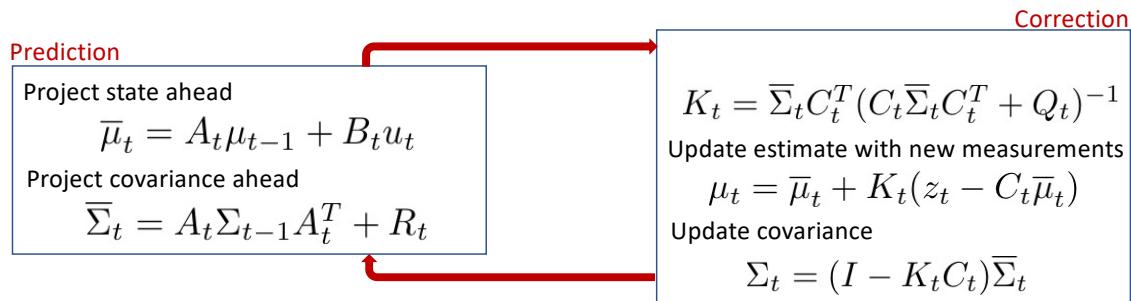
Sensor fusion example: Noise model tuning

- The **process noise covariance matrix R_t** - describes the confidence in the system model
 - Small values indicate higher confidence – predicted values are more weighted
 - Large values indicate lower confidence – measurements become dominant
- The **measurement noise covariance matrix Q_t** - describes the confidence in the measurements
 - Has a similar interpretation as R_t
- Both matrices need to be symmetric and positive definite

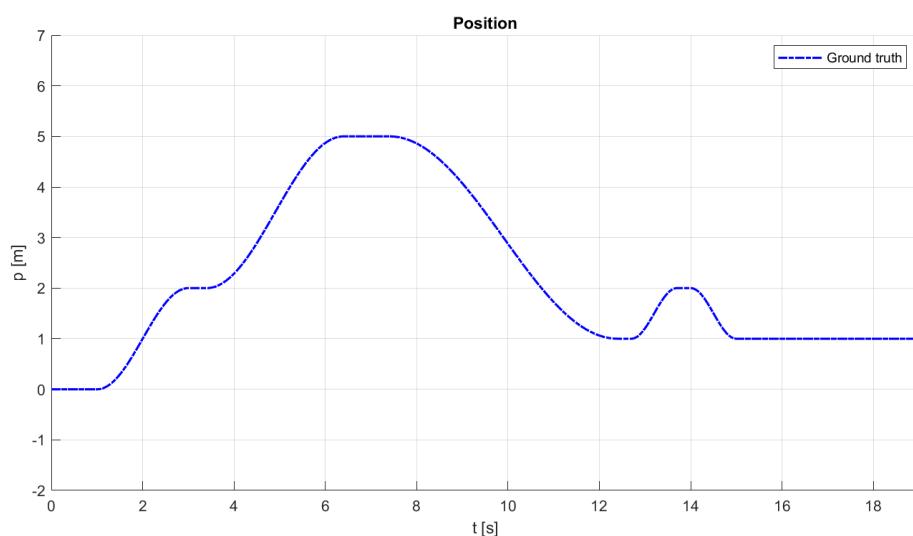
$$R_t = \begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.05 \end{bmatrix} \quad Q_t = \begin{bmatrix} \sigma_{lidar}^2 & 0 \\ 0 & \sigma_{imu}^2 \end{bmatrix}$$

Sensor fusion example: Algorithm

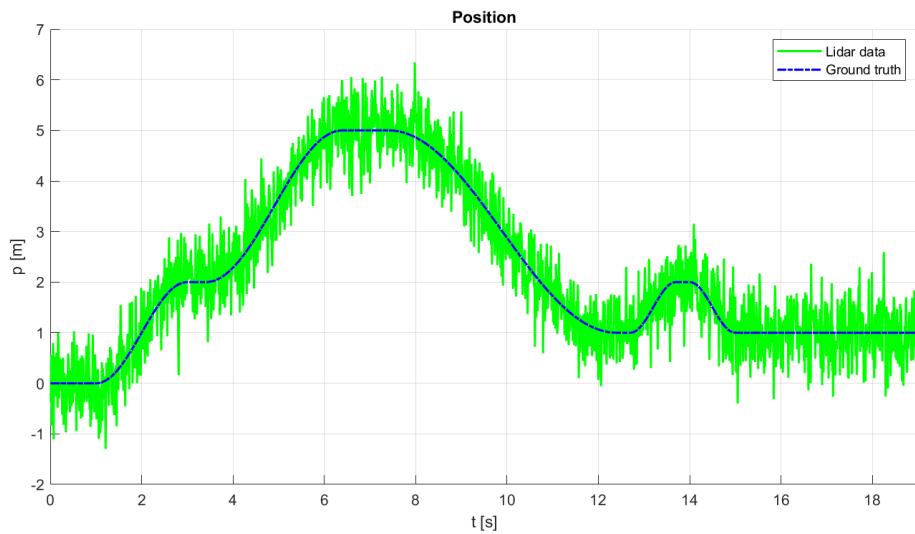
- Estimation results are obtained using the prediction-correction scheme



Sensor fusion example: Position estimation



Sensor fusion example: Position estimation

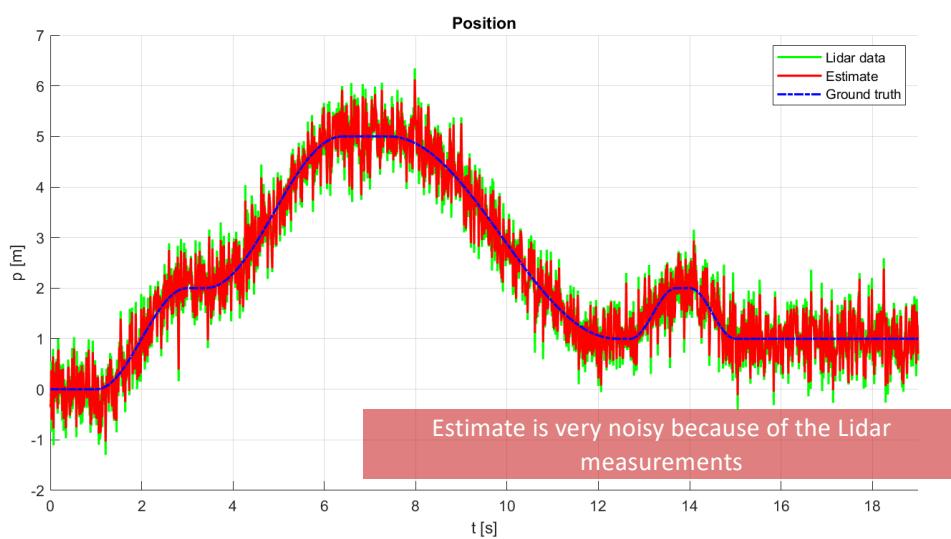


11/24/22

Lecture 16

39

Sensor fusion example: Position estimation

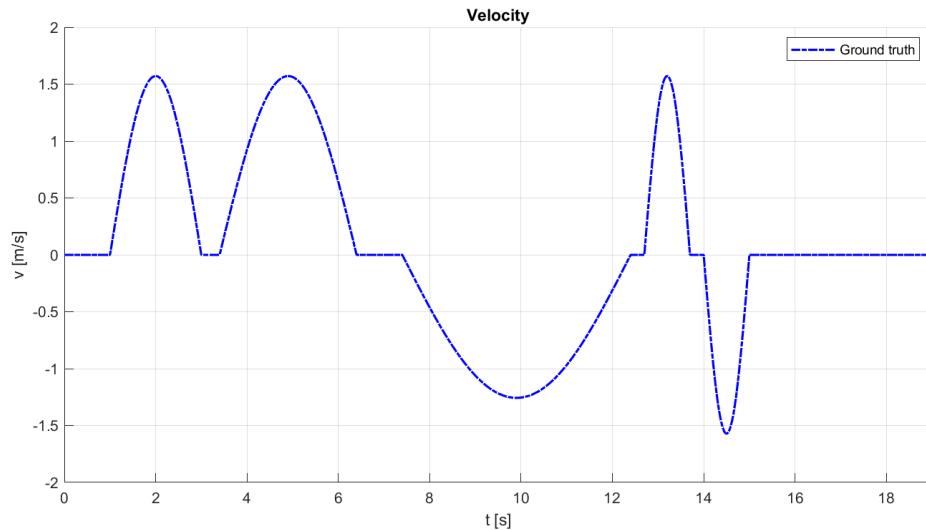


11/24/22

Lecture 16

40

Sensor fusion example: Velocity estimation

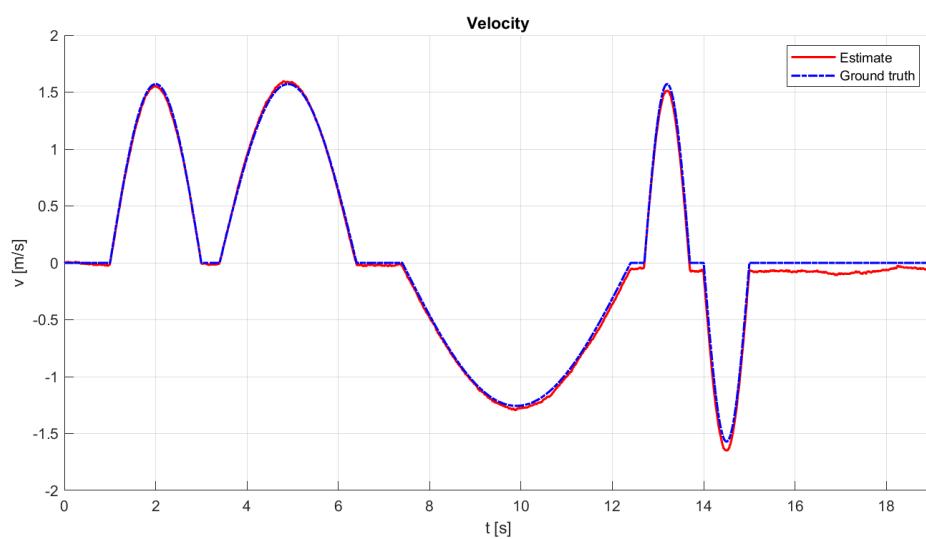


11/24/22

Lecture 16

41

Sensor fusion example: Velocity estimation

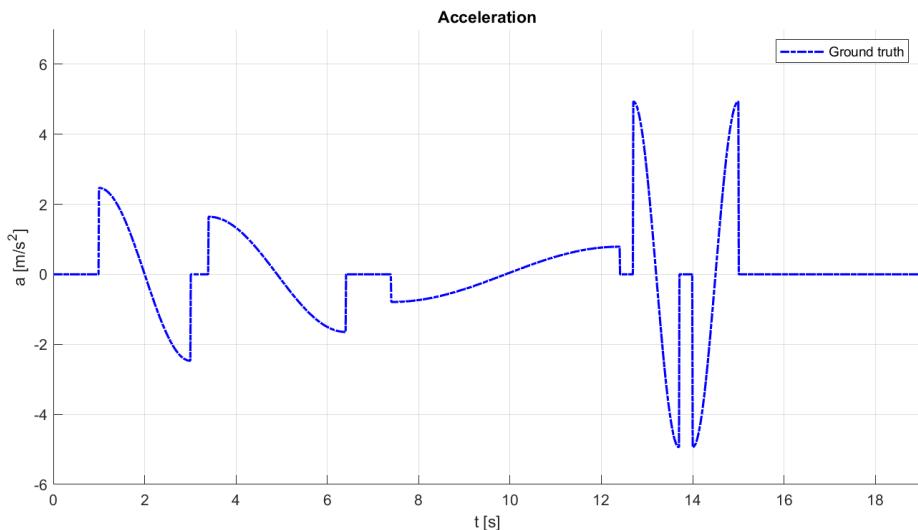


11/24/22

Lecture 16

42

Sensor fusion example: Acceleration estimation

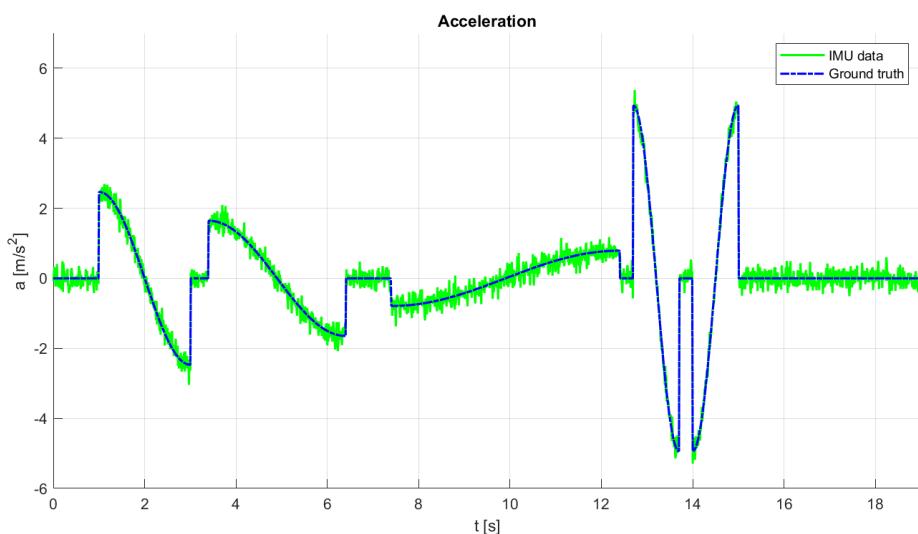


11/24/22

Lecture 16

43

Sensor fusion example: Acceleration estimation

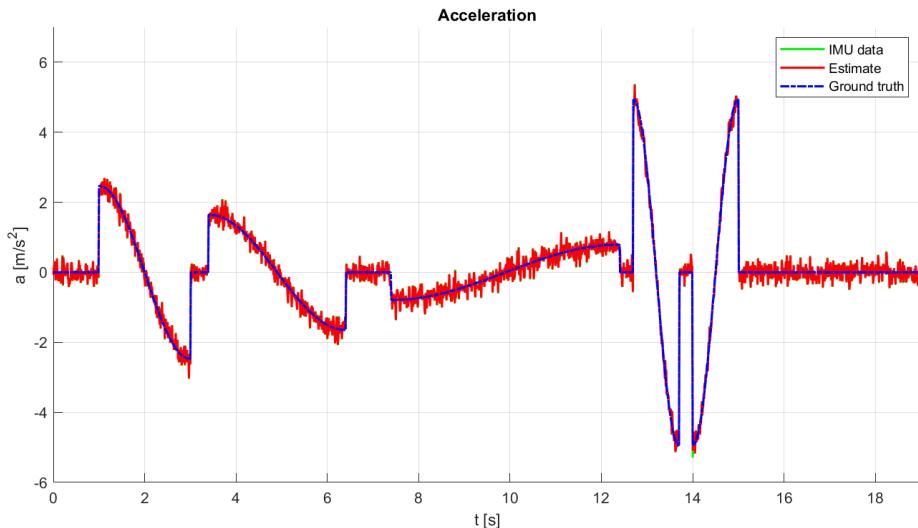


11/24/22

Lecture 16

44

Sensor fusion example: Acceleration estimation



Sensor fusion example: Measurement model

- In the previous scenario – the position estimate is quite noisy (because of the low precision of the Lidar measurements)
- Therefore, in the second scenario, position is measured with Lidar and GPS

$$\begin{bmatrix} p_{lidar} \\ p_{gps} \\ a_{imu} \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ v \\ a \end{bmatrix}_t + \begin{bmatrix} \delta_{lidar} \\ \delta_{gps} \\ \delta_{imu} \end{bmatrix}_t$$

