Visual Navigation for Flying Robots

Lecture Notes
Summer Term 2012

Lecturer: Dr. Jürgen Sturm
Teaching Assistant: Nikolas Engelhard

http://vision.in.tum.de/teaching/ss2012/visnav2012
Acknowledgements

This slide set would not have been possible without the help and support of many other people. In particular, I would like to thank all my great colleagues who made their lecture slides available for everybody on the internet or sent them to me personally.

My thanks go to (in alphabetical order)

- Alexander Kleiner
- Andrew Davison
- Andrew Zisserman
- Antonio Torralba
- Chad Jenkins
- Cyrill Stachniss
- Daniel Cremers
- Georgio Grisetti
- Jan Peters
- Jana Kosecka
- Jörg Müller
- Jürgen Hess
- Kai Arras
- Kai Wurm
- Kurt Konolige
- Li Fei-Fei
- Maxim Likhachev
- Margaritha Chli
- Nicholas Roy
- Paul Newman
- Richard Newcombe
- Richard Szeliski
- Roland Siegwart
- Sebastian Thrun
- Steve Seitz
- Steven Lavalle
- Szymon Rusinkiewicz
- Volker Grabe
- Vijay Kumar
- Wolfram Burgard
# Table of Contents

Introduction ................................................................................................................. 1

3D Geometry and Sensors ........................................................................................... 17

Probabilistic Models and State Estimation ................................................................. 37

Robot Control ............................................................................................................... 55

Visual Motion Estimation ............................................................................................. 69

Simultaneous Localization and Mapping (SLAM) ....................................................... 85

Bundle Adjustment and Stereo Correspondence ....................................................... 101

Place Recognition, ICP, and Dense Reconstruction ................................................... 117

Motion Planning .......................................................................................................... 133

Planning under Uncertainty, Exploration and Coordination ..................................... 149

Experimentation, Evaluation and Benchmarking ......................................................... 165
Welcome

Dr. Jürgen Sturm

Who are we?

- Computer Vision group:
  1 Professor, 2 Postdocs, 7 PhD students
- Research topics:
  Optical flow and motion estimation, 3D reconstruction, image segmentation, convex optimization
- My research goal:
  Apply solutions from computer vision to real-world problems in robotics.

Goal of this Course

- Provide an overview on problems/approaches for autonomous quadrocopters
- Strong focus on vision as the main sensor
- Areas covered: Mobile Robotics and Computer Vision
- Hands-on experience in lab course

Course Material

  http://szeliski.org/Book/

Lecture Plan

1. Introduction
2. Robots, sensor and motion models
3. State estimation and control
4. Guest talks
5. Feature detection and matching
6. Motion estimation
7. Simultaneous localization and mapping
8. Stereo correspondence
9. 3D reconstruction
10. Navigation and path planning
11. Exploration
12. Evaluation and Benchmarking

Organization

- Tue 10:15-11:45
  - Lectures, discussions
  - Lecturer: Jürgen Sturm
- Thu 14:15-15:45
  - Lab course, homework & programming exercises
  - Teaching assistant: Nikolas Engelhard

Course website
- Dates, additional material
- Exercises, deadlines
- http://cvpr.in.tum.de/teaching/ss2012/visnav2012
Lab Course

- Thu 14:15 – 15:45, given by Nikolas Engelhard
  - Exercises: room 02.09.23 (6x, obliged, homework discussion)
  - Robot lab: room 02.09.34/36 (in weeks without exercises, in case you need help, recommended!)

Exercises Plan

- Exercise sheets contain both theoretical and programming problems
- 3 exercise sheets + 1 mini-project
- Deadline: before lecture (Tue 10:15)
- Hand in by email (visnav2012@cvpr.in.tum.de)

Group Assignment and Schedule

- 3 Ardrones (red/green/blue) + Joystick + 2x Batteries + Charger + PC
- 20 students in the course, 2-3 students per group → 7-8 groups
- Either use lab computers or bring own laptop (recommended)
- Will put up lists for groups and robot schedule in robot lab (room 02.09.36)

VISNAV2012: Team Assignment

<table>
<thead>
<tr>
<th>Team Name</th>
<th>Student Name</th>
<th>Student Name</th>
</tr>
</thead>
</table>

VISNAV2012: Robot Schedule

- Each team gets one time slot with programming support
- The robots/PCs are also available during the rest of the week (but without programming support)

<table>
<thead>
<tr>
<th>Team</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thu 2pm – 3pm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu 3pm – 4pm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu 4pm – 5pm</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Safety Warning

- Quadrocopters are dangerous objects
- Read the manual carefully before you start
- Always use the protective hull
- If somebody gets injured, report to us so that we can improve safety guidelines
- If something gets damaged, report it to us so that we can fix it
- NEVER TOUCH THE PROPELLORS
- DO NOT TRY TO CATCH THE QUADROCOPTER WHEN IT FAILS – LET IT FALL/CRASH!
## Agenda for Today
- History of mobile robotics
- Brief intro on quadrocopters
- Paradigms in robotics
- Architectures and middleware

## General background
- Autonomous, automaton
  - self-willed (Greek, auto+matos)
- Robot
  - Karel Capek in 1923 play R.U.R. (Rossum’s Universal Robots)
  - labor (Czech or Polish, robota)
  - workman (Czech or Polish, robotnik)

## History
In 1966, Marvin Minsky at MIT asked his undergraduate student Gerald Jay Sussman to “spend the summer linking a camera to a computer and getting the computer to describe what it saw”. We now know that the problem is slightly more difficult than that. (Szeliski 2009, Computer Vision)

### Shakey the Robot (1966-1972)

### Stanford Cart (1961-80)
Rhino and Minerva (1998-99)
- Museum tour guide robots
- University of Bonn and CMU
- Deutsches Museum, Smithsonian Museum

Roomba (2002)
- Sensor: one contact sensor
- Control: random movements
- Over 5 million units sold

Neato XV-11 (2010)
- Sensors:
  - 1D range sensor for mapping and localization
  - Improved coverage

Darpa Grand Challenge (2005)

Kiva Robotics (2007)
- Pick, pack and ship automation

Fork Lift Robots (2010)
Quadrocopters (2001-)

Aggressive Maneuvers (2010)

Autonomous Construction (2011)

Mapping with a Quadrocopter (2011)

Our Own Recent Work (2011-)
- RGB-D SLAM (Nikolas Engelhard)
- Visual odometry (Frank Steinbrücker)
- Camera-based navigation (Jakob Engel)

Current Trends in Robotics
Robots are entering novel domains
- Industrial automation
- Domestic service robots
- Medical, surgery
- Entertainment, toys
- Autonomous cars
- Aerial monitoring/inspection/construction
Flying Robots

- Recently increased interest in flying robots
  - Shift focus to different problems (control is much more difficult for flying robots, path planning is simpler, ...)
- Especially quadrocopters because
  - Can keep position
  - Reliable and compact
  - Low maintenance costs
- Trend towards miniaturization

Application Domains of Flying Robots

- Stunts for action movies, photography, sportscasts
- Search and rescue missions
- Aerial photogrammetry
- Documentation
- Aerial inspection of bridges, buildings, ...
- Construction tasks
- Military
- Today, quadrocopters are often still controlled by human pilots

Quadrocopter Platforms

- Commercial platforms
  - Ascending Technologies
  - Height Tech
  - Parrot Ardrone
  - ...  
- Community/open-source projects
  - Mikrokopter
  - Paparazzi
  - ...

For more, see http://multicopter.org/wiki/Multicopter_Table

Quadrocopter

- Keep position:
  - Torques of all four rotors sum to zero
  - Thrust compensates for earth gravity
Quadrocopter: Basic Motions

- Ascend
- Descend
- Turn Left
- Turn Right
- Accelerate Forward
- Accelerate Backward
- Accelerate to the Right
- Accelerate to the Left

Autonomous Flight

- Low level control (not covered in this course)
  - Maintain attitude, stabilize
  - Compensate for disturbances
- High level control
  - Compensate for drift
  - Avoid obstacles
  - Localization and Mapping
  - Navigate to point
  - Return to take-off position
  - Person following

Challenges

- Limited payload
- Limited computational power
- Limited sensors
- Limited battery life
- Fast dynamics, needs electronic stabilization
- Quadrocopter is always in motion
- Safety considerations
Robot Ethics

- Where does the responsibility for a robot lie?
- How are robots motivated?
- Where are humans in the control loop?
- How might society change with robotics?
- Should robots be programmed to follow a code of ethics, if this is even possible?

Three Laws of Robotics (Asimov, 1942):

- A robot may not injure a human being or, through inaction, allow a human being to come to harm.
- A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
- A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Robot Design

Imagine that we want to build a robot that has to perform navigation tasks...

How would you tackle this?
- What hardware would you choose?
- What software architecture would you choose?

Robot Hardware/Components

- Sensors
- Actuators
- Control Unit/Software

Evolution of Paradigms in Robotics

- Classical robotics (mid-70s)
  - Exact models
  - No sensing necessary
- Reactive paradigms (mid-80s)
  - No models
  - Relies heavily on good sensing
- Hybrid approaches (since 90s)
  - Model-based at higher levels
  - Reactive at lower levels

Classical / hierarchical paradigm

- Inspired by methods from Artificial Intelligence (70’s)
- Focus on automated reasoning and knowledge representation
- STRIPS (Stanford Research Institute Problem Solver): Perfect world model, closed world assumption
- Shakey: Find boxes and move them to designated positions
Classical paradigm: Stanford Cart

- Take nine images of the environment, identify interesting points, estimate depth
- Integrate information into global world model
- Correlate images with previous image set to estimate robot motion
- On basis of desired motion, estimated motion, and current estimate of environment, determine direction in which to move
- Execute motion

Characteristics of hierarchical paradigm

Good old-fashioned Artificial Intelligence (GOFAI):

- Symbolic approaches
- Robot perceives the world, plans the next action, acts
- All data is inserted into a single, global world model
- Sequential data processing

Classical paradigm as horizontal/functional decomposition

Reactive Paradigm

- Sense-act type of organization
- Multiple instances of stimulus-response loops (called behaviors)
- Each behavior uses local sensing to generate the next action
- Combine several behaviors to solve complex tasks
- Run behaviors in parallel, behavior can override (subsume) output of other behaviors

Characteristics of Reactive Paradigm

- Situated agent, robot is integral part of the world
- No memory, controlled by what is happening in the world
- Tight coupling between perception and action via behaviors
- Only local, behavior-specific sensing is permitted (ego-centric representation)
Subsumption Architecture

- Introduced by Rodney Brooks in 1986
- Behaviors are networks of sensing and acting modules (augmented finite state machines)
- Modules are grouped into layers of competence
- Layers can subsume lower layers

Level 1: Avoid

Level 2: Wander

Roomba Robot

- Exercise: Model the behavior of a Roomba robot.

Level 3: Follow Corridor

Navigation with Potential Fields

- Treat robot as a particle under the influence of a potential field
- Robot travels along the derivative of the potential
- Field depends on obstacles, desired travel directions and targets
- Resulting field (vector) is given by the summation of primitive fields
- Strength of field may change with distance to obstacle/target
Primitive Potential Fields

- Uniform
- Perpendicular
- Attractive
- Repulsive
- Tangential

Example: reach goal and avoid obstacles

Corridor Following Robot

- Level 1 (collision avoidance)
  add repulsive fields for the detected obstacles
- Level 2 (wander)
  add a uniform field into a (random) direction
- Level 3 (corridor following)
  replaces the wander field by three fields (two perpendicular, one parallel to the walls)

Characteristics of Potential Fields

- Simple method which is often used
- Easy to visualize
- Easy to combine different fields (with parameter tuning)
- But: Suffer from local minima
  - Random motion to escape local minimum
  - Backtracking
  - Increase potential of visited regions
  - High-level planner

Hybrid deliberative/reactive Paradigm

- Combines advantages of previous paradigms
  - World model used in high-level planning
  - Closed-loop, reactive low-level control

Modern Robot Architectures

- Robots became rather complex systems
- Often, a large set of individual capabilities is needed
- Flexible composition of different capabilities for different tasks
Best Practices for Robot Architectures

- Modular
- Robust
- De-centralized
- Facilitate software re-use
- Hardware and software abstraction
- Provide introspection
- Data logging and playback
- Easy to learn and to extend

Robotic Middleware

- Provides infrastructure
- Communication between modules
- Data logging facilities
- Tools for visualization
- Several systems available
  - Open-source: ROS (Robot Operating System), Player/Stage, CARMEN, YARP, OROCOS
  - Closed-source: Microsoft Robotics Studio

Example Architecture for Navigation

```
User interface / mission planning
Global path planning
Localization module
Local path planning + collision avoidance
Sensor interface(s)
Actuator interface(s)
Sensor driver(s)
Actuator driver(s)
Robot Hardware
```

Stanley’s Software Architecture

```
Touareg interface
Laser mapper
Wireless E-Stop
Top level control
Laser 2 interface
Laser 3 interface
Laser 4 interface
Laser 5 interface
Camera interface
Radar interface
Vision mapper
Radar mapper
UKF Pose estimation
Wheel velocity
GPS position
GPS compass
IMU interface
Surface assessment
Health monitor
Road finder
Laser map
Vehicle state (pose, velocity)
velocity limit
map
vision map
vehicle state
obstacle list
trajectory
RDDF database
driving mode
pause/disable command
Power server interface
Process controller
GLOBAL SERVICES
health status
data
Data logger
File system
Communication requests
Communication channels
Inter-process communication (IPC) server
Time server
road center
```

PR2 Software Architecture

- Two 7-DOF arms, grippers, torso, 2-DOF head
- 7 cameras, 2 laser scanners
- Two 8-core CPUs, 3 network switches
- 73 nodes, 328 message topics, 174 services

Communication Paradigms

- Message-based communication
- Direct (shared) memory access
## Forms of Communication

- Push
- Pull
- Publisher/subscriber
- Publish to blackboard
- Remote procedure calls / service calls
- Preemptive tasks / actions

### Push
- Broadcast
- One-way communication
- Send as the information is generated by the producer P

### Pull
- Data is delivered upon request by the consumer C (e.g., a map of the building)
- Useful if the consumer C controls the process and the data is not required (or available) at high frequency

### Publisher/Subscriber
- The consumer C requests a subscription for the data by the producer P (e.g., a camera or GPS)
- The producer P sends the subscribed data as it is generated to C
- Data generated according to a trigger (e.g., sensor data, computations, other messages, …)

### Publish to Blackboard
- The producer P sends data to the blackboard (e.g., parameter server)
- A consumer C pull data from the blackboard B
- Only the last instance of data is stored in the blackboard B

### Service Calls
- The client C sends a request to the server S
- The server returns the result
- The client waits for the result (synchronous communication)
- Also called: Remote Procedure Call
Actions (Preemptive Tasks)
- The client requests the execution of an enduring action (e.g., navigate to a goal location)
- The server executes this action and sends continuously status updates
- Task execution may be canceled from both sides (e.g., timeout, new navigation goal,...)