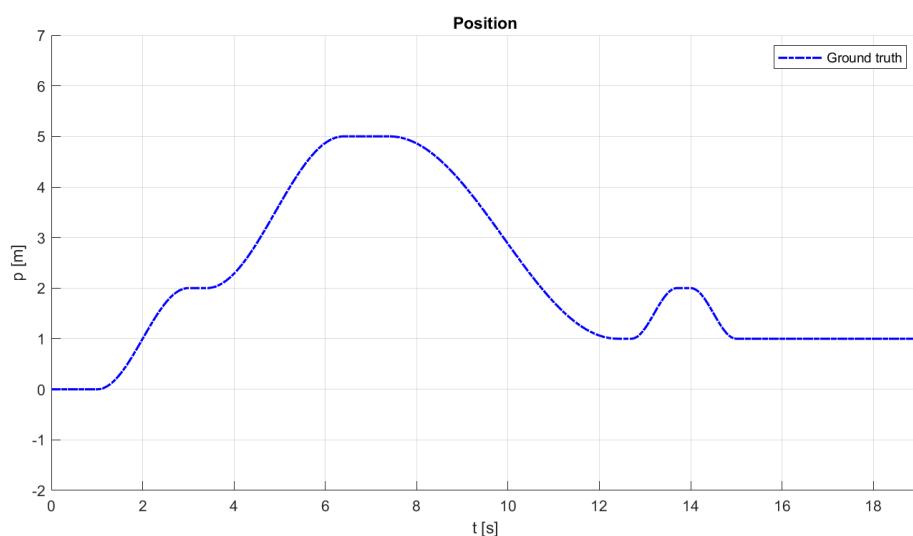
$$z_t = C_t \mu_t + \delta_t$$

Sensor fusion example: Noise model tuning

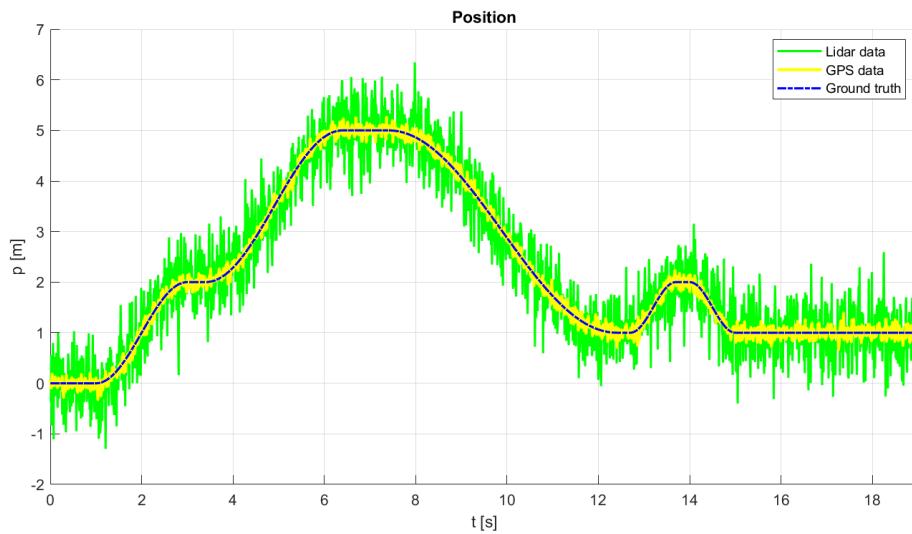
- The measurement noise covariance matrix Q_t for this scenario has an additional GPS variance

$$Q_t = \begin{bmatrix} \sigma_{lidar}^2 & 0 & 0 \\ 0 & \sigma_{gps}^2 & 0 \\ 0 & 0 & \sigma_{imu}^2 \end{bmatrix} = \begin{bmatrix} 0.5^2 & 0 & 0 \\ 0 & 0.1^2 & 0 \\ 0 & 0 & 0.2^2 \end{bmatrix}$$

Sensor fusion example: Position estimation



Sensor fusion example: Position estimation

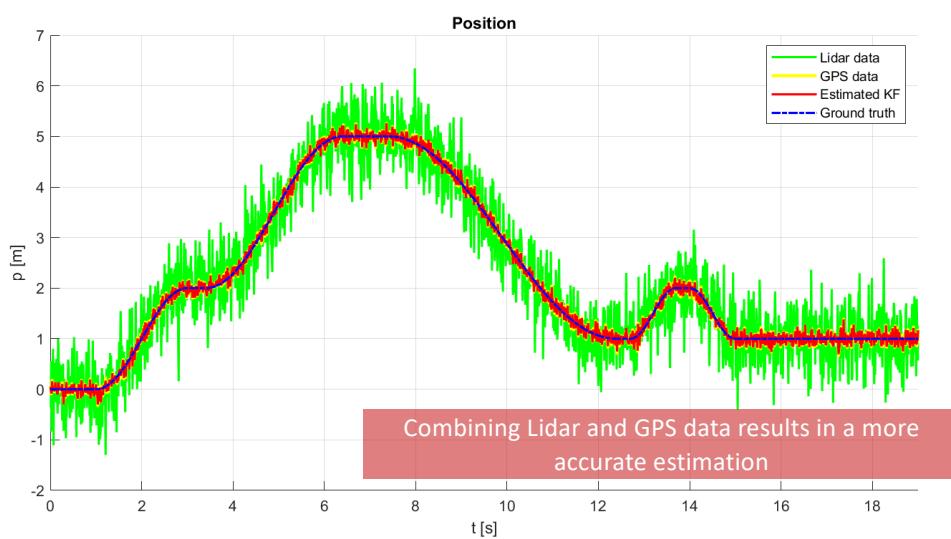


11/24/22

Lecture 16

49

Sensor fusion example: Position estimation

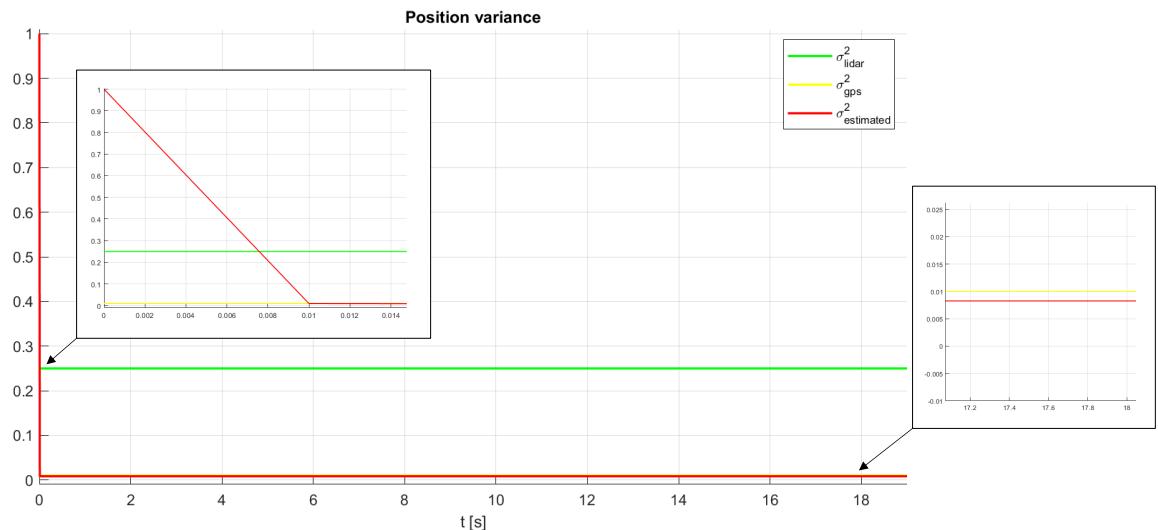


11/24/22

Lecture 16

50

Sensor fusion example: Position variance



11/24/22

Lecture 16

51

Sensor fusion example: Conclusion

- Problem: Vehicle state estimation using Kalman filter
- The example pointed out:
 - How to create a motion model and a measurement model
 - How to fuse the data from different types of sensors
 - How to set the initial state vector and the initial covariance matrix
 - How to chose appropriate values for process noise and measurement noise covariance matrices
 - How to achieve a more accurate state estimation by adding more sensors
 - How fusion of data decreases the overall estimation variance



11/24/22

Lecture 16

52

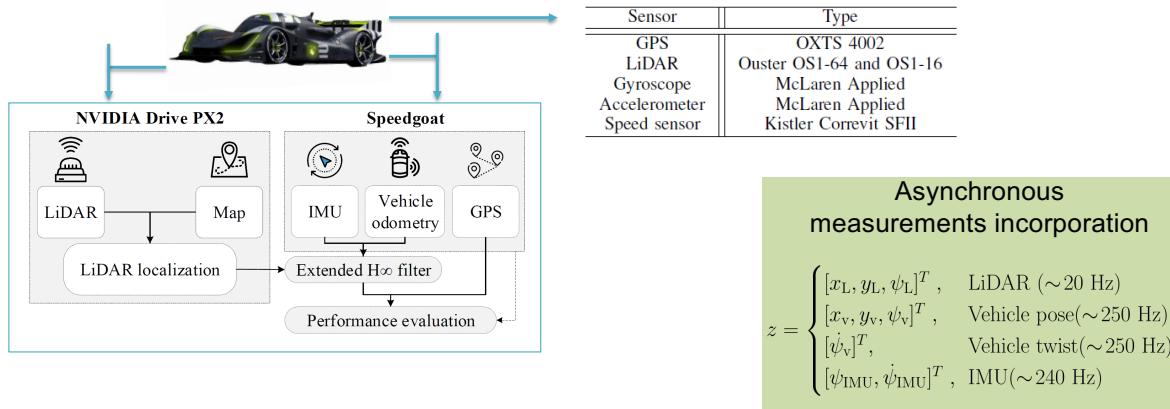
Useful trick

- Augment the state vector with some auxiliary states and then apply the KF to the augmented state space model
- What can we handle?
 - Colored state noise
 - Colored measurement noise
 - Sensor offset and drifts
 - Sensor faults (sudden offset)
 - Actuator fault (sudden offset)

Common problems in multi-sensor data fusion

- **Registration:** Coordinates (both time and space) of different sensors or fusion agents must be aligned.
- **Bias:** Even if the coordinate axis are aligned, due to the transformations, biases can result. These have to be compensated.
- **Correlation:** Even if the sensors are independently collecting data, processed information to be fused can be correlated.
- **Data association:** multi-target tracking problems introduce a major complexity to the fusion system.
- **Out-of-sequence measurements:** Due to delayed communications between local agents, measurements belonging to a target whose more recent measurement has already been processed, might arrive to a fusion center.
- ...

Example: Asynchronous measurements



Allows to incorporate sensors with different update rates correctly.

Vehicle motion model:
explicit dependence on the sampling time Δt

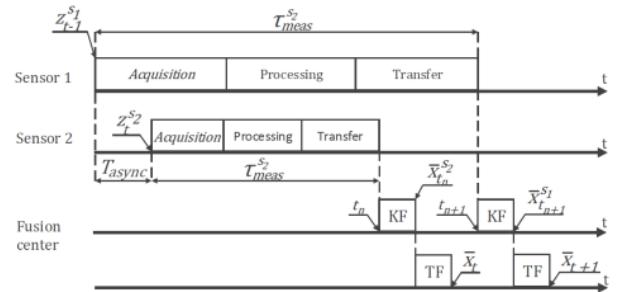
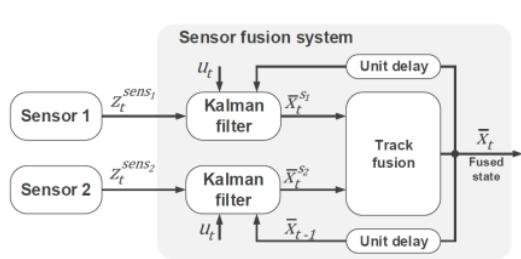
11/24/22

Lecture 16

55

Example: Out-of-sequence measurements

- Might lead to incorrect temporal order, which in turn causes a negative time measurement update (NTMU) in the fusion algorithm (e.g., EKF).
- As a result, the process of sensor fusion is not performed correctly.
- A wrong representation of the environment is created!



[Source: A. Mehmed, Runtime monitoring of automated driving systems, 2019]

11/24/22

Lecture 16

56

Example: Out-of-sequence measurements

- Timestamping data at arrival (Centralized Method)
 - Measurement cycle time $T_c=1/\text{fps}$
- Timestamping at the time of acquisition (Distributed Method)
 - Global time is needed
- Triggering method (by external source)

11/24/22

Lecture 16

57

Sensor fusion using the Autoware stack



AUTOWARE.AI



AUTOWARE.AUTO



AUTOWARE.IO

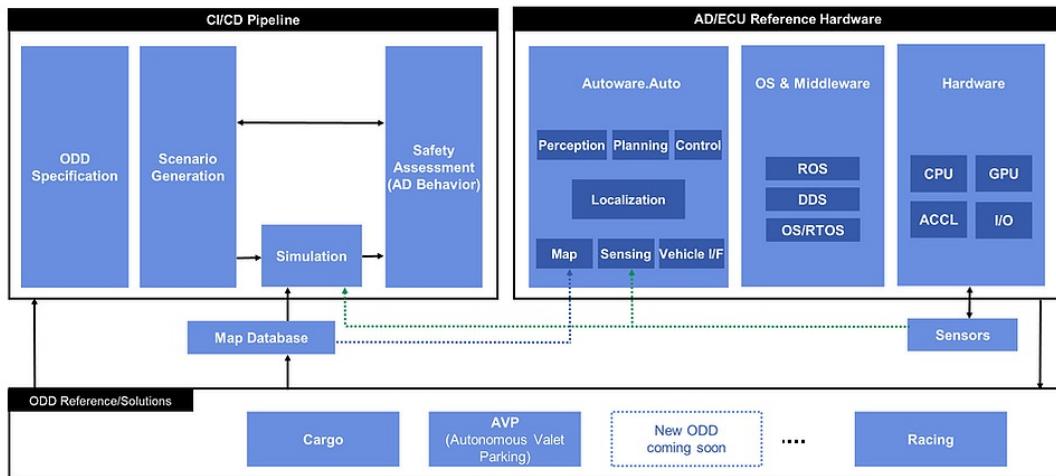
Autoware.org, 2021

11/24/22

Lecture 16

58

Sensor fusion using the Autoware stack



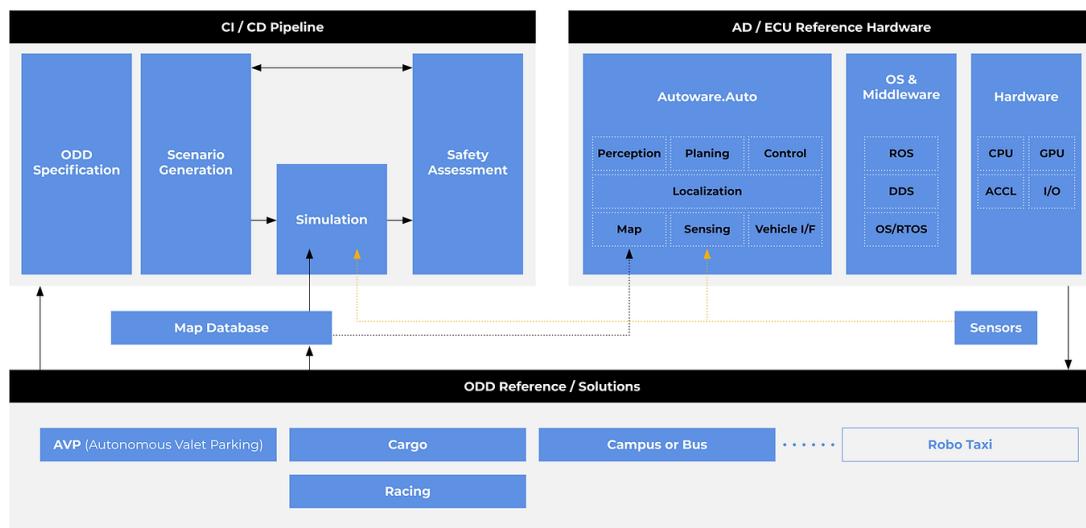
Autoware.org, 2021

11/24/22

Lecture 16

59

Eco-system around the driving stack



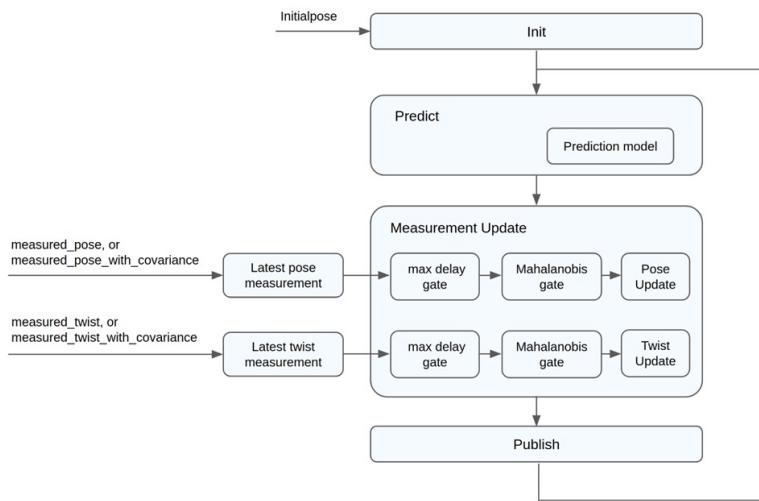
Autoware.org, 2022

11/24/22

Lecture 16

60

Localization using the EKF



https://gitlab.com/autowarefoundation/autoware.ai/core_perception/tree/master/ekf_localizer

Live demo / Autoware

1. Localization with odometry only (IMU)
2. Localization with GNSS without noise
3. Localization with GNSS with noisy data
4. Localization with GNSS with noise and bias
5. Localization with Lidar
 - parameter tuning
 - Lidar pose has an unknown time delay and unknown noise

Principles of Robot Autonomy I

Multi-sensor perception and sensor fusion II

Daniel Watzenig



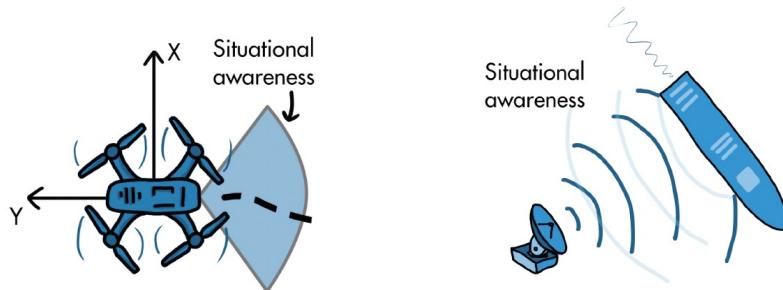
Today's lecture

- Aim
 - Introduce the topic of sensor fusion for multi-sensor perception
 - Learn about fundamental ideas of multi-object tracking for autonomous systems
 - Sensor fusion of radar and visual data (high-level fusion)
- Topics
 - Introduction: Understanding tracking filters, measurement noise, and process noise
 - Single-object tracking: Using a tracker to determine position and motion of a remote object
 - Multi-object tracking: Overcoming the challenges of tracking several objects at once
- Readings
 - Blackman S. S., and Popoli R., *Design and analysis of modern tracking systems*, 1999.
 - MathWorks, *Multi-Object Tracking for Autonomous Systems and Surveillance Systems*, 2020.