Robot Operating System (ROS)
- We will use ROS in the lab course
- [http://www.ros.org/](http://www.ros.org/)
- Installation instructions, tutorials, docs

Concepts in ROS
- Nodes: programs that communicate with each other
- Messages: data structure (e.g., “Image”)
- Topics: typed message channels to which nodes can publish/subscribe (e.g., “/camera1/image_color”)
- Parameters: stored in a blackboard

Software Management
- Package: atomic unit of building, contains one or more nodes and/or message definitions
- Stack: atomic unit of releasing, contains several packages with a common theme
- Repository: contains several stacks, typically one repository per institution

Useful Tools
- roscreate-pkg
- rosmake
- roscore
- rosnode list/info
- rostopic list/echo
- rosbag record/play
- rosrun

Tutorials in ROS
<table>
<thead>
<tr>
<th>Exercise Sheet 1</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ On the course website</td>
<td>▪ History of mobile robotics</td>
</tr>
<tr>
<td>▪ Solutions are due in 2 weeks (May 1&lt;sup&gt;st&lt;/sup&gt;)</td>
<td>▪ Brief intro on quadrocopters</td>
</tr>
<tr>
<td>▪ Theory part:</td>
<td>▪ Paradigms in robotics</td>
</tr>
<tr>
<td>Define the motion model of a quadrocopter (will be covered next week)</td>
<td>▪ Architectures and middleware</td>
</tr>
<tr>
<td>▪ Practical part:</td>
<td></td>
</tr>
<tr>
<td>Playback a bag file with data from quadrocopter &amp; plot trajectory</td>
<td></td>
</tr>
</tbody>
</table>

**Questions?**

▪ See you next week!
Organization: Lecture

- Student request to change lecture time to Tuesday afternoon due to time conflicts with other course
- Problem: At least 3 students who are enrolled for this lecture have time Tuesday morning but not on Tuesday afternoon
- Therefore: No change
- Lectures are important, please choose which course to follow
- Note: Still students on the waiting list

Organization: Lab Course

- Robot lab: room 02.09.38 (around the corner)
- Exercises: room 02.09.23 (here)
- You have to sign up for a team before May 1st (team list in student lab)
- After May 1st, remaining places will be given to students on waiting list
- This Thursday: Visual navigation demo at 2pm in the student lab (in conjunction with TUM Girls’ Day)

Today’s Agenda

- Linear algebra
- 2D and 3D geometry
- Sensors

Vectors

- Vector and its coordinates
  \[ \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n \]
- Vectors represent points in an n-dimensional space

Vector Operations

- Scalar multiplication
- Addition/subtraction
- Length
- Normalized vector
- Dot product
- Cross product
Vector Operations

- Scalar multiplication
- Addition/subtraction
- Length
- Normalized vector
- Dot product
- Cross product

\[ \|x\|_2 = \|x\| = \sqrt{x_1^2 + x_2^2 + \ldots} \]

\[ \hat{x} = \frac{x}{\|x\|} \]

Cross Product

- Definition
- Matrix notation for the cross product
- Verify that \( x \times y = [x]_\times y \)
Matrices

- Rectangular array of numbers
  \[ X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix} \in \mathbb{R}^{n \times m} \]
  - First index refers to row
  - Second index refers to column

- Column vectors of a matrix
  \[ X = \begin{pmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{pmatrix} \]
  - Geometric interpretation: for example, column vectors can form basis of a coordinate system

- Row vectors of a matrix
  \[ X = \begin{pmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{pmatrix} \]

Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- Matrix-vector multiplication
- Matrix-matrix multiplication
- Inversion

Matrices

- Square matrix
- Diagonal matrix
- Upper and lower triangular matrix
- Symmetric matrix
- Skew-symmetric matrix
- (Semi-)positive definite matrix
- Invertible matrix
- Orthonormal matrix
- Matrix rank

\[ a^T X a \geq 0 \]
Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- Matrix-vector multiplication
- Matrix-matrix multiplication
- Inversion

Matrix-Vector Multiplication

Definition

\[ X \cdot b = \sum_{k=1}^{n} x_{ik} \cdot b_k \]

Geometric interpretation:
A linear combination of the columns of \( X \) scaled by the coefficients of \( b \) → coordinate transformation

Matrix-Matrix Multiplication

Operator \( \mathbb{R}^{n \times m} \times \mathbb{R}^{m \times p} \rightarrow \mathbb{R}^{n \times p} \)

Definition

\[ C = AB \]

\[ = A \left( b_{11} \quad b_{12} \quad \cdots \quad b_{1p} \right) \]

Interpretation: transformation of coordinate systems
Can be used to concatenate transforms

Matrix Multiplication

Not commutative (in general)
\( AB \neq BA \)

Associative
\( A(BC) = (AB)C \)

Transpose
\( (AB)^T = B^T A^T \)
Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- Matrix-vector multiplication
- Matrix-matrix multiplication
- Inversion

Matrix Inversion

- If \( A \) is a square matrix of full rank, then there is a unique matrix \( B = A^\top \) such that \( AB = I \).
- Different ways to compute, e.g., Gauss-Jordan elimination, LU decomposition, ...
- When \( A \) is orthonormal, then \( A^{-1} = A^\top \)

Recap: Linear Algebra

- Vectors
- Matrices
- Operators

- Now let’s apply these concepts to 2D+3D geometry

Geometric Primitives in 2D

- 2D point \( \mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2 \)

- Augmented vector \( \tilde{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \in \mathbb{R}^3 \)

- Homogeneous coordinates \( \tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^2 \)

Geometric Primitives in 2D

- 2D line \( \tilde{\mathbf{l}} = (a, b, c)^\top \)

- 2D line equation \( \tilde{x} \cdot \tilde{\mathbf{l}} = ax + by + c = 0 \)

Homogeneous vectors that differ only by scale represent the same 2D point

Convert back to inhomogeneous coordinates by dividing through last element

\[
\tilde{x} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \tilde{x} / \tilde{w} \\ \tilde{y} / \tilde{w} \\ 1 \end{pmatrix} = \tilde{w} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \tilde{w} \tilde{x}
\]

Points with \( \tilde{w} = 0 \) are called points at infinity or ideal points
Geometric Primitives in 2D

- Normalized line equation vector
  \[ \mathbf{l} = (\hat{n}_x, \hat{n}_y, d)^T = (\hat{n}, d)^T \] with \( \|\hat{n}\| = 1 \)
  where \( d \) is the distance of the line to the origin

Geometric Primitives in 2D

- Line joining two points
  \[ \mathbf{l} = \hat{x}_1 \times \hat{x}_2 \]

- Intersection point of two lines
  \[ \hat{x} = \mathbf{l}_1 \times \mathbf{l}_2 \]

Geometric Primitives in 3D

- 3D point
  \( x = (x, y, z)^T \in \mathbb{R}^3 \)

- Augmented vector
  \[ \bar{x} = (x, y, z, 1)^T \in \mathbb{R}^4 \]

- Homogeneous coordinates
  \[ \bar{x} = (\bar{x}, \bar{y}, \bar{z}, \bar{w}) \in \mathbb{R}^3 \]

Geometric Primitives in 3D

- 3D line
  \[ \mathbf{m} = (a, b, c, d)^T \]

- 3D plane equation
  \[ \bar{x} \cdot \mathbf{m} = ax + by + cz + d = 0 \]

- Normalized plane
  with unit normal vector
  \[ \mathbf{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d)^T = (\hat{n}, d) \] \( (\|\hat{n}\| = 1) \)
  and distance \( d \)

Geometric Primitives in 2D

- Polar coordinates of a line: \((\theta, d)^T\)
  (e.g., used in Hough transform for finding lines)
  \[ \hat{n} = (\cos \theta, \sin \theta)^T \]
2D Planar Transformations

- Translation
  \[ x' = x + t \]
  where \( I \) is the identity matrix (2x2) and \( \mathbf{0} \) is the zero vector

- Rotation + translation (2D rigid body motion, or 2D Euclidean transformation)
  \[ x' = Rx + t \]
  \[ \begin{bmatrix} 1 & 0 & t \end{bmatrix} \bar{x} \]
  or
  \[ x' = \begin{bmatrix} \cos \theta & -\sin \theta & \mathbf{0}^T \\ \sin \theta & \cos \theta & \mathbf{0}^T \\ \mathbf{0}^T & \mathbf{0}^T & 1 \end{bmatrix} \bar{x} \]
  where \( R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \) is an orthonormal rotation matrix, i.e., \( RR^T = I \)

- Distances (and angles) are preserved

- Scaled rotation/similarity transform
  \[ x' = sRx + t \]
  \[ \begin{bmatrix} s \cos \theta & -s \sin \theta & \mathbf{0}^T \\ s \sin \theta & s \cos \theta & \mathbf{0}^T \\ \mathbf{0}^T & \mathbf{0}^T & 1 \end{bmatrix} \bar{x} \]
  where \( s \) is a scalar

- Preserves angles between lines

- Affine transform
  \[ x' = A\bar{x} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \bar{x} \]

- Parallel lines remain parallel

- Projective/perspective transform
  \[ x' = \bar{H} \bar{x} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \bar{x} \]