Part 1: Introduction

Perception – critical component of autonomous system

Multi-object tracking and **sensor fusion** are core of a perception system



The system needs to be able to maintain situational awareness.

Part 1: Introduction

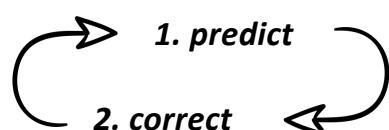
Multi-object tracking - the core is the ability to estimate the motion of each object separately



Estimation filters - different types are used in tracking

- the most fundamental and simple filter is the **Kalman filter**

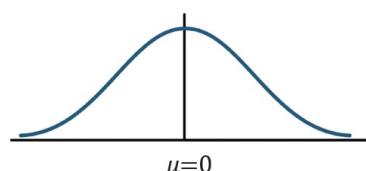
Kalman filter - uses a two-step process to estimate state:



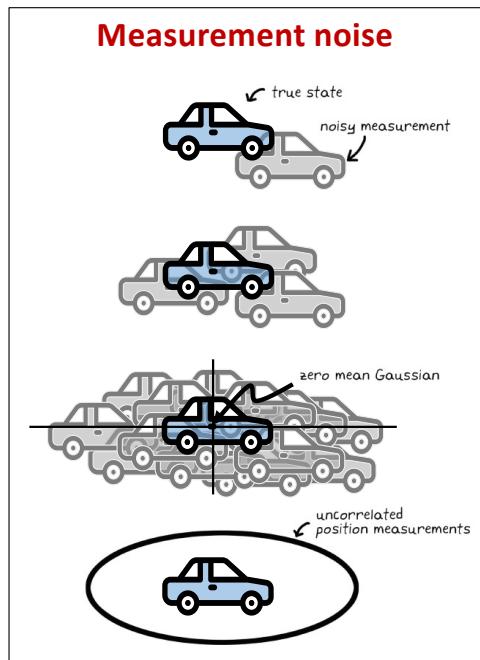
1. Linear system model: $x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$

$$z_t = C_t x_t + \delta_t$$

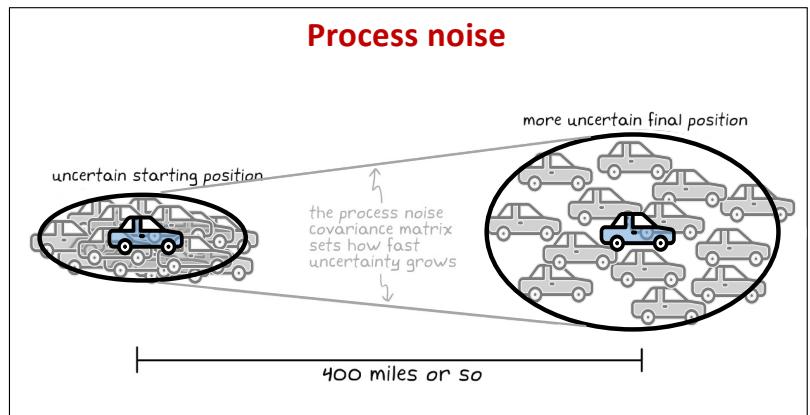
2. Gaussian noise distribution:



Part 1: Introduction



11/28/22



Combining prediction and measurement to get more accurate and more reliable estimates of a system state.

Lecture 17

5

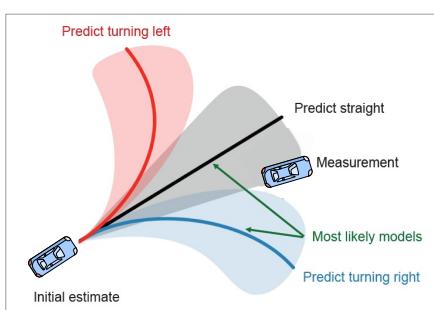
Part 2: Single-object tracking



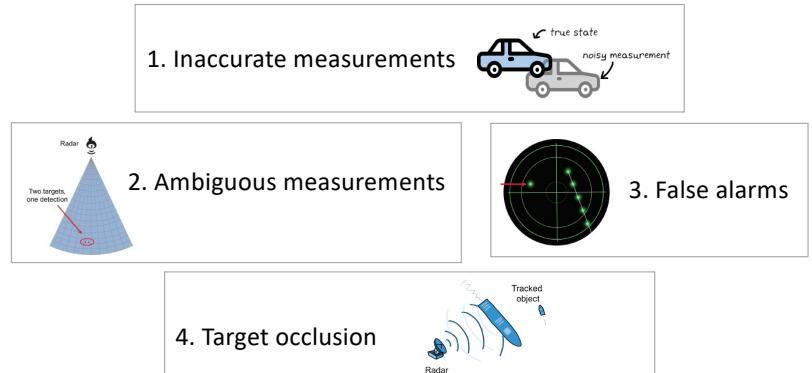
Task is to determine state (e.g., position or velocity) by fusing the results from sensors and models

Tracking becomes more challenging:

a) Predicting the state of a tracked object



b) Challenges in remote measurements



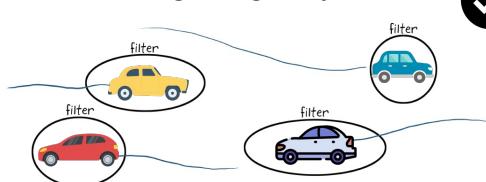
11/28/22

Lecture 17

6

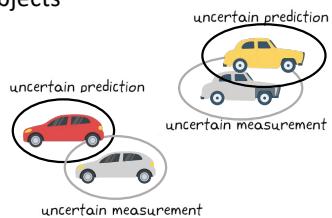
Part 3: Multi-object tracking

Is the tracking of multiple objects at the same time tougher than tracking a single object?



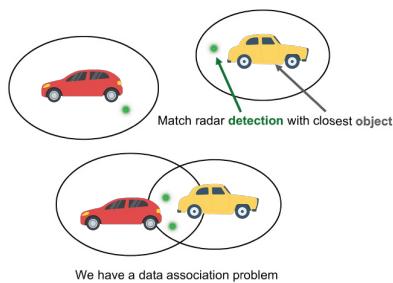
The difficulty of multi-object tracking

1. Uncertainties in predictions and in measurements of the objects

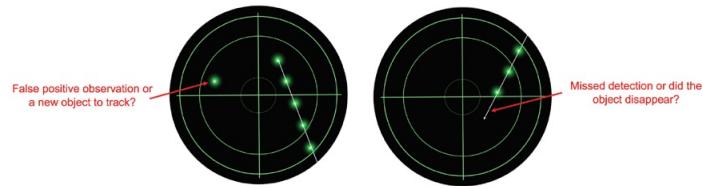


11/28/22

2. Data association problem



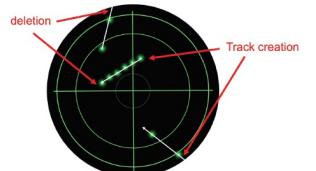
4. Track maintenance due to uncertainties



Lecture 17

7

3. Track maintenance



Part 3: Multi-object tracking

When tracking multiple objects:

- What are the ways to approach the data association problem?
- What are the ways to address the track maintenance problem?

Multi-object tracking flow chart

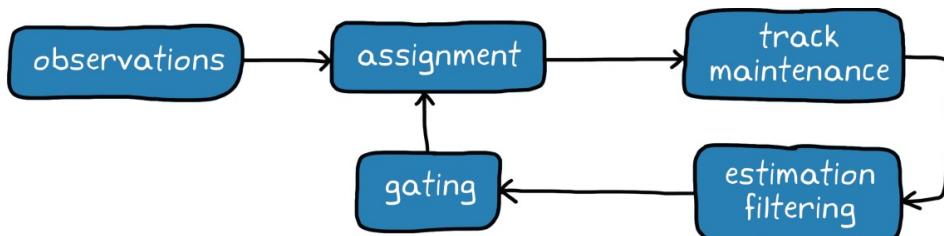


Figure adapted from *Design and Analysis of Modern Tracking Systems* by Samuel Blackman and Robert Popoli (Artech House Radar Library).

11/28/22

Lecture 17

8

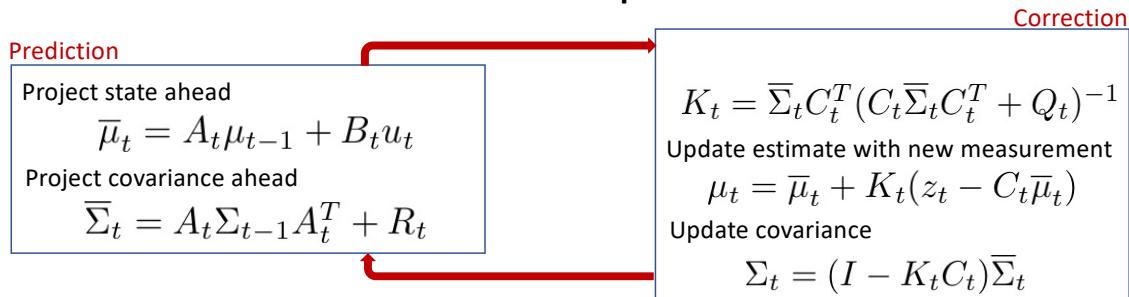
Part 3: Multi-object tracking

Recall: Kalman Filter from previous lectures

Description of the system and the measurement models: $x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$

- Independent process noise ϵ_t is $\mathcal{N}(0, R_t)$
- Independent measurement noise δ_t is $\mathcal{N}(0, Q_t)$

Kalman filter equations



Part 3: Multi-object tracking

Observation or detection occurs when the sensor measures an object

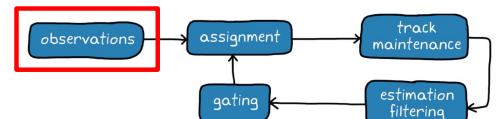
Measurement vector within the Kalman filter structure:

$$z_t = \begin{bmatrix} \text{speed} \\ \text{heading} \\ \dots \end{bmatrix}_t = \begin{bmatrix} 400 \\ 17 \\ \dots \end{bmatrix}_t$$

Measured quantities, e.g., speed or heading



Measured attributes, e.g., color or car type



Useful for improving the track confirmation performance

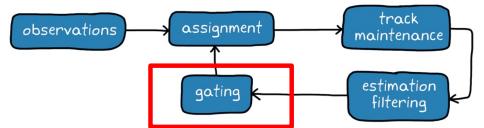
Types of observations



*Note: only point targets will be discussed

Part 3: Multi-object tracking

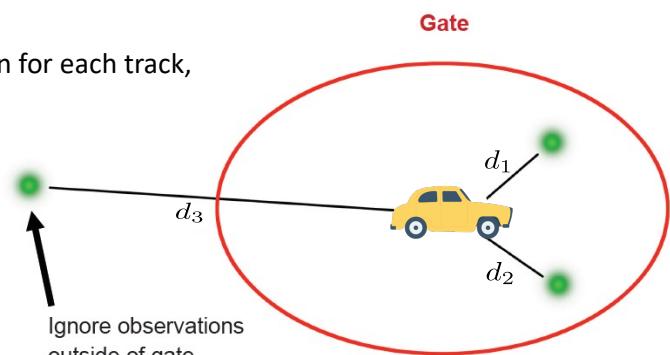
Computational challenge to look at every observation and consider how likely it is to be assigned to every track



Gating - screening mechanism

- ignoring observations outside of a specific region for each track,
- speeds up the assignment process.

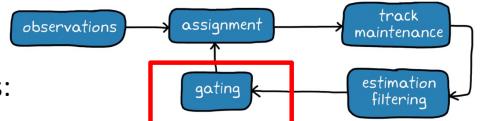
Gating impacts the assignment algorithms – they consider only the observations that are worth looking at.



Part 3: Multi-object tracking

- Residual (or innovation) vector and its covariance matrix are defined as:
$$\tilde{z}_t = z_t - C_t \bar{\mu}_t$$

$$S_t = C_t \bar{\Sigma}_t C_t^T + Q_t$$
- If the measurement is of dimension M , the M -dimensional Gaussian probability density for the residual is:
$$p(\tilde{z}_t) = \det((2\pi)^M S_t)^{-\frac{1}{2}} \exp(-\frac{1}{2} \tilde{z}_t^T S_t^{-1} \tilde{z}_t)$$



1. Rectangular gates

The simplest gating technique – an observation satisfies the gates of a given track if all elements \tilde{z}_l of the residual vector \tilde{z}_t satisfy:

$$|\tilde{z}_l| \leq K_{Gl} \sigma_r$$

➤ Residual standard deviation is defined as:

$$\sigma_r = \sqrt{\sigma_o^2 + \sigma_p^2}$$

Measurement variance Prediction variance (appropriate diagonal element taken from the KF covariance matrix)

➤ Typical choice of rectangular gating coefficients is:

$$K_{Gl} \geq 3.0$$

Part 3: Multi-object tracking

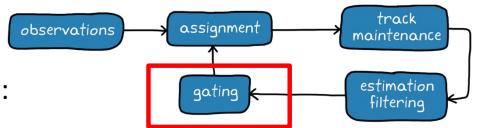
- Residual (or innovation) vector and its covariance matrix are defined as:

$$\tilde{z}_t = z_t - C_t \bar{\mu}_t$$

$$S_t = C_t \bar{\Sigma}_t C_t^T + Q_t$$

- If the measurement is of dimension M , the M -dimensional Gaussian probability density for the residual is:

$$p(\tilde{z}_t) = \det((2\pi)^M S_t)^{-\frac{1}{2}} \exp(-\frac{1}{2} \tilde{z}_t^T S_t^{-1} \tilde{z}_t)$$



2. Ellipsoidal gates

The measurements will be in the area $d^2 = \tilde{z}_t^T S_t^{-1} \tilde{z}_t \leq G$ with a probability defined by the gate threshold G .

- This area is called **validation gate**. The shape of the validation gate is a hyper-ellipsoid (an ellipse in 2d)
- G is taken from the inverse χ^2 cumulative distribution at a level α and M degrees of freedom
- Typical values for α are 0.95 or 0.99
- The validation gate is a **region of acceptance** such that $100(1 - \alpha)\%$ of true measurements are **rejected**.

Part 3: Multi-object tracking

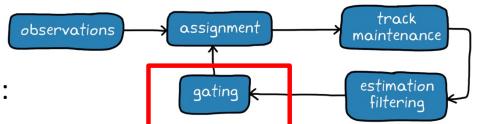
- Residual (or innovation) vector and its covariance matrix are defined as:

$$\tilde{z}_t = z_t - C_t \bar{\mu}_t$$

$$S_t = C_t \bar{\Sigma}_t C_t^T + Q_t$$

- If the measurement is of dimension M , the M -dimensional Gaussian probability density for the residual is:

$$p(\tilde{z}_t) = \det((2\pi)^M S_t)^{-\frac{1}{2}} \exp(-\frac{1}{2} \tilde{z}_t^T S_t^{-1} \tilde{z}_t)$$



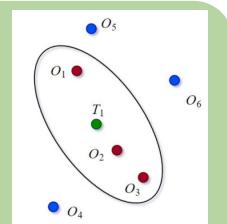
2. Ellipsoidal gates

The measurements will be in the area $d^2 = \tilde{z}_t^T S_t^{-1} \tilde{z}_t \leq G$ with a probability defined by the gate threshold G .

- If $d^2 \leq G$: detection is inside the gate of the track, and it will be considered for association.
- If $d^2 > G$: the possibility of the detection associated with the track is removed.

Example: T_1 is the predicted track estimate, while $O_1 - O_6$ are six detections.

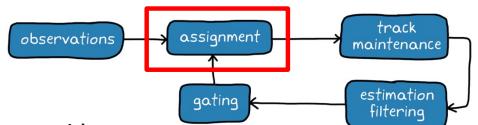
Based on the gating result, O_1 , O_2 and O_3 are within the validation gate.



Part 3: Multi-object tracking

One or more sensors generate multiple detections from multiple targets in a scan.

Assignment is the process of matching an observation to a tracked object (a track).



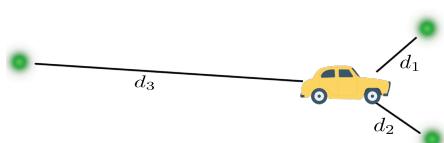
Assigning detections is very challenging:

- the number of targets or detections is large
- conflicts between different assignment hypotheses



Assignment problems, depending on the dimension, are categorized into:

- 2-D assignment problem – assigns n targets to m observations
- S-D assignment problem – assigns n targets to a set (m_1, m_2, m_3, \dots) of observations



2-D assignment approaches will be explained:

- ✓ GNN – adopts a global nearest data assignment approach
- ✓ JPDA - adopts a joint probability data association approach

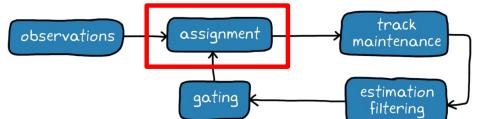
More about other assignment algorithms, e.g.,

<https://www.mathworks.com/help/fusion/multi-object-trackers.html>

Part 3: Multi-object tracking

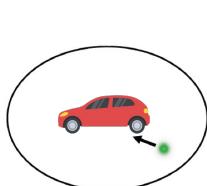
1. Global Nearest Neighbor (GNN) Method

- a single hypothesis assignment method
- the goal is to:
 - assign the global nearest observations to existing tracks and
 - create new track hypotheses for unassigned detections



GNN assignment problem:

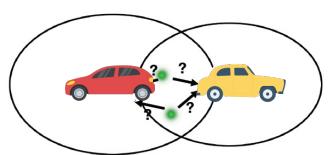
- easily solved if there are no conflicts of association between tracks.
- tracker assigns a track to its nearest neighbor



GNN works well with sparse objects.

Conflict situations:

- when there is more than one observation within a track's validation gate or
- an observation is in the gates of more than one track



GNN doesn't work well for clustered objects.

Part 3: Multi-object tracking

1. Global Nearest Neighbor (GNN) Method

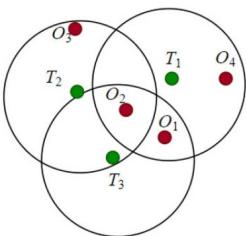
- To resolve the conflicts, the tracker must evaluate a cost matrix.

Define a *generalized statistical distance* d_{Gij}^2 between observation j to track i as:

$$d_{Gij}^2 = d_{ij}^2 + \ln[|S_{ij}|] \rightarrow \begin{array}{l} \text{logarithm of the determinant of the residual covariance matrix} \\ \text{(used to penalize tracks with greater prediction uncertainty)} \end{array}$$

Mahalanobis distance

Example:



Tracks	Observations			
	O_1	O_2	O_3	O_4
T_1	9	6	X	6
T_2	X	3	10	X
T_3	8	4	X	X

- Table shows a hypothetical cost matrix
- Optimal solutions are highlighted,
- Non-allowed assignments denoted by X.

Detection O_3 :

- unassigned,
- the tracker creates a new tentative track

11/28/22

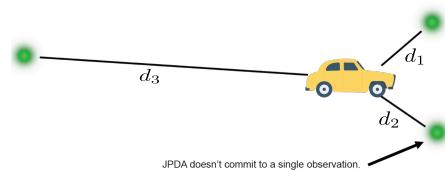
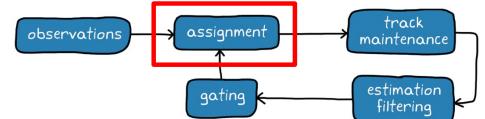
Lecture 17

17

Part 3: Multi-object tracking

2. Joint Probabilistic Data Association (JPDA) Method

- applies a soft assignment,
- detections within the validation gate of a track make a weighted contributions to the track based on their probability of association.

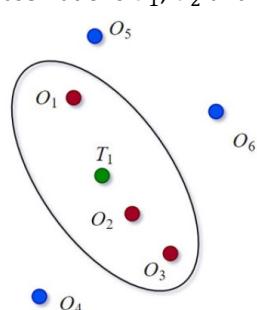


Example: JPDA tracker calculates the possibility of association between track T_1 and observations O_1, O_2 and O_3

Weighted sum of the residuals associated with track T_1 :

$$\tilde{z}_1 = \sum_{j=1}^3 p_{1j} \tilde{z}_{1j} \quad \tilde{z}_1 \text{ is used to update track } T_1 \text{ in the correction step of the tracking filter}$$

- p_{11}, p_{12}, p_{13} are association probabilities of the three observations
- $\tilde{z}_{11}, \tilde{z}_{12}, \tilde{z}_{13}$ are residuals relative to the track T_1



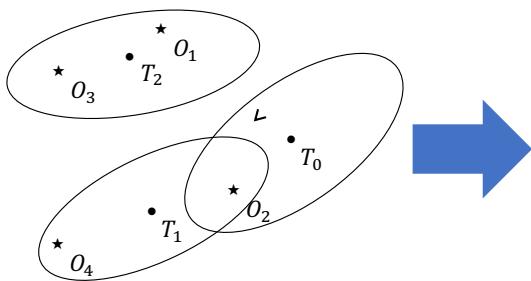
11/28/22

Lecture 17

18

Part 3: Multi-object tracking

JPDA example



Association matrix

		Track number			
		-1	0	1	2
Measurement number	1	1	0	0	1
	2	1	1	1	0
3	1	0	0	1	
4	1	0	1	0	

Probabilities of measurement-to-track associations

		Track number			
		-1	0	1	2
Measurement number	1	0	0	0.7	
	2	0.6	0.3	0	
3	0	0	0.2		
4	0	0.5	0		

Hypotheses 0...N

$p=0.1$

		Track number			
		-1	0	1	2
Measurement number	1	1	0	0	0
	2	1	0	0	0
3	1	0	0	0	
4	1	0	0	0	

$p=0.4$

		Track number			
		-1	0	1	2
Measurement number	1	0	0	0	1
	2	1	0	0	0
3	1	0	0	0	
4	1	0	0	0	

$p=0.3$

Example for track 1:

- 30 % that O_2 is correct
- 50 % that O_4 is correct
- 20 % that no measurement is correct

11/28/22

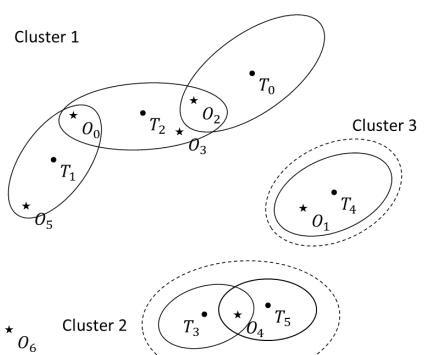
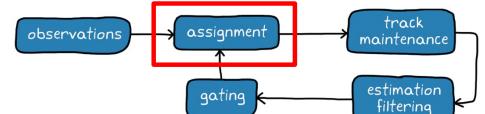
Lecture 17

19

Part 3: Multi-object tracking

JPDA takeaways

- Bayes-based data association
- Fuse measurements **weighted by the probability** of measurement-to-track association
- Probability depends on:
 - Innovation, predicted measurement, and innovation covariance
 - Modeled probability of track detection
 - Modeled probability of clutter
 - Nonparametric version is used where all numbers of clutter measurements are equally likely
- Too many hypotheses? Clustering
- MHT, Extended object tracking (probability hypotheses densities, phd)



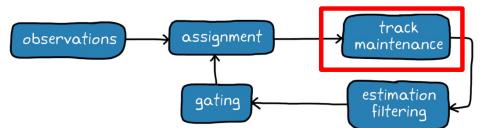
11/28/22

Lecture 17

20

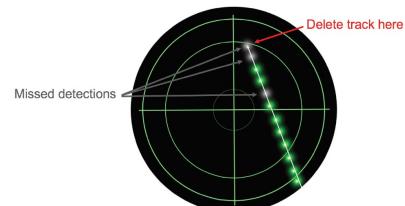
Part 3: Multi-object tracking

Track maintenance algorithms - used to delete and create tracks



1. Deleting a track

- only if it has not been assigned to a detection at least M times during the last N updates,
- M and N are tuning parameters.

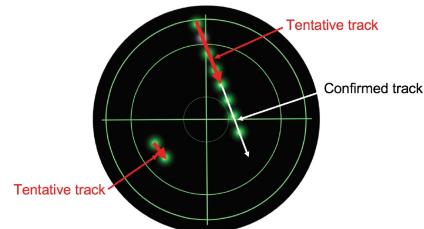


2. Creating a track

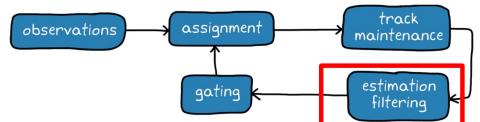
Is a single un-assigned observation a new object or not?

Create a tentative track— maintained but not treated as a real object

- confirmed - when detected M times in the last N updates,
- removed – the same logic as removing a confirmed track.

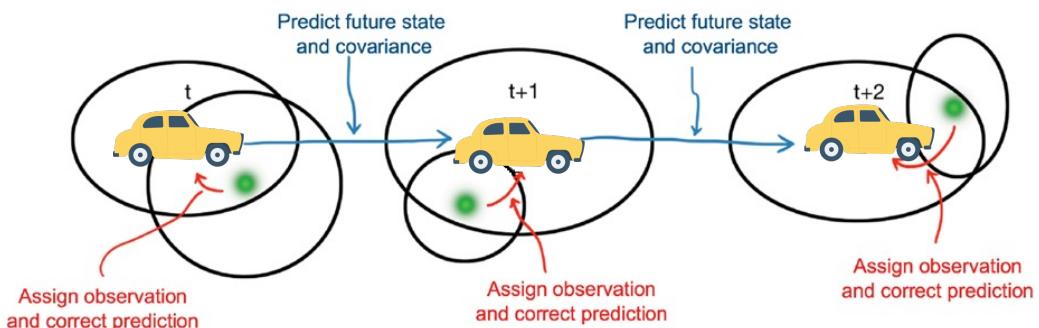


Part 3: Multi-object tracking



A set of estimation filters are running - one filter for each tracked object

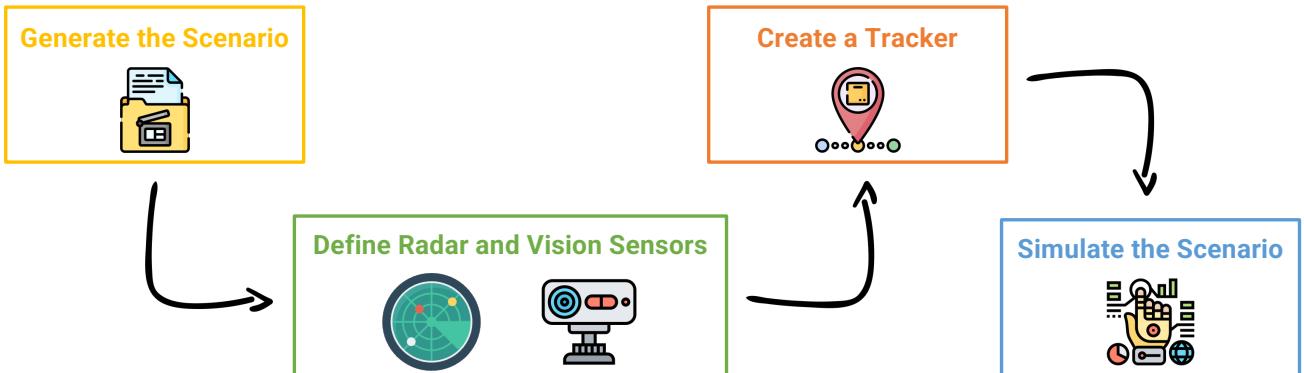
- identical to single-object tracking,
- different types of filters, e.g., interacting multiple model filter or the single model Kalman filter.



Part 4: Sensor fusion using radar and vision data

This example is part of the *Sensor Fusion and Tracking Toolbox* (*Matlab code but easily to transfer to e.g., Python*)

- shows how to generate a scenario, simulate sensor detections, and use sensor fusion to track simulated vehicles
- main benefit - ability to create rare and potentially dangerous events and test the vehicle algorithms with them



11/28/22

Lecture 17

23

<https://www.mathworks.com/help/driving/ug/sensor-fusion-using-synthetic-radar-and-vision-data.html>

Part 4: Sensor fusion using radar and vision data

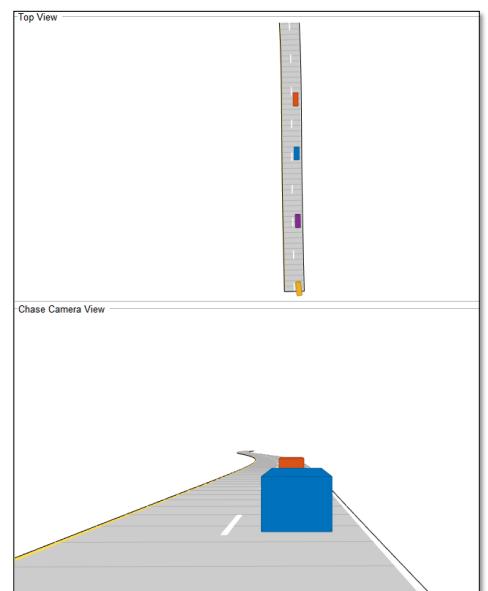


Scenario generation comprises of:

- generating a road network,
- defining vehicles that move on the roads.

Scenario in this example:

- highway road with two lanes is defined,
- **ego vehicle** and three cars around it:
 - **one** overtakes the ego vehicle and passes it on the left,
 - **one** drives right in front of the ego vehicle,
 - **one** drives right behind the ego vehicle.



11/28/22

Lecture 17

24

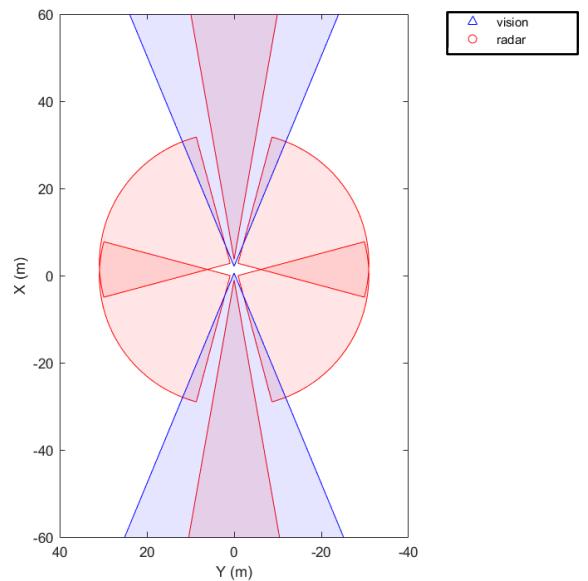
Part 4: Sensor fusion using radar and vision data

Define Radar and Vision Sensors



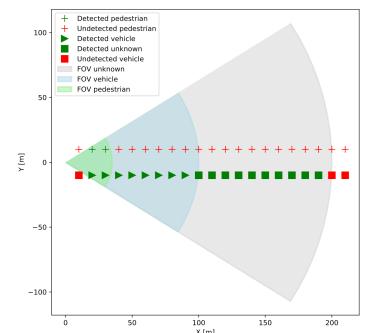
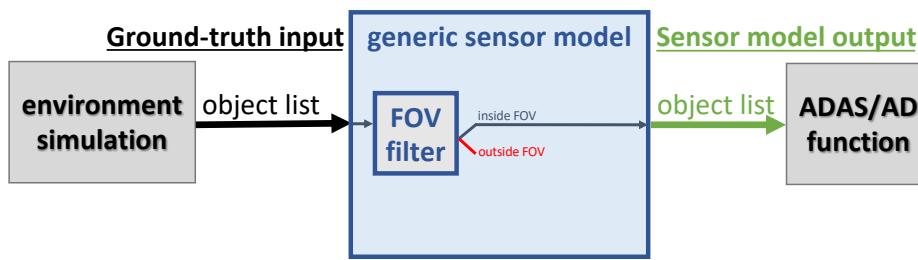
Sensors defined on the ego vehicle:

- 6 radar sensors:
 - 2 long-range radar sensors covering 20 degrees (in front and back),
 - 4 short-range radar sensors covering 90 degrees (two per side),
- 2 vision sensors:
 - Front-facing camera located at front windshield,
 - Rear-facing camera located at rear windshield,
- 360 degrees field of view is covered,
- sensors have some overlap and some coverage gap.