- Note that \( \bar{H} \) is homogeneous (only defined up to scale)
- Resulting coordinates are homogeneous
- Parallel lines remain parallel
2D Transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Matrix</th>
<th># DoF</th>
<th>Preserves</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>translation</td>
<td>$[I \</td>
<td>t]_{2 \times 3}$</td>
<td>2</td>
<td>orientation</td>
</tr>
<tr>
<td>rigid (Euclidean)</td>
<td>$[R \</td>
<td>t]_{2 \times 3}$</td>
<td>3</td>
<td>lengths</td>
</tr>
<tr>
<td>similarity</td>
<td>$[sR \</td>
<td>t]_{2 \times 3}$</td>
<td>4</td>
<td>angles</td>
</tr>
<tr>
<td>affine</td>
<td>$[A]_{2 \times 3}$</td>
<td>6</td>
<td>parallelism</td>
<td></td>
</tr>
<tr>
<td>projective</td>
<td>$[\tilde{H}]_{2 \times 3}$</td>
<td>8</td>
<td>straight lines</td>
<td></td>
</tr>
</tbody>
</table>

3D Transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Matrix</th>
<th># DoF</th>
<th>Preserves</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>translation</td>
<td>$[I \</td>
<td>t]_{3 \times 4}$</td>
<td>3</td>
<td>orientation</td>
</tr>
<tr>
<td>rigid (Euclidean)</td>
<td>$[R \</td>
<td>t]_{3 \times 4}$</td>
<td>6</td>
<td>lengths</td>
</tr>
<tr>
<td>similarity</td>
<td>$[sR \</td>
<td>t]_{3 \times 4}$</td>
<td>7</td>
<td>angles</td>
</tr>
<tr>
<td>affine</td>
<td>$[A]_{3 \times 4}$</td>
<td>12</td>
<td>parallelism</td>
<td></td>
</tr>
<tr>
<td>projective</td>
<td>$[\tilde{H}]_{4 \times 4}$</td>
<td>15</td>
<td>straight lines</td>
<td></td>
</tr>
</tbody>
</table>

3D Rotations

- Rotation matrix (also called the special orientation group $SO(3)$)
- Euler angles
- Axis/angle
- Unit quaternion

Rotation Matrix

- Orthonormal 3x3 matrix

$$ R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} $$

- Column vectors correspond to coordinate axes
- Special orientation group $R \in SO(3)$
- Main disadvantage: Over-parameterized (9 parameters instead of 3)

Euler Angles

- Product of 3 consecutive rotations
- Roll-pitch-yaw convention is very common in aerial navigation (DIN 9300)

Translation

$$ \bar{x}' = \begin{pmatrix} 1 & t \\ 0^T & 1 \end{pmatrix} \bar{x} $$

Euclidean transform (translation + rotation), (also called the Special Euclidean group SE(3))

$$ \bar{x}' = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \bar{x} $$

Scaled rotation, affine transform, projective transform...
Euler Angles

- Yaw $\psi$, Pitch $\theta$, Roll $\phi$ to rotation matrix
  \[
  R = R_z(\phi)R_y(\theta)R_x(\psi)
  \]
  \[
  = \begin{pmatrix}
  1 & 0 & 0 \\
  0 & \cos \phi & \sin \phi \\
  0 & -\sin \phi & \cos \phi
  \end{pmatrix}
  \begin{pmatrix}
  \cos \theta & 0 & -\sin \theta \\
  0 & 1 & 0 \\
  \sin \theta & 0 & \cos \theta
  \end{pmatrix}
  \begin{pmatrix}
  \cos \psi & \sin \psi & 0 \\
  -\sin \psi & \cos \psi & 0 \\
  0 & 0 & 1
  \end{pmatrix}
  \]
- Rotation matrix to Yaw-Pitch-Roll
  \[
  \phi = \text{Atan2} \left( -r_{31}, \sqrt{r_{11}^2 + r_{21}^2} \right)
  \]
  \[
  \psi = \text{Atan2} \left( -r_{32}, \frac{r_{31}}{\cos(\phi)} \right)
  \]
  \[
  \theta = \text{Atan2} \left( -r_{33}, \frac{r_{23}}{\cos(\phi)} \right)
  \]

Gimbal Lock

- When the axes align, one degree-of-freedom (DOF) is lost...

Axis/Angle

- Represent rotation by
  - rotation axis $\hat{n}$
  - rotation angle $\theta$
  - 4 parameters $(\hat{n}, \theta)$

- 3 parameters $\omega = \theta \hat{n}$
  - length is rotation angle
  - also called the angular velocity
  - minimal but not unique (why?)

Derivation of Angular Velocities

- Assume we have a rotational motion in $SO(3)$
  \[ R(t) \in SO(3) \quad t \in \mathbb{R} \]

- As this rotations are orthonormal matrices, we have
  \[ R(t)R^T(t) = I \]

- Now take the derivative on both sides (w.r.t. $t$)
  \[
  \dot{R}(t)R^T(t) + R(t)\dot{R}^T(t) = 0
  \]
  \[
  \dot{R}(t)R^T(t) = -(\dot{R}(t)R^T(t))^T
  \]

- Thus, $\dot{R}(t)R^T(t)$ must be skew-symmetric, i.e.,
  \[ [\omega(t)]_x = \dot{R}(t)R^T(t) \]

Derivation of Angular Velocities

- Linear ordinary differential equation (ODE)
  \[
  \dot{R}(t) = [\omega]_x R(t) = \begin{pmatrix}
  0 & -\omega_3 & \omega_2 \\
  \omega_3 & 0 & -\omega_1 \\
  -\omega_2 & \omega_1 & 0
  \end{pmatrix} R(t)
  \]

- Solution of this ODE
  \[ R(t) = \exp([\omega]_x) R(0) \]

- Conversions
  \[ R = \exp([\omega]_x) \quad [\omega]_x = \log R \]
Derivation of Angular Velocities

→ Linear ordinary differential equation (ODE)

\[ \dot{R}(t) = [\omega] \times R(t) = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} R(t) \]

- The space of all skew-symmetric matrices is called the tangent space
  \[ \text{so}(3) = \{ [\omega] \times \mid \omega \in \mathbb{R}^3 \} \]
- Space of all rotations in 3D (Special orientation group)
  \[ \text{SO}(3) = \{ R \in \mathbb{R}^{3 \times 3} \mid R^T R = I, \det R = 1 \} \]

Exponential Twist

- The exponential map can be generalized to Euclidean transformations (incl. translations)
- Tangent space \( \text{se}(3) = \text{so}(3) \times \mathbb{R}^3 \)
- (Special) Euclidean group \( \text{SE}(3) = \text{SO}(3) \times \mathbb{R}^3 \)
  (group of all Euclidean transforms)
- Rigid body velocity
  \[ \xi = (\omega_x, \omega_y, \omega_z, v_x, v_y, v_z) \in \mathbb{R}^6 \]
  angular vel. linear vel.

Unit Quaternions

- Quaternion \( q = (q_x, q_y, q_z, q_w)^T \in \mathbb{R}^4 \)
- Unit quaternions have \( \| q \| = 1 \)
- Opposite sign quaternions represent the same rotation \( q = -q \)
- Otherwise unique

Conversion

- Rodriguez’ formula
  \[ R(\mathbf{n}, \theta) = I + \sin \theta[\mathbf{n}]_\times + (1 - \cos \theta)[\mathbf{n}]_\times^2 \]
- Inverse
  \[ \theta = \cos^{-1} \left( \frac{\text{trace}(R) - 1}{2} \right), \quad \mathbf{n} = \frac{1}{2 \sin \theta} \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} \]

see: An Invitation to 3D Vision, Y. Ma, S. Soatto, J. Kosecka, S. Sastry, Chapter 2 (available online)

Exponential Twist

- Convert to homogeneous coordinates
  \[ \mathbf{\dot{\xi}} = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \\ 0 & 0 & 0 \end{pmatrix} \] \( \in \text{se}(3) \)
- Exponential map between \( \text{se}(3) \) and \( \text{SE}(3) \)
  \[ M = \exp \mathbf{\dot{\xi}} \]
- There are also direct formulas (similar to Rodriguez)