Part 4: Sensor fusion using radar and vision data

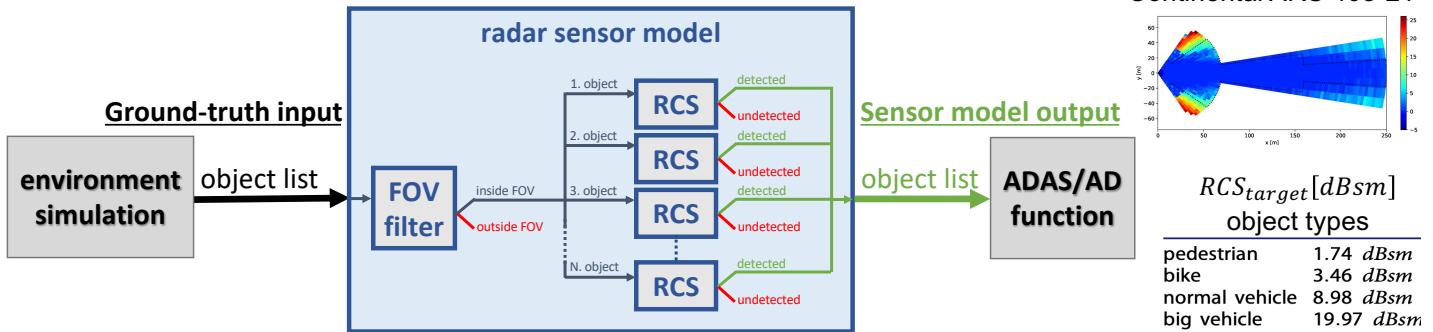
Basic functionality of a sensor model: FOV (field of view) filter



- **Boundary conditions:** maximum range r_{sensor} and opening angle φ_{sensor} of sensor's FOV
- **Input:** x, y -positions of objects in sensor's coordinate system
- **Task:** perform coordinate transformation and evaluate if $r_{target} < r_{sensor}$ and $\varphi_{target} < \varphi_{sensor}$
- **Output:** objects inside FOV

Part 4: Sensor fusion using radar and vision data

RCS (radar cross section) based radar model



- Boundary conditions:** $RCS_{min} [dBsm]$ detection thresholds of radar sensor
e.g. linear increase: $RCS_{min}(r = 0m) = 0.1dBsm; RCS_{min}(r = 250m) = 20dBsm$
- Input:** x, y -positions and RCS -values of objects
- Task:** perform coordinate transformation and evaluate if $RCS_{target} > RCS_{min}(r_{target})$
- Output:** objects detected by radar

Part 4: Sensor fusion using radar and vision data

Underlying equations

- Coordinate transformation

$$r_{target} = \sqrt{x_{target}^2 + y_{target}^2}$$

$$\varphi_{target} = \arctan\left(\frac{y_{target}}{x_{target}}\right)$$

- Detection threshold of radar at target location:

$$RCS_{min}(r_{target}) = RCS_{min}(r_0) + \frac{RCS_{min}(r_1) - RCS_{min}(r_0)}{r_1} * r_{target}$$

- Object detection:

$$RCS_{target} > RCS_{min}(r_{target}) \rightarrow \text{object detected}$$

Code implementation

```

object_list = [(50,2,10), (120,-10,5), ...]
for (x,y,RCS) in object_list:
    r = sqrt(x**2+y**2); phi = arctan(y/x)
    object_list_transformed.add((r,phi,RCS))

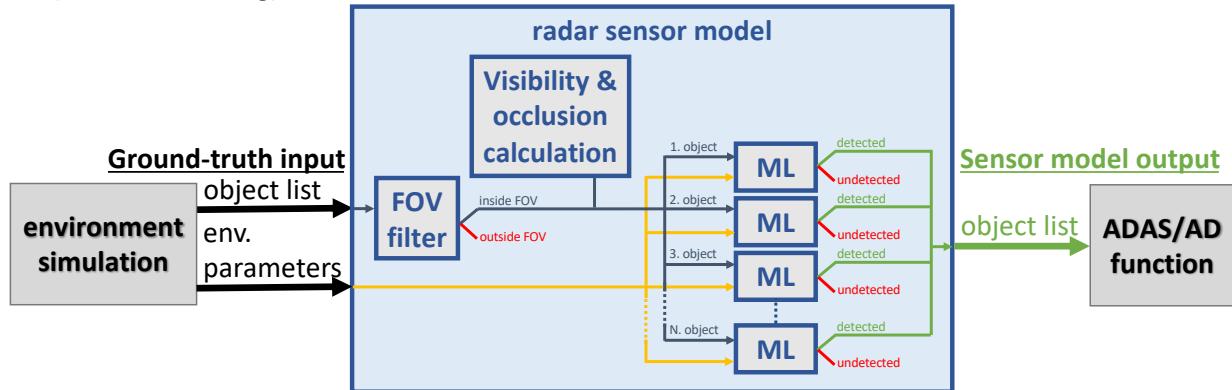
RCS_min_r0 = 0.1
r1 = 250; RCS_min_r1 = 20

for (r,phi,RCS) in object_list_transformed:
    RCS_min = RCS_min_r0 + r*(RCS_min_r1- RCS_min_r0)/r1
    if RCS_min < RCS:
        objects_detected.add((r,phi,RCS))

```

Part 4: Sensor fusion using radar and vision data

ML (machine learning) based detection



- **Pre-trained ML-algorithm** decides based on multiple inputs (x, y -positions, object type, occlusion, visibility, weather conditions, solar irradiance, RCS_{target} , R_{target} , etc.) whether object is detected or not
- **Visibility** can be included as e.g. visible (not occluded) part of object in percentage
- **Occlusion** can be included as e.g. number of objects that are in direct line-of-sight between sensor and target

Part 4: Sensor fusion using radar and vision data

Create a Tracker



- to track the vehicles that are close to the ego vehicle,
- Note:
 1. initialize a constant velocity motion model
 2. initialize the Kalman filter that works with position and velocity

It is responsible for the following:

- A. Assigning detections to tracks.
- B. Initializing new tracks based on unassigned detections. All tracks are initialized as 'Tentative', accounting for the possibility that they resulted from a false detection.
- C. Confirming tracks if they have more than M assigned detections in N frames.
- D. Updating existing tracks based on assigned detections.
- E. Coasting (predicting) existing unassigned tracks.
- F. Deleting tracks if they have remained unassigned (coasted) for too long

Part 4: Sensor fusion using radar and vision data

Simulate the Scenario

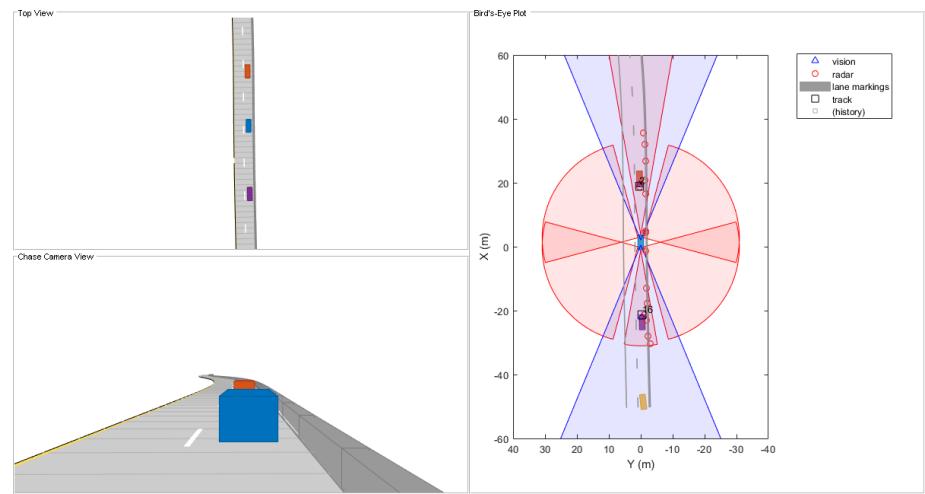


Simulation loop:

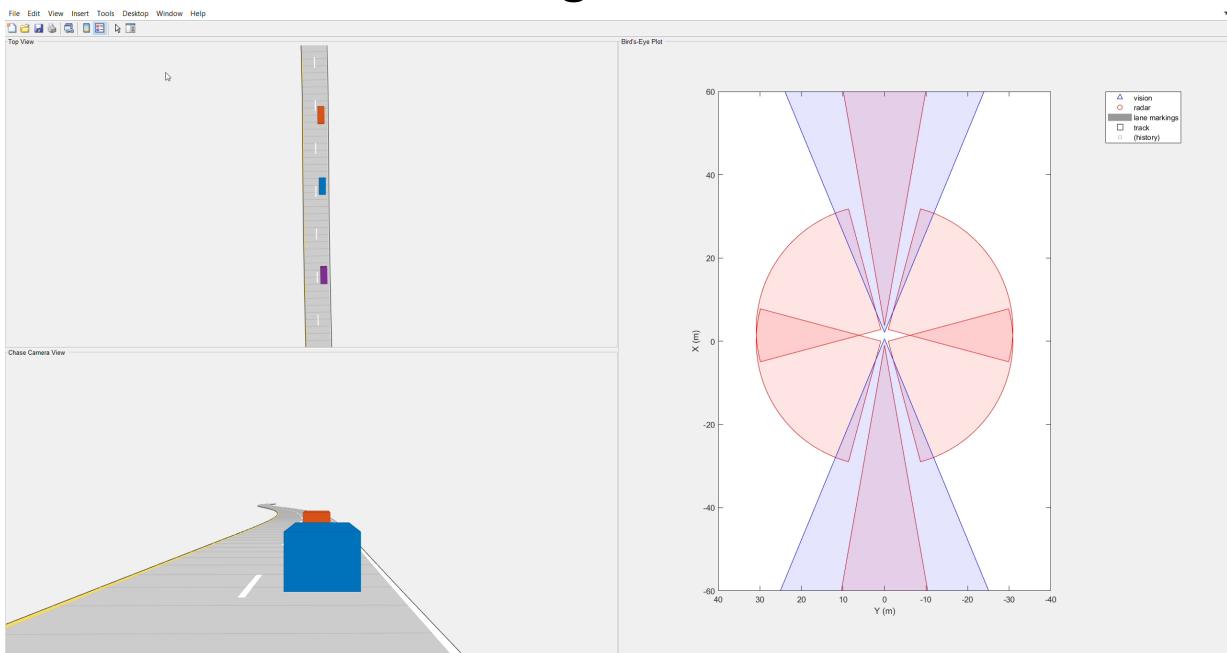
- moves the vehicles,
- calls the sensor simulation,
- performs the tracking.

Sampling times:

- scenario generation every 10 ms,
- sensors detect every 100 ms.

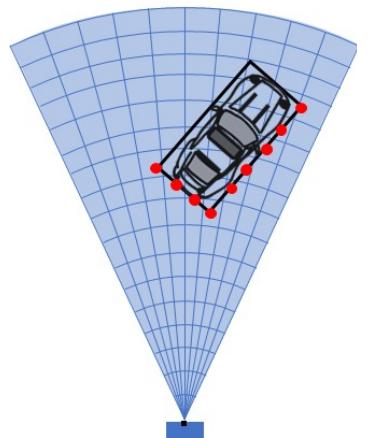


Part 4: Sensor fusion using radar and vision data



Part 4: Sensor fusion using radar and vision data

- How to track objects that return **multiple detections** in a single sensor scan?
- How to track objects with **high-resolution radar sensors**?
- Extended objects present new challenges to conventional trackers
- Standard trackers assume a single detection per object per sensor.
- **Extended object trackers**
 - Can handle multiple detections per object.
 - Estimate position and velocity, but also the dimensions and orientation of the object.
- **Prominent algorithms:**
 - Gamma Gaussian inverse Wishart probability hypothesis density (phd) tracker
 - Gaussian-mixture phd tracker
 - ...

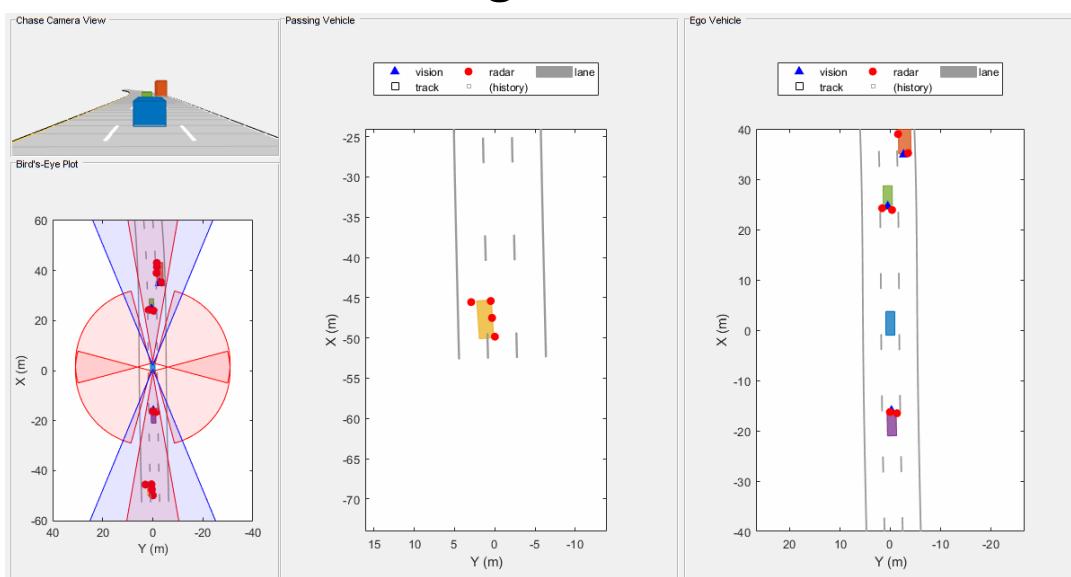


11/28/22

Lecture 17

33

Part 4: Sensor fusion using radar and vision data



- Gaussian mixture phd tracker (here: MATLAB implementation)
- Can handle multiple detections per object per sensor (here: 6 radars, 2 cameras)
- It estimates the size and orientation of the object (along with pose and velocity)

11/28/22

Lecture 17

34

Concluding remarks

- **Multi-object tracking** and **multi-sensor data fusion** - core of the autonomous systems perception
- Tracking is essential for guidance, navigation, and control of autonomous systems.
- **Typical tracking system**
 - estimates targets (number of targets and their states),
 - evaluates the situational environment in an area of interest by taking detections,
 - tracks the targets over time.

Takeaways

- **Single target tracking (STT)**
 - assumes only one target,
 - does not require data assignment or association,
 - the detection is directly fed to an estimator / filter.
- **Multiple target tracking (MTT), multi-object tracking (MOT)**
 - multiple detections from multiple targets,
 - use of one or more sensors,
 - one or more tracks are used to estimate the states of the targets.
- **Extended object tracking**
 - high-resolution radar/lidar sensors,
 - can handle multiple detections per object.
 - estimate position and velocity, but also the dimensions and orientation of the object.

AA 274

Principles of Robotic Autonomy

Stereo vision and structure from motion



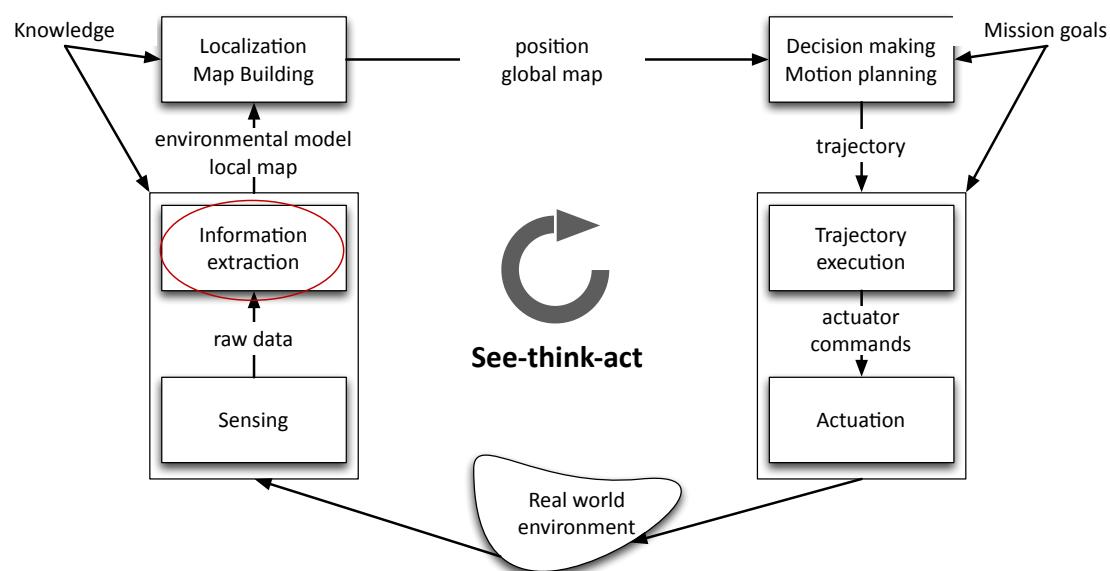
Logistics

- It's the final (project) stretch!
 - All sections are open office hours for project discussion with TAs
 - Final project check-in was due Tuesday
 - Final project demos: Thursday, December 15th, 3:30 – 6:30pm
 - Sign up for the correct slots with correct group number

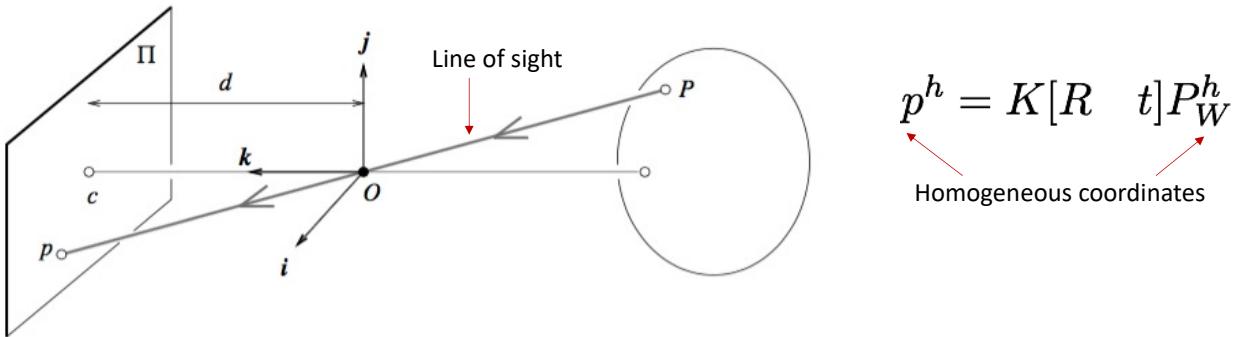
Today's lecture

- Aim
 - Learn fundamental geometric concepts needed for 3D reconstruction
 - Learn basic techniques to recover scene structure, chiefly stereo and structure from motion
- Readings
 - SNS: 4.2.5 – 4.2.7
 - D. A. Forsyth and J. Ponce [FP]. Computer Vision: A Modern Approach (2nd Edition). Prentice Hall, 2011. Sections 7.1 and 7.2.

The see-think-act cycle



Measuring depth



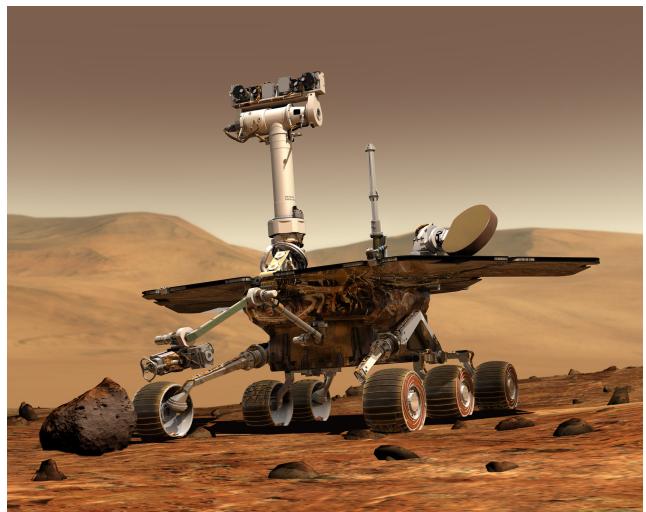
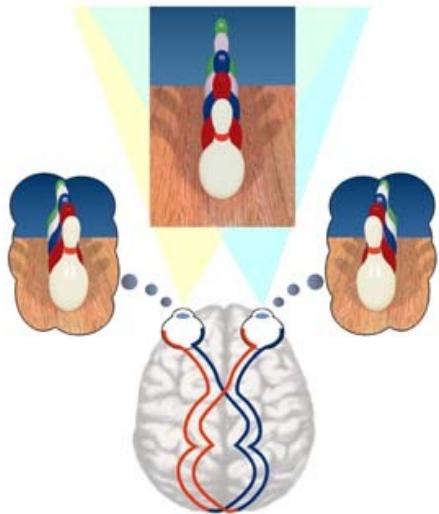
Once the camera is calibrated, can we measure the location of a point P in 3D given its known observation p ?

- No: one can only say that P is located *somewhere* along the line joining p and O !

Recovering structure

- **Structure:** 3D scene to be reconstructed by having access to 2D images
- Common methods
 1. Through recognition of landmarks (e.g., orthogonal walls)
 2. Depth from focus: determines distance to one point by taking multiple images with better and better focus
 3. Stereo vision: processes two distinct images taken at the *same time* and assumes that the relative pose between the two cameras is *known*
 4. Structure from motion: processes two images taken with the same or different cameras at *different times* and from different *unknown* positions

Stereopsis (why we have two eyes)

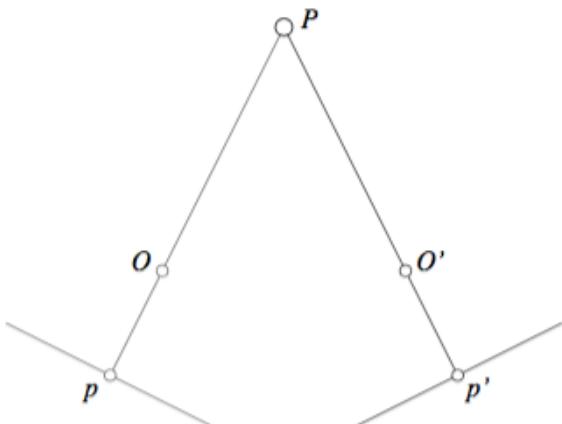


12/08/22

AA 274 | Lecture 18

7

Binocular reconstruction



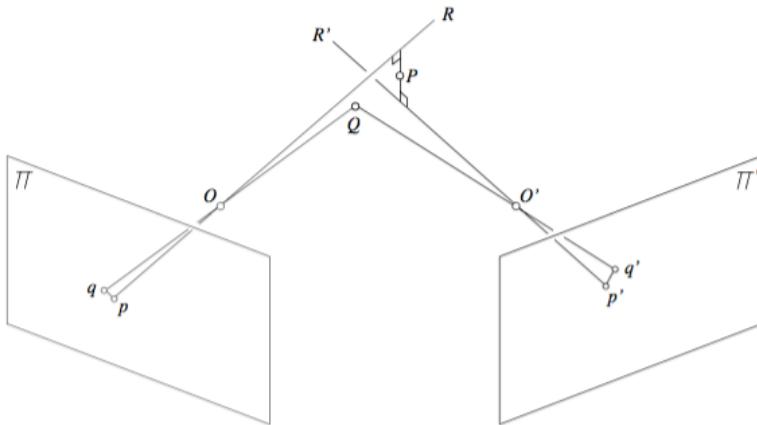
- **Given:** calibrated stereo rig and two image matching points p and p'
- **Find** corresponding scene point by intersecting the two rays \overline{Op} and $\overline{O'p'}$ (process known as **triangulation**)

12/08/22

AA 274 | Lecture 18

8

Approximate triangulation



- Due to noise, triangulation problem is often solved as finding the point Q with images q and q' that minimizes

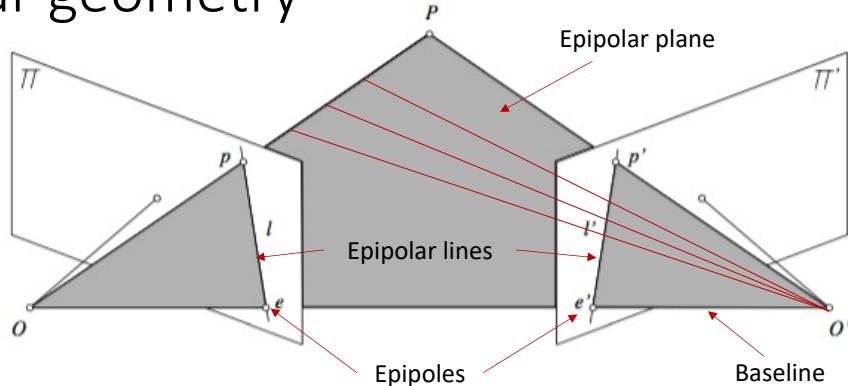
$$d^2(p, q) + d^2(p', q')$$

Re-projection error

Stereo vision process

- Stereo vision consists of two steps:
 1. *fusion* of features observed by two (or more) cameras -> **correspondence**
 2. *reconstruction* of their three-dimensional preimages -> **triangulation**
- Step 2 is relatively easy; Step 1 requires you to establish correct correspondences and avoid erroneous depth measurements
- Several constraints can be leveraged to simplify Step 1 (e.g., similarity constraint, continuity constraints, etc.); most important: **epipolar constraint**

Epipolar geometry



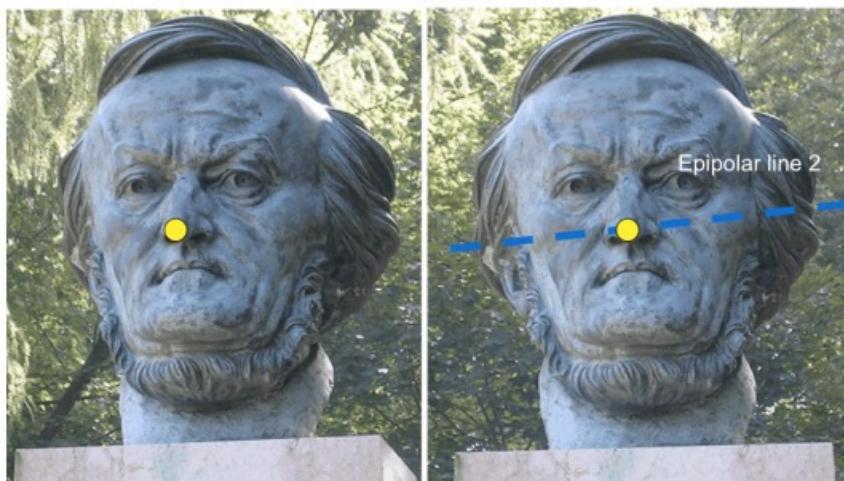
- Consider images p and p' of a point P observed by two cameras from O, O'
- These five points all belong to the *epipolar plane* defined by p, O, O' , or equivalently, p', O, O'
- **Epipolar constraint:** potential matches for p must lie on epipolar line l' (and vice-versa)

12/08/22

AA 274 | Lecture 18

11

Epipolar constraint



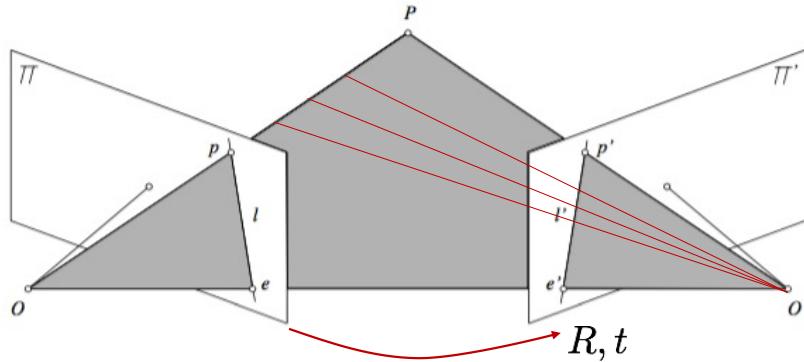
- Search for matches can be restricted to the epipolar line instead of the whole image! → one dimensional search

12/08/22

AA 274 | Lecture 18

12

Epipolar constraint: derivation



- Epipolar constraint: \overline{Op} , $\overline{O'p'}$, and $\overline{OO'}$ must be coplanar, or

$$\overline{Op} \cdot [\overline{OO'} \times \overline{O'p'}] = 0$$

Aside: matrix notation for cross product

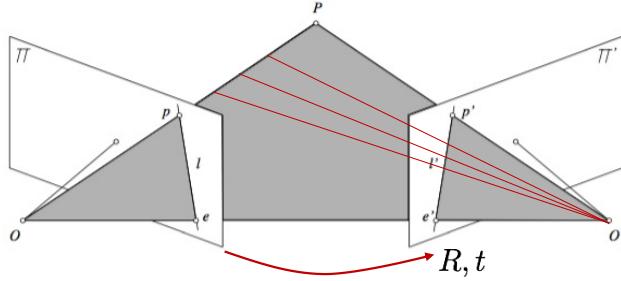
- Cross product can be expressed as the product of a **skew-symmetric** matrix and a vector

$$a \times b = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [a]_{\times} b$$

$\underbrace{\phantom{[a]_{\times}}}_{:= [a]_{\times}}$

Epipolar constraint: derivation.

More details: CS231a



- Assume that the world reference system is co-located with camera 1
- After some algebra, epipolar constraint becomes [FP, Section 7.1]

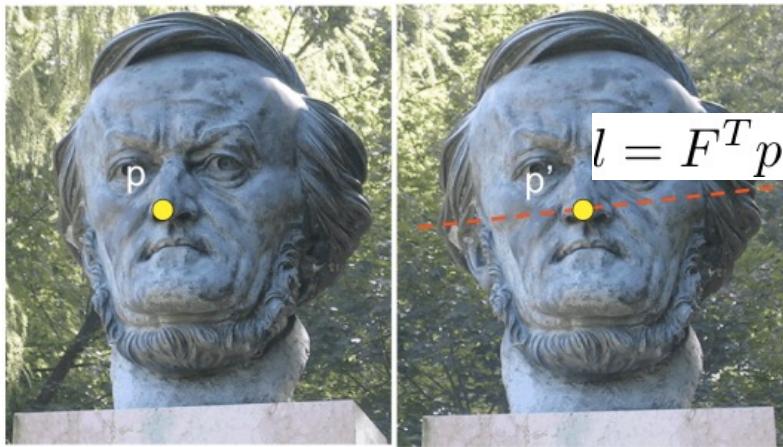
$$p^T F p' = 0$$

where: $F = K^{-T} [t]_{\times} R K'^{-1}$

Key facts

- F is referred to as the **fundamental matrix**
- $l = Fp'$ (resp. $l' = F^T p$) represents the epipolar line corresponding to the point p' (resp. p) in the first (resp. second) image. This exploits the homogenous notation for lines.
- $F^T e = F e' = 0 \rightarrow F$ is also singular (as t is parallel to the coordinate vectors of the epipoles)
- F has 7 DoF (9 elements – common scaling – $\det(F)=0$)

Usefulness of fundamental matrix



- Assume F is given
- Given a point in image 1, one can compute the corresponding epipolar line in image 2 **without any additional information needed!**

12/08/22

AA 274 | Lecture 18

17

Estimating the fundamental matrix

- 8-point algorithm

$$p = [u, v, 1]^T, \quad p' = [u', v', 1]^T \quad \Rightarrow \quad [u, v, 1] \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = 0$$

$$\Rightarrow [uu', uv', u, vu', vv', v, u', v', 1] \begin{bmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{bmatrix} = 0 \quad \Rightarrow \quad Wf = 0$$

f \nwarrow \uparrow
 $n \times 9$ matrix of known coefficients

- Given $n \geq 8$ correspondences, one then solves

$$\min_{f \in R^9} \|Wf\|^2 \Rightarrow \tilde{F}$$

subject to $\|f\|^2 = 1$

12/08/22

AA 274 | Lecture 18

18

Enforcing the rank constraint

- \tilde{F} satisfies the epipolar constraints, but is not necessarily singular (hence, is not necessarily a proper fundamental matrix)
- Enforce rank constraint (again, via SVD decomposition)

Find F that minimizes $\|F - \tilde{F}\|^2$ ← Frobenius norm
subject to $\det(F) = 0$

- 8-point algorithm

1. Use linear least squares to compute \tilde{F}
2. Enforce rank-2 constraint via SVD

Parallel image planes

- Assume image planes are parallel
- Epipolar lines are horizontal
- v coordinates are equal
 - Easier triangulation
 - Easier correspondence problem
- Is it possible to warp images to simulate a parallel image plane?