Unit Quaternions

- Advantage: multiplication and inversion operations are really fast
- Quaternion-Quaternion Multiplication
  \[ q_0 q_1 = (v_0, w_0)(v_1, w_1) = (v_0 \times v_1 + w_0 v_1 + w_1 v_0, w_0 w_1 - v_0 v_1) \]
- Inverse (flip sign of \( v \) or \( w \))
  \[ q_0/q_1 = (v_0, w_0)/(v_1, w_1) = (v_0, w_0)(v_1, -w_1) = (v_0 \times v_1 + w_0 v_1 - w_1 v_0, -w_0 w_1 - v_0 v_1) \]
Unit Quaternions

- Quaternion-Vector multiplication (rotate point \( p \) with rotation \( q \))
  \[
  p' = v\tilde{p}/q
  \]
  with \( \tilde{p} = (x, y, z, 0)^\top \)
- Relation to Axis/Angle representation
  \[
  q = (v, w) = (\sin \frac{\theta}{2}\hat{n}, \cos \frac{\theta}{2})
  \]

Spherical Linear Interpolation (SLERP)

- Useful for interpolating between two rotations
  
  \[
  \text{procedure slerp}(q_0, q_1, \alpha): \\
  1. q_r = q_1/q_0 = (v_r, w_r) \\
  2. \text{if } w_r < 0 \text{ then } q_r = -q_r \\
  3. \theta_r = 2\tan^{-1}(||v_r||/w_r) \\
  4. \hat{n}_r = N(v_r) = v_r/||v_r|| \\
  5. \theta_s = \alpha \theta_r \\
  6. q_s = (\sin \frac{\theta_s}{2}\hat{n}_r, \cos \frac{\theta_s}{2})
  \]

3D to 2D Projections

- Orthographic projections
- Perspective projections

3D to 2D Perspective Projection

- 3D point \( p \) (in the camera frame)
- 2D point \( x \) (on the image plane)
- Pin-hole camera model
  \[
  \tilde{x} = \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 
  \end{pmatrix} \tilde{p}
  \]
  - Remember, \( \tilde{x} \) is homogeneous, need to normalize
    \[
    \tilde{x} = \begin{pmatrix}
    \tilde{x} \\
    \tilde{y} \\
    \tilde{z}
  \end{pmatrix} \Rightarrow x = \begin{pmatrix}
  \tilde{x}/\tilde{z} \\
  \tilde{y}/\tilde{z}
  \end{pmatrix}
  \]
Camera Intrinsics
- So far, 2D point is given in meters on image plane
- But: we want 2D point be measured in pixels (as the sensor does)

Camera Extrinsic
- Assume $\tilde{p}_w$ is given in world coordinates
- Transform from world to camera (also called the camera extrinsics)
  \[ \tilde{p} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \tilde{p}_w \]
- Full camera matrix
  \[ \tilde{x} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tilde{p} \]
- Focal length $f_x, f_y$
- Camera center $c_x, c_y$
- Skew $s$

Recap: 2D/3D Geometry
- points, lines, planes
- 2D and 3D transformations
- Different representations for 3D orientations
  - Choice depends on application
  - Which representations do you remember?
- 3D to 2D perspective projections
- You really have to know 2D/3D transformations by heart (read Szeliski, Chapter 2)

C++ Libraries for Lin. Alg./Geometry
- Many C++ libraries exist for linear algebra and 3D geometry
- Typically conversion necessary
- Examples:
  - C arrays, std::vector (no linear alg. functions)
  - gsl (gnu scientific library, many functions, plain C)
  - boost::array (used by ROS messages)
  - Bullet library (3D geometry, used by ROS tf)
  - Eigen (both linear algebra and geometry, my recommendation)

Example: Transform Trees in ROS
- TF package represents 3D transforms between rigid bodies in the scene as a tree
### Classification of Sensors

- **What:**
  - Proprioceptive sensors
    - Measure values internally to the system (robot)
    - Examples: battery status, motor speed, accelerations, ...
  - Exteroceptive sensors
    - Provide information about the environment
    - Examples: compass, distance to objects, ...
- **How:**
  - Passive sensors
    - Measure energy coming from the environment
  - Active sensors
    - Emit their proper energy and measure the reaction
    - Better performance, but influence on environment

### Example: Ardrone Sensors

- Tactile sensors
  - Contact switches, bumpers, proximity sensors, pressure
- Wheel/motor sensors
  - Potentiometers, brush/optical/magnetic/inductive/capacitive encoders, current sensors
- Heading sensors
  - Compass, infrared, inclinometers, gyroscopes, accelerometers
- Ground-based beacons
  - GPS, optical or RF beacons, reflective beacons
- Active ranging
  - Ultrasonic sensor, laser rangefinder, optical triangulation, structured light
- Motion/speed sensors
  - Doppler radar, Doppler sound
- Vision-based sensors
  - CCD/CMOS cameras, visual servoing packages, object tracking packages

### Characterization of Sensor Performance

- Bandwidth or Frequency
- Delay
- Sensitivity
- Cross-sensitivity (cross-talk)
- Error (accuracy)
  - Deterministic errors (modeling/calibration possible)
  - Random errors
- Weight, power consumption, ...
Sensors

- Motor/wheel encoders
- Compass
- Gyroscope
- Accelerometers
- GPS
- Range sensors
- Cameras

Motor/wheel encoders

- Working principle:
  - Regular: counts the number of transitions but cannot tell direction
  - Quadrature: uses two sensors in quadrature phase-shift, ordering of rising edge tells direction
  - Sometimes: Reference pulse (or zero switch)

Magnetic Compass

- Sensing principle: Hall sensor
- Construction: 3 orthogonal sensors

Magnetic Declination

- Angle between magnetic north and true north
- Varies over time
- Good news ;-) by 2050, magnetic declination in central Europe will be zero

Motor/wheel encoders

- Device for measuring angular motion
- Often used in (wheeled) robots
- Output: position, speed (possibly integrate speed to get odometry)
<table>
<thead>
<tr>
<th>Mechanical Gyroscope</th>
<th>Modern Gyroscopes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measures orientation (standard gyro) or angular velocity (rate gyro, needs integration for angle)</td>
<td>Vibrating structure gyroscope (MEMS)</td>
</tr>
<tr>
<td>Spinning wheel mounted in a gimbal device (can move freely in 3 dimensions)</td>
<td>Based on Coriolis effect</td>
</tr>
<tr>
<td>Wheel keeps orientation due to angular momentum (standard gyro)</td>
<td>“Vibration keeps its direction under rotation”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Accelerometer</th>
<th>MEMS Accelerometers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measures all external forces acting upon them (including gravity)</td>
<td>Micro Electro-Mechanical Systems (MEMS)</td>
</tr>
<tr>
<td>Acts like a spring-damper system</td>
<td>Spring-like structure with a proof mass</td>
</tr>
<tr>
<td>To obtain inertial acceleration (due to motion alone), gravity must be subtracted</td>
<td>Damping results from residual gas</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inertial Measurement Unit</th>
<th>Inertial Measurement Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-axes MEMS gyroscope</td>
<td>IMU: Device that uses gyroscopes and accelerometers to estimate (relative) position, orientation, velocity and accelerations</td>
</tr>
<tr>
<td>Provides angular velocity</td>
<td>Integrate angular velocities to obtain absolute orientation</td>
</tr>
<tr>
<td>Integrate for angular position</td>
<td>Subtract gravity from acceleration</td>
</tr>
<tr>
<td>Problem: Drifts slowly over time (e.g., 1deg/hour), called the bias</td>
<td>Integrate acceleration to linear velocities</td>
</tr>
<tr>
<td>3-axes MEMS accelerometer</td>
<td>Integrate linear velocities to position</td>
</tr>
<tr>
<td>Provides accelerations (including gravity)</td>
<td>Note: All IMUs are subject to drift (position is integrated twice!), needs external reference</td>
</tr>
</tbody>
</table>
| Can we use these sensors to estimate our position? |"
**Example: AscTec Autopilot Board**

*Image of a circuit board with labels.*

**GPS**

- 24+ satellites, 12-hour orbit, 20.190 km height
- 6 orbital planes, 4+ satellites per orbit, 60° distance
- Satellite transmits orbital location + time
- 50 bits/s, msg has 1500 bits → 12.5 minutes

**Range Sensors**

- Sonar
- Laser range finder
- Time of flight camera
- Structured light (will be covered later)

---

**GPS**

- Position from pseudorange
  - Requires measurements of 4 different satellites
  - Low accuracy (3-15 m) but absolute
- Position from pseudorange + phase shift
  - Very precise (1 mm) but highly ambiguous
  - Requires reference receiver (RTK/dGPS) to remove ambiguities

**Range Sensors**

- Emit signal to determine distance along a ray
- Make use of propagation speed of ultrasound/light
- Traveled distance is given by $d = c \cdot t$
- Sound speed: 340 m/s
- Light speed: 300,000 km/s
**Ultrasonic Range Sensors**
- Range between 12cm and 5m
- Opening angle around 20 to 40 degrees
- Soft surfaces absorb sound
- Reflections → ghosts
- Lightweight and cheap

**Laser Scanner**
- Measures phase shift
- Pro: High precision, wide field of view, safety approved for collision detection
- Con: Relatively expensive + heavy

**Laser Scanner**
- 2D scanners
- 3D scanners

**Camera**
- Let’s design a camera
  - Idea 1: put a piece of film in front of an object
  - Do we get a reasonable image?