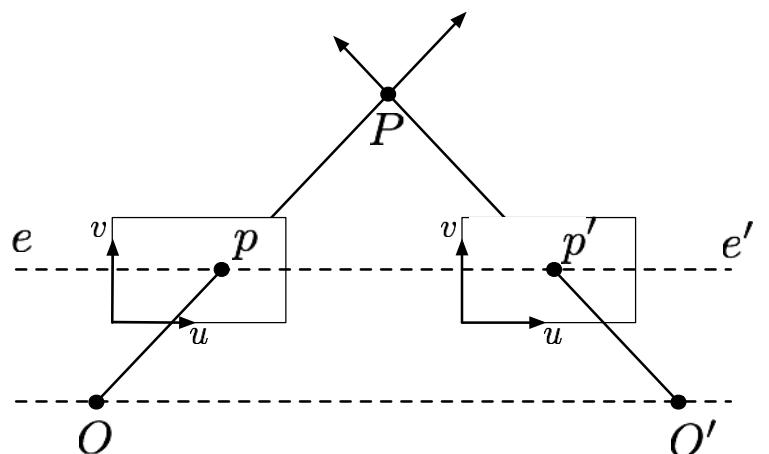
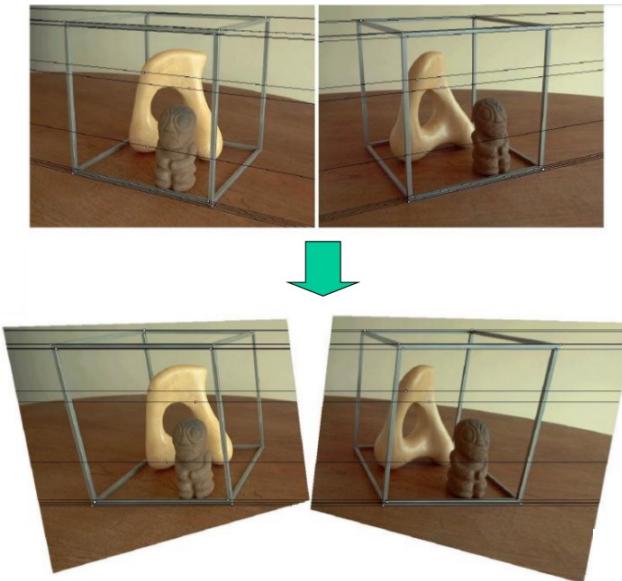


Image rectification



12/08/22

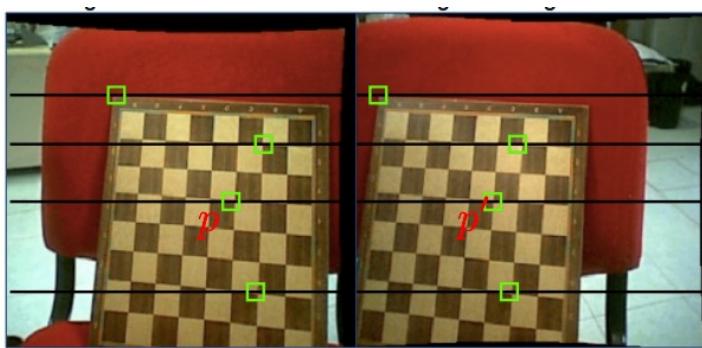
AA 274 | Lecture 18

21

- Achieved by applying an appropriate projective transformation
- Several algorithms exist
- From now on, we assume rectified image pairs

Back to stereo vision process

- Recall that stereo vision consists of two steps:
 1. *fusion* of features observed by two (or more) cameras (**correspondence**)
 2. *reconstruction* of their three-dimensional preimages (**triangulation**)
- **Correspondence problem**



12/08/22

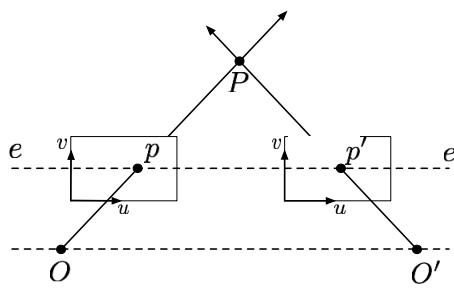
AA 274 | Lecture 18

Goal: find corresponding observations p and p'
Exploits epipolar constraints
Two classes of algos: *area-based* and *feature-based*
Hard problem: occlusions, repetitive patterns, etc.; more on this later

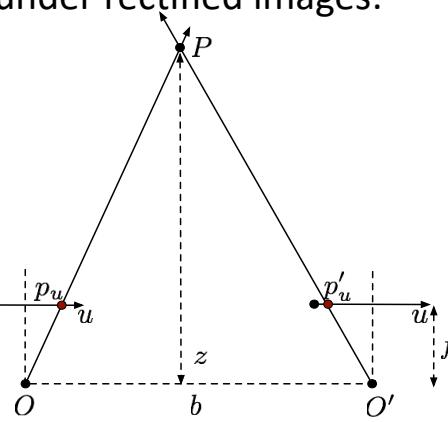
22

Triangulation under rectified images

- We already saw how to triangulate correspondences in the general case
- **Triangulation problem** under rectified images:



12/08/22



AA 274 | Lecture 18

From similar triangles:

$$z = \frac{bf}{p_u - p'_u}$$

disparity

Large baseline: Object might be visible from one camera, but not the other
Small baseline: large depth error

23

Disparity map

- Disparity: pixel displacement between corresponding points
- Disparity map: holds the disparity values for every pixel
- Nearby objects experience largest disparity

Stereo pair



12/08/22

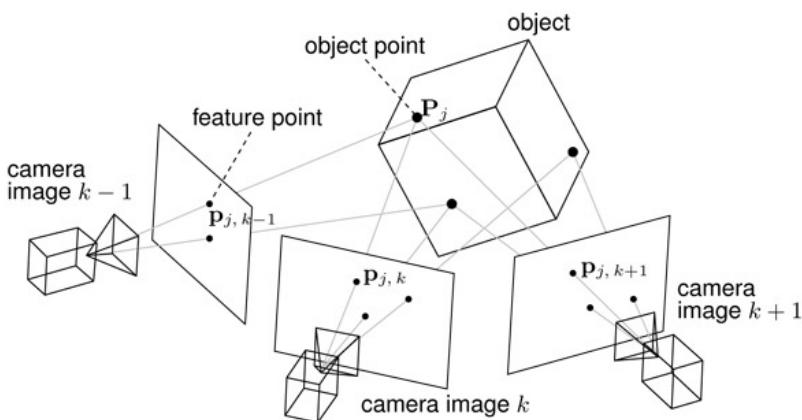
Disparity map



AA 274 | Lecture 18

24

Method #3: structure from motion (SFM)



Given m images of n fixed 3D points

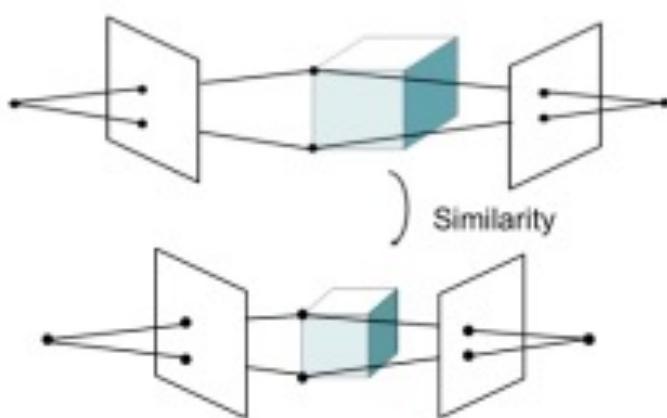
$$p_{j,k}^h = M_k P_j^h$$

Find:

- m projection matrices M_k (**motion**)
- n 3D points P_j (**structure**)

SFM ambiguity

- It is not possible to recover the absolute scale of the observed scene

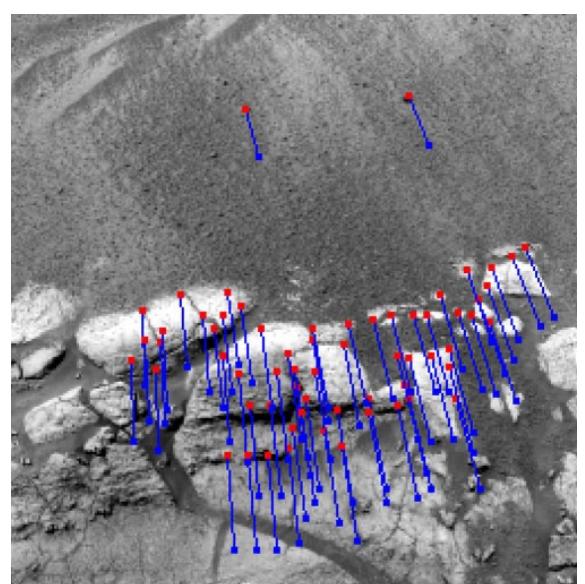


Solution to SFM problem (high-level)

- Several approaches available:
 - Algebraic approach (by fundamental matrix)
 - Bundle adjustment
- Algebraic approach (2-views)
 1. Compute fundamental matrix F (e.g., via 8-point algorithm)
 2. Use F to estimate projection camera matrices
 3. Use projection camera matrices for triangulation

Application of SFM: visual odometry

- **Visual odometry**: estimate the motion of the robot by using visual input (and possibly additional information)
 - Single camera: absolute scale must be estimated in other ways
 - Stereo camera: measurements are directly provided in absolute scale



Principles of Robot Autonomy I

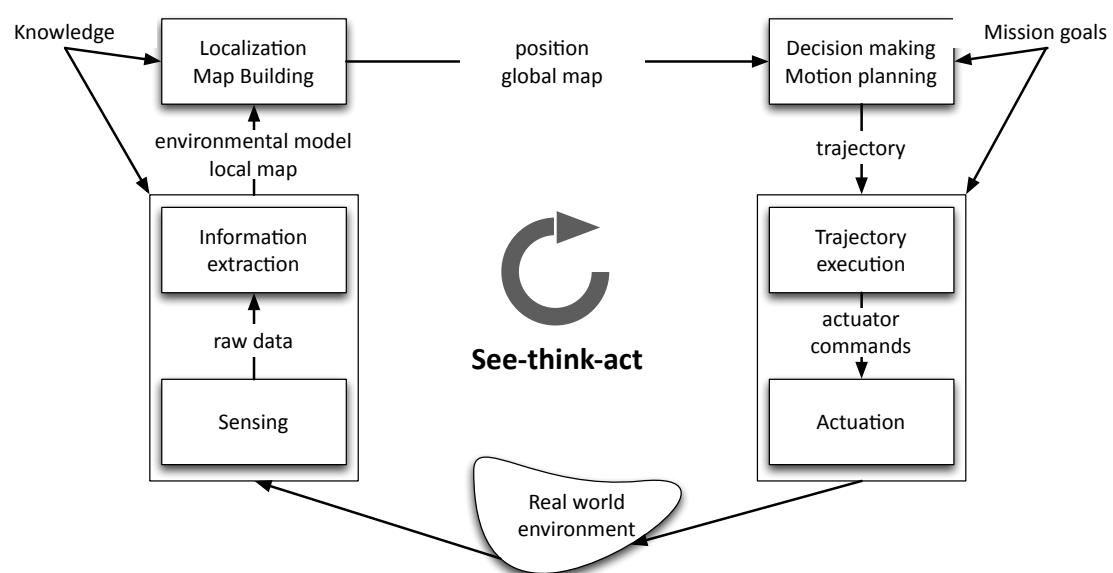
Finite state machines



Stanford
University



The see-think-act cycle



Today's lecture

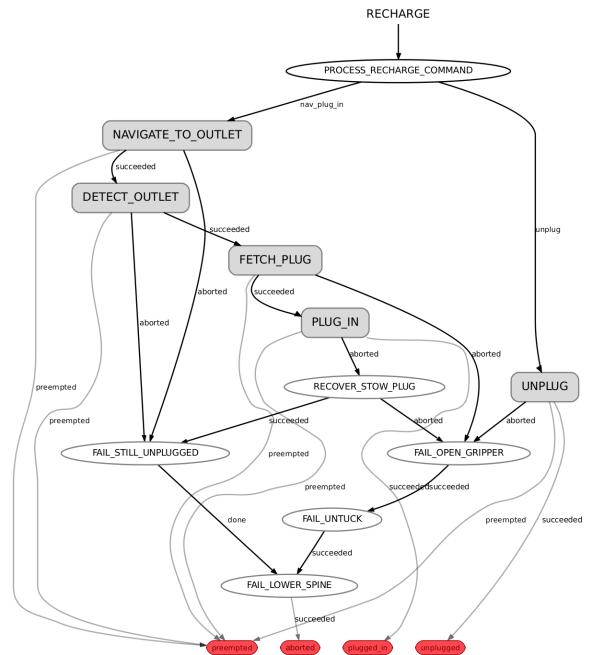
- Aim
 - Introduce and formalize the concept of Finite State Machines (FSMs)
 - Discuss their relevance, strengths and limitations
 - Introduce tools to allow you to use them effectively
- Readings
 - Chapter 4 of Leslie Kaelbling, Jacob White, Harold Abelson, Dennis Freeman, Tomás Lozano-Pérez, and Isaac Chuang. *6.01SC Introduction to Electrical Engineering and Computer Science I*. Spring 2011. Massachusetts Institute of Technology: MIT OpenCourseWare.

12/08/22

AA 274 | Lecture 19

3

Motivation



12/08/22

AA 274 | Lecture 19

4

Finite State Machines

Definition: A computational model for systems whose output depends on the **entire history** of their inputs.

A finite state machine is a modeling framework, NOT an algorithm (similar to Markov decision processes, probability densities, factor graphs etc.)

Finite State Machines in practice

- In practice, used in many different ways
 - Synthetically (specifies a program)
 - E.g. a product manager and an engineer specifies how an ATM machine should “behave” before starting its implementation
 - Analytically (describe the behavior of a combination of systems)
 - E.g. two self-driving cars could be modeled as FSMs. An engineer could try to see if they might end up stuck in some infinite loop at an intersection
 - Predictively (to predict interaction with an environment)
 - A self-driving car could have an internal model of a pedestrian as an FSM and use it to figure out how it should behave around it

Why are we teaching FSMs?

- For the practitioner: designing the extremely complex state machines required to fly drones, drive self-driving cars or operate warehouse robots is still one of the most time-consuming/difficult tasks faced by companies...
- How do we handle the failure of a combination of sensors gracefully?
- How do we negotiate an intersection?
- How do I get my turtlebot to start backtracking after a collision?

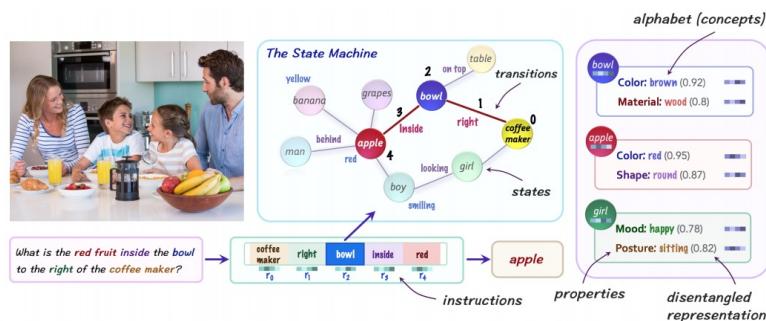
12/08/22

AA 274 | Lecture 19

7

Why are we teaching FSMs?

- For the researcher: It's a fundamental building block of how we understand computation, and still relevant to research today...



Hudson, Drew A., and Christopher D. Manning. "Learning by abstraction: The neural state machine." *arXiv preprint arXiv:1907.03950* (2019).

12/08/22

AA 274 | Lecture 19

8

Mathematical definition

- Sets:
 - A set of states S
 - A set of inputs I , called the input vocabulary
 - A set of outputs O , called the output vocabulary
- Maps:
 - Next-state function that maps input and the state to the next state $n(i_t, s_t) \rightarrow s_{t+1}$
 - Output function $o(i_t, s_t) \rightarrow o_t$
- An initial state s_0

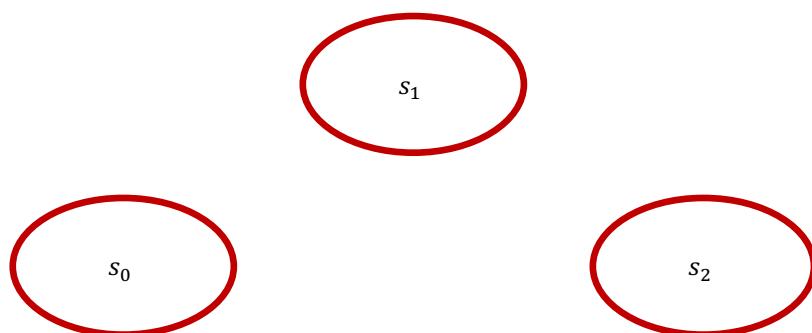
Graphical representation

- Given the sets (S, I, O) , it is common to express the maps (n, o) by using a graph

$S: \{s_0, s_1, s_2\}$

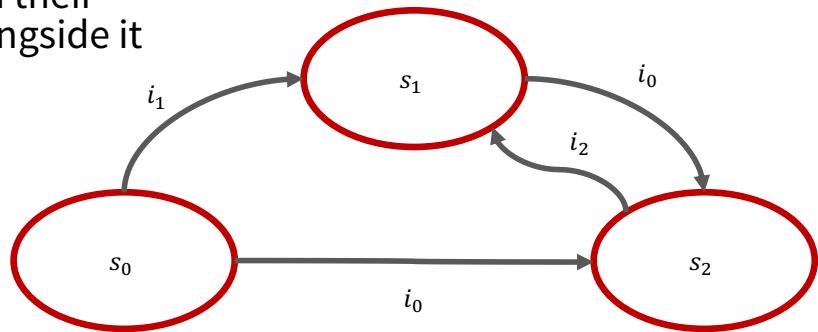
$I: \{i_0, i_1, i_2\}$

$O: \{o_0, o_1\}$



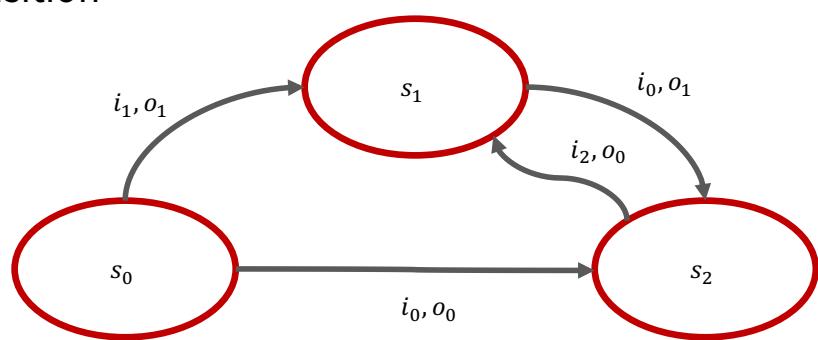
Graphical representation

The transition (next-state) map is represented by arrows between states, with their associated input alongside it



Graphical representation

The output map is written alongside each transition



Example: parking gate control

The gate can be in one of three positions: ‘top’, ‘middle’ or ‘bottom’

A sensor tells the gate if a car is waiting in front of it

A sensor tells the gate if a car has just passed through it

The gate can take the following actions: raise the gate, lower the gate, no operation (nop).