**Camera Lens**
- A lens focuses light onto the film
  - Rays passing through the optical center are not deviated
Camera Lens

- A lens focuses light onto the film
- Rays passing through the center are not deviated
- All rays parallel to the **Optical Axis** converge at the **Focal Point**

Lens Distortions

- Radial distortion of the image
  - Caused by imperfect lenses
  - Deviations are most noticeable for rays that pass through the edge of the lens

Digital Cameras

- Vignetting
- De-bayering
- Rolling shutter and motion blur
- Compression (JPG)
- Noise

Lens Distortions

- Radial distortion of the image
  - Caused by imperfect lenses
  - Deviations are most noticeable for rays that pass through the edge of the lens
  - Typically compensated with a low-order polynomial
    \[
    \hat{x}_c = x_c (1 + \kappa_2 r_c^2 + \kappa_4 r_c^4) \\
    \hat{y}_c = y_c (1 + \kappa_2 r_c^2 + \kappa_4 r_c^4)
    \]

Dead Reckoning and Odometry

- Estimating the position \( x_t \) based on the issued controls (or IMU) readings \( u_t \)
- Integrated over time \( x_t = f(x_{t-1}, u_t) \)
## Exercise Sheet 1

- Odometry sensor on Ardrone is an integrated package
- Sensors
  - Down-looking camera to estimate motion
  - Ultrasonic sensor to get height
  - 3-axes gyroscopes
  - 3-axes accelerometer
- IMU readings $\mathbf{u}_t$
  - Horizontal speed ($v_x/v_y$)
  - Height ($z$)
  - Roll, Pitch, Yaw
- Integrate these values to get robot pose $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t)$
  - Position ($x/y/z$)
  - Orientation (e.g., $r/p/y$)

## Summary

- Linear Algebra
- 2D/3D Geometry
- Sensors
Visual Navigation for Flying Robots

Probabilistic Models and State Estimation

Dr. Jürgen Sturm

Guest Talks

- Camera-Based Navigation of a Low-Cost Quadrocopter (J. Engel, J. Sturm, D. Cremers), Submitted to International Conference on Robotics and Systems (IROS), under review.

Perception

- Perception and models are strongly linked
- Example: Human Perception

Organization

- Next week: Three scientific guest talks
- Recent research results from our group (2011/12)

Perception and models are strongly linked
Models in Human Perception

- Count the black dots

State Estimation

What parts of the world state are (most) relevant for a flying robot?

- Position
- Velocity
- Obstacles
- Map
- Positions and intentions of other robots/humans
- ...

Models and State Estimation

State Estimation

(Deterministic) Sensor Model

- Robot perceives the environment through its sensors

$$z = h(x)$$

- Goal: Infer the state of the world from sensor readings

$$x = h^{-1}(z)$$
(Deterministic) Motion Model

- Robot executes an action $u$
  (e.g., move forward at 1m/s)

- Update belief state according to motion model
  
  \[ x' = g(x, u) \]

Probabilistic Robotics

- Sensor observations are noisy, partial, potentially missing (why?)
- All models are partially wrong and incomplete (why?)
- Usually we have prior knowledge (why?)

Agenda for Today

- Motivation ✓
- Bayesian Probability Theory
- Bayes Filter
- Normal Distribution
- Kalman Filter

The Axioms of Probability Theory

Notation: $P(A)$ refers to the probability that proposition $A$ holds

1. $0 \leq P(A) \leq 1$
2. $P(\Omega) = 1$ \hspace{1cm} $P(\emptyset) = 0$
3. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

A Closer Look at Axiom 3

\[ P(A \cup B) = P(A) + P(B) - P(A \cap B) \]
## Discrete Random Variables

- **X** denotes a **random variable**
- **X** can take on a countable number of values in \( \{x_1, x_2, \ldots, x_n\} \)
- \( P(X = x_i) \) is the **probability** that the random variable **X** takes on value \( x_i \)
- \( P(\cdot) \) is called the **probability mass function**

**Example:**

\[
P(\text{Room}) = \begin{cases} 0.7 & \text{(office)} \\ 0.2 & \text{corridor} \\ 0.08 & \text{lab} \\ 0.02 & \text{kitchen} \end{cases}
\]

## Continuous Random Variables

- **X** takes on continuous values
- \( p(X = x) \) or \( p(x) \) is called the **probability density function** (PDF)

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

**Example**

![Probability Density Function](image)

## Proper Distributions Sum To One

- **Discrete case**
  \[ \sum_x P(x) = 1 \]
- **Continuous case**
  \[ \int p(x) \, dx = 1 \]

## Joint and Conditional Probabilities

- **P(X = x and Y = y) = P(x, y)**
- If \( X \) and \( Y \) are **independent** then
  \[ P(x, y) = P(x)P(y) \]
- \( P(x \mid y) \) is the probability of \( x \) **given** \( y \)
  \[ P(x \mid y) = P(x, y) \]
- If \( X \) and \( Y \) are independent then
  \[ P(x \mid y) = P(x) \]

## Conditional Independence

- **Definition of conditional independence**
  \[ P(x, y \mid z) = P(x \mid z)P(y \mid z) \]
- Equivalent to
  \[
P(x \mid z) = P(x \mid y, z) \\
P(y \mid z) = P(y \mid x, z)
\]
- **Note:** this does not necessarily mean that
  \[ P(x, y) = P(x)P(y) \]

## Marginalization

- **Discrete case**
  \[ P(x) = \sum_y P(x, y) \]
- **Continuous case**
  \[ p(x) = \int p(x, y) \, dy \]
Example: Marginalization

Law of Total Probability

- Discrete case
  \[ P(x) = \sum_y P(x, y) = \sum_y P(x | y)P(y) \]

- Continuous case
  \[ p(x) = \int p(x, y)dy = \int p(x | y)p(y)dy \]

Expected Value of a Random Variable

- Discrete case
  \[ E[X] = \sum_i x_i P(x_i) \]

- Continuous case
  \[ E[X] = \int xP(X = x)dx \]

- The expected value is the weighted average of all values a random variable can take on.
- Expectation is a linear operator
  \[ E[aX + b] = aE[X] + b \]

Covariance of a Random Variable

- Measures the squared expected deviation from the mean
  \[ \text{Cov}[X] = E[X - E[X]]^2 = E[X^2] - E[X]^2 \]

The State Estimation Problem

We want to estimate the world state \( x \)
- From sensor measurements \( z \)
- and controls (or odometry readings) \( u \)

We need to model the relationship between these random variables, i.e.,
\[ p(x | z) \quad p(x' | x, u) \]

Causal vs. Diagnostic Reasoning

- \( P(x | z) \) is diagnostic
- \( P(z | x) \) is causal
- Often causal knowledge is easier to obtain
- Bayes rule allows us to use causal knowledge:

\[
P(x | z) = \frac{P(z | x)P(x)}{P(z)}
\]

prior on world state
observation likelihood
prior on sensor observations
Bayes Formula

\[ P(x, z) = P(x \mid z)P(z) = P(z \mid x)P(x) \]

\[ \Rightarrow \]

\[ P(x \mid z) = \frac{P(z \mid x)P(x)}{P(z)} = \text{likelihood} \cdot \text{prior} \quad \text{evidence} \]

Bayes Rule with Background Knowledge

\[ P(x \mid y, z) = \frac{P(y \mid x, z)P(x \mid z)}{P(y \mid z)} \]

Example: Sensor Measurement

- Binary sensor \( Z \in \{\text{bright}, \text{\neg bright}\} \)
- Binary world state \( X \in \{\text{home}, \text{\neg home}\} \)
- Sensor model \( P(Z = \text{bright} \mid X = \text{home}) = 0.6 \)
  \( P(Z = \text{bright} \mid X = \text{\neg home}) = 0.3 \)
- Prior on world state \( P(X = \text{home}) = 0.5 \)
- Assume: Robot observes light, i.e., \( Z = \text{bright} \)
- What is the probability \( P(X = \text{home} \mid Z = \text{bright}) \) that the robot is above the landing zone?

Normalization

- Direct computation of \( P(z) \) can be difficult
- Idea: Compute improper distribution, normalize afterwards

\[ L(x \mid z) = P(z \mid x)P(x) \]

Step 1:

\[ P(z) = \sum_{x} P(z \mid x)P(x) = \sum_{x} L(x \mid z) \]

Step 2:

\[ P(x \mid z) = \frac{L(x \mid z)}{P(z)} \]

Example: Sensor Measurement

- Quadrocopter seeks the landing zone
- Landing zone is marked with many bright lamps
- Quadrocopter has a brightness sensor

Example: Sensor Measurement

- Sensor model \( P(Z = \text{bright} \mid X = \text{home}) = 0.6 \)
  \( P(Z = \text{bright} \mid X = \text{\neg home}) = 0.3 \)
- Prior on world state \( P(X = \text{home}) = 0.5 \)
- Probability after observation (using Bayes)

\[
P(X = \text{home} \mid Z = \text{noise}) = P(\text{bright} \mid \text{home})P(\text{home})
\]

\[
= \frac{P(\text{bright} \mid \text{home})P(\text{home}) + P(\text{bright} \mid \text{\neg home})P(\text{\neg home})}{0.6 \cdot 0.5 + 0.3 \cdot 0.5} = \frac{0.3 + 0.15}{0.3 + 0.15} = 0.67
\]
Example: Sensor Measurement

- Sensor model: $P(Z = \text{bright} \mid X = \text{home}) = 0.6$
  $P(Z = \text{bright} \mid X = \text{\neg home}) = 0.3$
- Prior on world state: $P(X = \text{home}) = 0.5$
- Probability after observation (using Bayes):
  
  $$P(X = \text{home} \mid Z = \text{noise}) = \frac{P(\text{bright} \mid \text{home})P(\text{home})}{0.6 \cdot 0.5 + 0.3 \cdot 0.5} = \frac{0.3}{0.3 + 0.15} = 0.67$$

Combining Evidence

- Suppose our robot obtains another observation $z_2$ (either from the same or a different sensor)
- How can we integrate this new information?
- More generally, how can we estimate $p(x \mid z_1, z_2, \ldots)$?

  Bayes formula gives us:
  
  $$P(x \mid z_1, \ldots, z_n) = \frac{P(z_n \mid x, z_1, \ldots, z_{n-1})P(x \mid z_1, \ldots, z_{n-1})}{P(z_n \mid z_1, \ldots, z_{n-1})}$$

Recursive Bayesian Updates

- Suppose our robot obtains another observation $z_2$ (either from the same or a different sensor)
- How can we integrate this new information?
- More generally, how can we estimate $p(x \mid z_1, z_2, \ldots)$?

  Bayes formula gives us:
  
  $$P(x \mid z_1, \ldots, z_n) = \frac{P(z_n \mid x, z_1, \ldots, z_{n-1})P(x \mid z_1, \ldots, z_{n-1})}{P(z_n \mid z_1, \ldots, z_{n-1})}$$

- Markov Assumption:
  $z_n$ is independent of $z_1, \ldots, z_{n-1}$ if we know $x$

  $$P(x \mid z_1, \ldots, z_n) = \frac{P(z_n \mid x)P(x \mid z_1, \ldots, z_{n-1})}{P(z_n \mid z_1, \ldots, z_{n-1})}$$

  $$= \eta P(z_n \mid x)P(x \mid z_1, \ldots, z_{n-1})$$
  
  $$= \eta = \prod_{i=1}^{n} P(z_i \mid x)P(x)$$
Example: Second Measurement
- Sensor model $P(Z_2 = \text{marker} \mid X = \text{home}) = 0.8$
  $P(Z_2 = \text{marker} \mid X = \neg\text{home}) = 0.1$
- Previous estimate $P(X = \text{home} \mid Z_1 = \text{bright}) = 0.67$
- Assume robot does not observe marker
- What is the probability of being home?

$$P(X = \text{home} \mid Z_1 = \text{bright}, Z_2 = \neg\text{marker})$$
$$\frac{P(\neg\text{marker} \mid \text{home})P(\text{home} \mid \text{bright})}{P(\neg\text{marker} \mid \text{home})P(\text{home} \mid \text{bright}) + P(\neg\text{marker} \mid \neg\text{home})P(\neg\text{home} \mid \text{bright})}$$
$$= \frac{0.2 \cdot 0.67}{0.2 \cdot 0.67 + 0.9 \cdot 0.33} = 0.31$$

Dr. Jürgen Sturm, Computer Vision Group, TUM
Visual Navigation for Flying Robots
44

The second observation lowers the probability that the robot is above the landing zone!

Example: Take-Off
- Action: $u \in \{\text{takeoff}\}$
- World state: $x \in \{\text{ground, air}\}$

Action Models
- To incorporate the outcome of an action $u$ into the current state estimate (“belief”), we use the conditional pdf $p(x' \mid u, x)$

This term specifies the probability that executing the action $u$ in state $x$ will lead to state $x'$

Typical Actions
- Quadrocopter accelerates by changing the speed of its motors
- Position also changes when quadrocopter does “nothing” (and drifts..)
- Actions are never carried out with absolute certainty
- In contrast to measurements, actions generally increase the uncertainty of the state estimate
Integrating the Outcome of Actions

- Discrete case
  \[ P(x' \mid u) = \sum_x P(x' \mid u, x)P(x) \]

- Continuous case
  \[ p(x' \mid u) = \int p(x' \mid u, x)p(x)dx \]

Example: Take-Off

- Prior belief on robot state: \( P(x = \text{ground}) = 1.0 \) (robot is located on the ground)
- Robot executes “take-off” action
- What is the robot’s belief after one time step?
  \[ P(x' = \text{ground}) = \sum_x P(x' = \text{ground} \mid u, x)P(x) \]
  \[ = P(x' = \text{ground} \mid u, x = \text{ground})P(x = \text{ground}) \]
  \[ + P(x' = \text{ground} \mid u, x = \text{air})P(x = \text{air}) \]
  \[ = 0.1 \cdot 1.0 + 0.01 \cdot 0.0 = 0.1 \]
- Question: What is the probability at t=2?