We want the following behavior:

- If a car wants to come through, need to raise the arm to ‘top’ position
- The gate has to stay there until the car has driven though the gate
- The gate has to go back down after the car has gone through



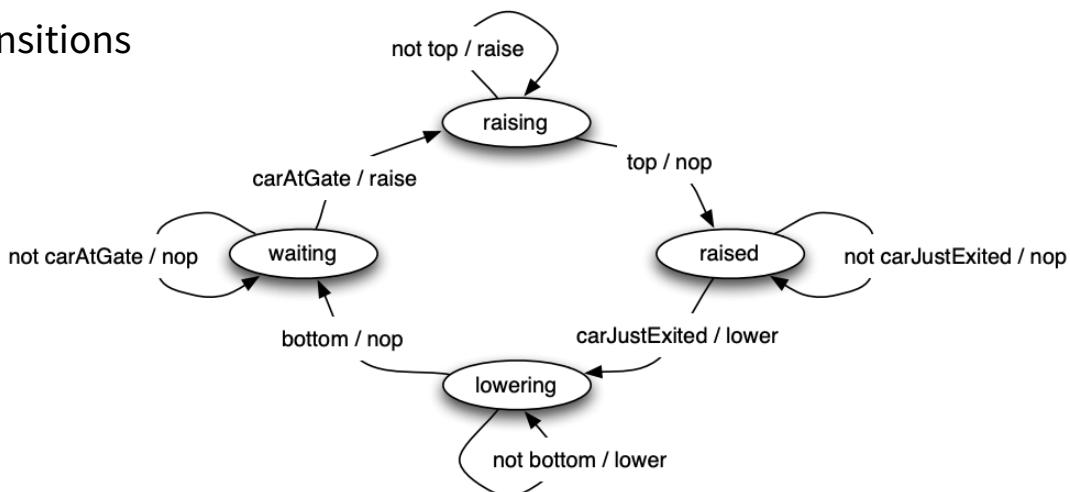
Example: parking gate control

- States: ‘waiting’, ‘raising’, ‘raised’, ‘lowering’
- Input: ‘no car at gate’, ‘car at gate’, ‘gate at top’, ‘not gate at top’, ‘gate at bottom’, ‘not gate at bottom’, ‘car just exited’, ‘not car just existed’
- Output: ‘raise’, ‘lower’, ‘nop’



Example: parking gate control

- Transitions

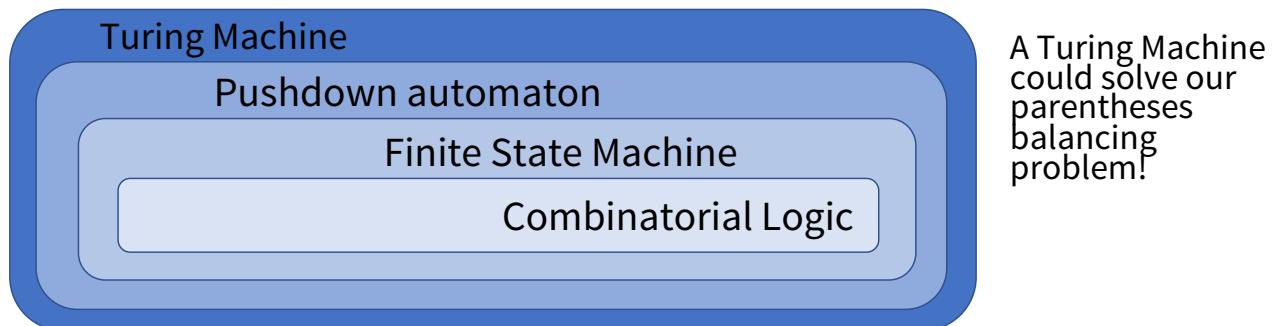


Example: parentheses balancing

- We want to design an automata that can read a string of text of any length and say whether or not the parentheses in the string are balanced or not
 - Balanced: "1 + (2 + 3 - (4 * 5))"
 - Not balanced: "1 + (2 + 3 - 4 * 5))"
- "... a string of text of **any length**..."
- A robot that can accomplish such a task would need an infinite number of states... and cannot therefore be represented by a **finite** state machine

FSM in the bigger picture of computation

- In terms of computational power, (deterministic) finite state machines are actually somewhat low on the totem pole of automata... with Turing Machines somewhere close to the top.



Architecture

- The architecture of finite state machines can become quite complex
- Additional states can generate an exponential number of transitions
- Strategies to keep the architecture tractable:
 1. Reduction of redundant states
 2. Hierarchical finite state machines
 3. Composition using common patterns

Finite State Machine optimization

- Algorithms exist to identify and combine states that have equivalent behavior
- Equivalent states:
 - Same output
 - For all input combinations, state transition to same or equivalent states
- Sketch of polynomial time algorithm:
 - Place all states in one set
 - Initially partition set based on output behavior
 - Successively partition resulting subsets based on next state transitions
 - Repeat until no further partitioning

12/08/22

AA 274 | Lecture 19

19

Finite State Machine optimization

Input Sequence	Present State	Next State X=0	X=1	Output X=0	X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0



Sequence detector for 010 or 110

(S0 S1 S2 S3 S4 S5 S6)

(S0 S1 S2 S3 S5) (S4 S6)

(S0 S3 S5) (S1 S2) (S4 S6)

(S0) (S3 S5) (S1 S2) (S4 S6)

Input Sequence	Present State	Next State X=0	X=1	Output X=0	X=1
Reset	S0	S1'	S1'	0	0
0 + 1	S1'	S3'	S4'	0	0
X0	S3'	S0	S0	0	0
X1	S4'	S0	S0	1	0

12/08/22

AA 274 | Lecture 19

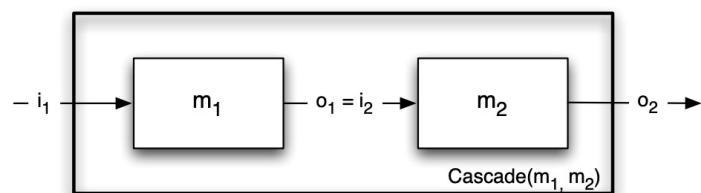
20

Hierarchical Finite State Machines

- Some states might not be equivalent, but it might still be beneficial to group closely related ones together
- This leads to the following two concepts:
 - Super-states (groups of states)
 - Generalized transitions (transitions between super-states)

Composition

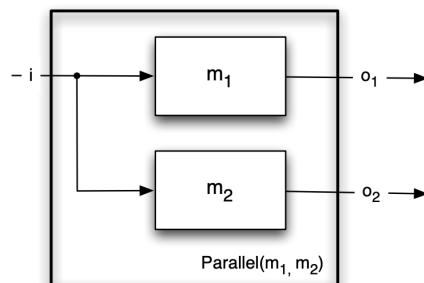
- Cascade
 - Requirement: output vocabulary of m_1 must match input vocabulary of m_2
 - Resulting state: concatenation of states
 - Resulting input: input of m_1
 - Resulting output: output of m_2



Composition

- Parallel

- Requirement: Input vocabularies must be the same
- Resulting state: concatenation of states
- Resulting input: same as input vocabulary of component machines
- Resulting output: concatenation of outputs



12/08/22

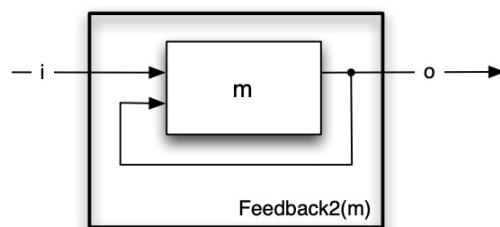
AA 274 | Lecture 19

23

Composition

- Feedback

- Requirement: Input and output vocabularies must be the same
- Resulting state: same
- Resulting input: partial input
- Resulting output: same



12/08/22

AA 274 | Lecture 19

24

Implementation

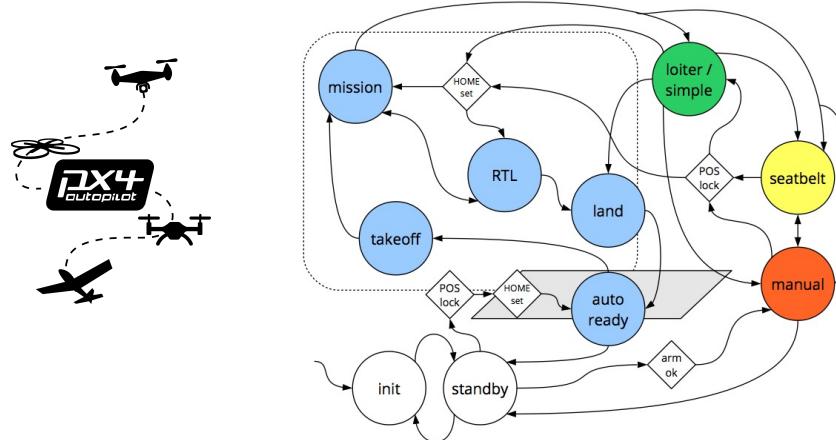
- Aim of this section
 - Understand that you do NOT have to use anything in particular in order to implement a FSM
 - Understand that there are however common ways to implement finite state machines
 - Grow awareness of tools available to help you build and analyze them

Implementation

- A common strategy is to exploit Object Oriented Programming (OOP) and implement a class that corresponds to your finite state machine
- The class keeps track of which state the FSM is in (e.g. in a variable)
- A loop repeats at some fixed rate
- Each loop, the FSM input is read (e.g. sensors, clock)
- The current state is executed (as an if/else block)
 - Actions that need to be taken (e.g. set actuator setpoints)
 - Transition to next state (e.g. state variable updated)

Example implementation

- PX4: in many ways the leading open source flight software for drones



12/08/22

AA 274 | Lecture 19

27

Example implementation

- Commander.cpp

```
1463     while (!should_exit()) {  
2355         bool nav_state_changed = set_nav_state(&status,
```

- state_machine_helper.cpp

```
441         switch (internal_state->main_state) {  
442             case commander_state_s::MAIN_STATE_ACRO:  
443                 status->nav_state = vehicle_status_s::NAVIGATION_STATE_ACRO;  
444                 break;
```

12/08/22

AA 274 | Lecture 19

28

Example implementation

- Your very own navigator.py!

```
# STATE MACHINE LOGIC
# some transitions handled by callbacks
if self.mode == Mode.IDLE:
    pass
elif self.mode == Mode.ALIGN:
    if self.aligned():
        self.current_plan_start_time = rospy.get_rostime()
        self.switch_mode(Mode.TRACK)
elif self.mode == Mode.TRACK:
    if self.near_goal():
        self.switch_mode(Mode.PARK)
    elif not self.close_to_plan_start():
        rospy.loginfo("replanning because far from start")
        self.replan()
    elif (rospy.get_rostime() - self.current_plan_start_time).to_sec() > self.current_plan_duration:
        rospy.loginfo("replanning because out of time")
        self.replan() # we aren't near the goal but we thought we should have been, so replan
elif self.mode == Mode.PARK:
    if self.at_goal():
        # forget about goal:
        self.x_g = None
        self.y_g = None
        self.theta_g = None
        self.switch_mode(Mode.IDLE)

self.publish_control()
rate.sleep()
```

ROS State Machines: SMACH

- A ROS tool that allows you to synthesize FSMs more easily
- Provides visualization tools
- Support hierarchical state machines
- Enables easy composition
- See <http://wiki.ros.org/smach/Tutorials/Getting%20Started>

SMACH: Basic Syntax

- Two main components:
 - SMACH State
 - SMACH Container (e.g. FSM)

SMACH: Basic Syntax

- SMACH State
 - The basic state abstraction. Corresponds 1:1 with the FSM states described earlier
 - Inherit from `smach.State` and must implement two functions:
 - `__init__`
 - `execute`
 - `execute` should return ‘outcomes’

SMACH: Basic Syntax

```
class Foo(smach.State):
    def __init__(self, outcomes=['outcome1', 'outcome2']):
        # Your state initialization goes here

    def execute(self, userdata):
        # Your state execution goes here
        if xxxx:
            return 'outcome1'
        else:
            return 'outcome2'
```

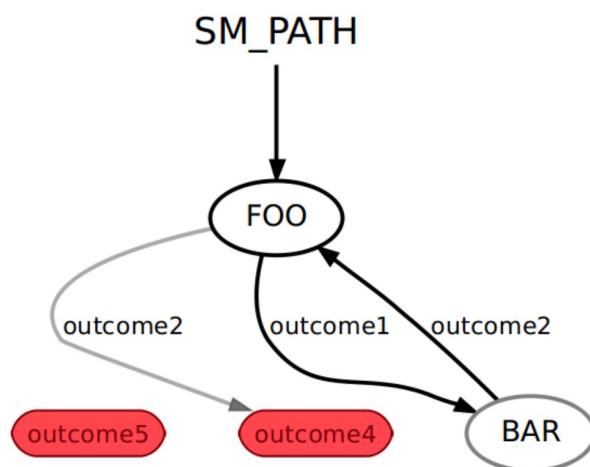
SMACH: Basic Syntax

- SMACH Container
 - Roughly corresponds to the idea of a finite state machine, with variations.
 - You are most likely to use the container `smach.StateMachine`
 - States can be added to containers
 - Containers can be composed

SMACH: Basic Syntax

```
sm = smach.StateMachine(outcomes=[ 'outcome4' , 'outcome5' ])
with sm:
    smach.StateMachine.add('FOO', Foo(),
                           transitions={ 'outcome1' : 'BAR',
                                         'outcome2' : 'outcome4' })
    smach.StateMachine.add('BAR', Bar(),
                           transitions={ 'outcome2' : 'FOO' })
```

SMACH: Basic Example



SMACH: Basic Example

```
# define state Foo
class Foo(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['outcome1', 'outcome2'])
        self.counter = 0

    def execute(self, userdata):
        rospy.loginfo('Executing state FOO')
        if self.counter < 3:
            self.counter += 1
            return 'outcome1'
        else:
            return 'outcome2'
```

12/08/22

AA 274 | Lecture 19

37

SMACH: Basic Example

```
# define state Bar
class Bar(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['outcome2'])

    def execute(self, userdata):
        rospy.loginfo('Executing state BAR')
        return 'outcome2'
```

12/08/22

AA 274 | Lecture 19

38

SMACH: Basic Example

```
# main
def main():
    rospy.init_node('smach_example_state_machine')

    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['outcome4', 'outcome5'])

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('FOO', Foo(),
                               transitions={'outcome1':'BAR',
                                             'outcome2':'outcome4'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome2':'FOO'})

    # Execute SMACH plan
    outcome = sm.execute()
```

12/08/22

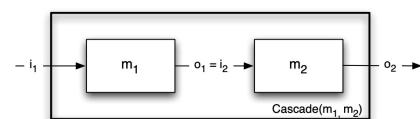
AA 274 | Lecture 19

39

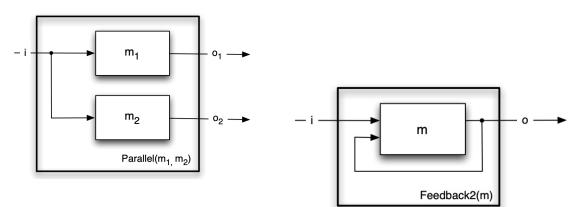
SMACH: Composition

- The composition operations described earlier (cascade, parallel, feedback) are also possible in SMACH

Cascade -> `smach.Sequence`



Parallel -> `smach.Concurrence`



Feedback -> `smach.Iterator`

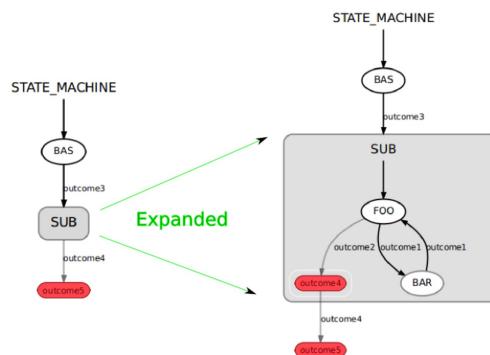
12/08/22

AA 274 | Lecture 19

40

SMACH: Visualization

- The package `smach_visualizer` allows you to easily inspect and monitor your state machine



Thanks for a great quarter!