Markov Assumption

- Observations depend only on current state
  \[ P(z_t \mid x_{0:t-1}, z_{t-1}, u_{1:t}) = P(z_t \mid x_t) \]
- Current state depends only on previous state and current action
  \[ P(x_t \mid x_{0:t-1}, z_{t-1}, u_{1:t}) = P(x_t \mid x_{t-1}, u_t) \]
- Underlying assumptions
  - Static world
  - Independent noise
  - Perfect model, no approximation errors

Bayes Filter

- Given:
  - Stream of observations \( z \) and actions \( u \):
    \[ d_t = (u_1, z_1, \ldots, u_t, z_t)^T \]
  - Sensor model \( P(z \mid x) \)
  - Action model \( P(x' \mid x, u) \)
  - Prior probability of the system state \( P(x) \)
- Wanted:
  - Estimate of the state \( x \) of the dynamic system
  - Posterior of the state is also called belief
    \[ Bel(x_t) = P(x_t \mid u_1, z_1, \ldots, u_t, z_t) \]
  1. Apply motion model
    \[ Bel(x_t) = \sum_{x_{t-1}} P(x_t \mid x_{t-1}, u_t)Bel(x_{t-1}) \]
  2. Apply sensor model
    \[ Bel(x_t) = \eta P(z_t \mid x_t)Bel(x_t) \]

Note: Bayes filters also work on continuous state spaces (replace sum by integral)
Example: Localization

- **Discrete state** \( x \in \{1, 2, \ldots, w\} \times \{1, 2, \ldots, h\} \)
- Belief distribution can be represented as a grid
- This is also called a **histogram filter**

\[
P(x_{ij}) = a_{ij}
\]

\[
P = 1.0
\]

\[
P = 0.0
\]

Example: Localization

- **Action** \( u \in \{\text{north, east, south, west}\} \)
- Robot can move one cell in each time step
- Actions are not perfectly executed
- Example: move east

\[
x_{t-1} = , u = \text{east} \Rightarrow
\]

60% success rate, 10% to stay/move too far/move one up/move one down

Example: Localization

- **Observation** \( z \in \{\text{marker, –marker}\} \)
- One (special) location has a marker
- Marker is sometimes also detected in neighboring cells

Example: Localization

- **t=0**
- Prior distribution (initial belief)
- Assume we know the initial location (if not, we could initialize with a uniform prior)
Example: Localization

- $t=1$, $u=\text{east}$, $z=\text{no-marker}$
- Bayes filter step 1: Apply motion model

Bayes filter step 2: Apply observation model

Example: Localization

- $t=2$, $u=\text{east}$, $z=\text{marker}$
- Bayes filter step 2: Apply motion model

Bayes filter step 1: Apply observation model

Bayes Filter - Summary

- Markov assumption allows efficient recursive Bayesian updates of the belief distribution
- Useful tool for estimating the state of a dynamic system
- Bayes filter is the basis of many other filters
  - Kalman filter
  - Particle filter
  - Hidden Markov models
  - Dynamic Bayesian networks
  - Partially observable Markov decision processes (POMDPs)

Kalman Filter

- Bayes filter with continuous states
- State represented with a normal distribution
- Developed in the late 1950’s
- Kalman filter is very efficient (only requires a few matrix operations per time step)
- Applications range from economics, weather forecasting, satellite navigation to robotics and many more
- Most relevant Bayes filter variant in practice $\rightarrow$ exercise sheet 2
Normal Distribution

- Univariate normal distribution
  \[
  X \sim \mathcal{N}(\mu, \sigma) \\
p(X = x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)
  \]

- Multivariate normal distribution
  \[
  X \sim \mathcal{N}(\mu, \Sigma) \\
p(x) = \mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right)
  \]

Properties of Normal Distributions

- Linear transformation \(\rightarrow\) remains Gaussian
  \[
  X \sim \mathcal{N}(\mu, \Sigma), Y \sim AX + B \\
  \Rightarrow Y \sim \mathcal{N}(A\mu + B, A\Sigma A^T)
  \]

- Intersection of two Gaussians \(\rightarrow\) remains Gaussian
  \[
  X_1 \sim \mathcal{N}(\mu_1, \Sigma_1), X_2 \sim \mathcal{N}(\mu_2, \Sigma_2) \\
  \Rightarrow p(X_1, X_2) = \mathcal{N}\left(\frac{\Sigma_1 \mu_1 + \Sigma_2 \mu_2}{\Sigma_1 + \Sigma_2}, \frac{1}{\Sigma_1 + \Sigma_2}\right)
  \]

Linear Process Model

- Consider a time-discrete stochastic process (Markov chain)

\[
\begin{align*}
&\ldots \quad u_{t-1} \quad u_t \quad u_{t+1} \\
&\quad \downarrow \quad \downarrow \quad \downarrow \\
&x_{t-1} \quad x_t \quad x_{t+1} \\
&\quad \downarrow \quad \downarrow \quad \downarrow \\
&z_{t-1} \quad z_t \quad z_{t+1} \\
&\quad \downarrow \quad \downarrow \quad \downarrow \\
&\ldots
\end{align*}
\]

- Consider a time-discrete stochastic process
- Represent the estimated state (belief) by a Gaussian
  \[
  x_t \sim \mathcal{N}(\mu_t, \Sigma_t)
  \]
- Assume that the system evolves linearly over time, then
  \[
x_t = Ax_{t-1}
  \]
**Linear Process Model**

- Consider a time-discrete stochastic process
- Represent the estimated state (belief) by a Gaussian $x_t \sim \mathcal{N}(\mu_t, \Sigma_t)$
- Assume that the system evolves linearly over time and depends linearly on the controls

$$x_t = Ax_{t-1} + Bu_t$$

**Linear Observations**

- Further, assume we make observations that depend linearly on the state

$$z_t = Cx_t$$

**Kalman Filter**

Estimates the state $x_t$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_t = Ax_{t-1} + Bu_t + \epsilon_t$$

and (linear) measurements of the state

$$z_t = Cx_t + \delta_t$$

with $\delta_t \sim \mathcal{N}(0, R)$ and $\epsilon_t \sim \mathcal{N}(0, Q)$

**Variables and Dimensions**

- **State** $x \in \mathbb{R}^n$
- **Controls** $u \in \mathbb{R}^l$
- **Observations** $z \in \mathbb{R}^k$
- **Process equation**

$$x_t = A x_{t-1} + B u_t + \epsilon_t$$

- **Measurement equation**

$$z_t = C x_t + \delta_t$$
Kalman Filter

- Initial belief is Gaussian
  \[ \text{Bel}(x_0) = \mathcal{N}(x_0; \mu_0, \Sigma_0) \]
- Next state is also Gaussian (linear transformation)
  \[ x_t \sim \mathcal{N}(Ax_{t-1} + Bu_t, Q) \]
- Observations are also Gaussian
  \[ z_t \sim \mathcal{N}(Cx_t, R) \]

From Bayes Filter to Kalman Filter

For each time step, do
1. Apply motion model
   \[
   \overline{\text{Bel}}(x_t) = \int p(x_t \mid x_{t-1}, u_t) \text{Bel}(x_{t-1}) \, dx_{t-1} \\
   = \mathcal{N}(x_t; A\mu_{t-1} + Bu_t, A\Sigma A^T + Q) \\
   = \mathcal{N}(x_t; \bar{\mu}_t, \bar{\Sigma}_t)
   \]

2. Apply sensor model
   \[
   \mu_t = \bar{\mu}_t + K_t(z_t - C\bar{\mu}_t) \\
   \Sigma_t = (I - K_tC)\bar{\Sigma}_t \\
   \text{with } K_t = \bar{\Sigma}_tC^T(\Sigma_iC^T + R)^{-1}
   \]

Kalman Filter

For the interested readers:
See Probabilistic Robotics for full derivation (Chapter 3)
Nonlinear Dynamical Systems

- Most realistic robotic problems involve nonlinear functions
- Motion function
  \[ x_t = g(u_t, x_{t-1}) \]
- Observation function
  \[ z_t = h(x_t) \]

Taylor Expansion

- Solution: Linearize both functions
- Motion function
  \[ g(x_{t-1}, u_t) \approx g(u_{t-1}, u_t) + \frac{\partial g(u_{t-1}, u_t)}{\partial x_{t-1}}(x_{t-1} - u_{t-1}) \]
  \[ = g(u_{t-1}, u_t) + G_t(x_{t-1} - u_{t-1}) \]
- Observation function
  \[ h(x_t) \approx h(\hat{u}_t) + \frac{\partial h(\hat{u}_t)}{\partial x_t}(x_t - \hat{u}_t) \]
  \[ = h(\hat{u}_t) + H_t(x_t - \hat{u}_t) \]

Extended Kalman Filter

For each time step, do
1. Apply motion model
   \[ \mu_t = g(\mu_{t-1}, u_t) \]
   \[ \Sigma_t = G_t \Sigma_{t-1} G_t^T + Q \]
   \[ G_t = \frac{\partial g(\mu_{t-1}, u_t)}{\partial x_{t-1}} \]
2. Apply sensor model
   \[ \mu_t = \hat{\mu}_t + K_t(z_t - h(\hat{\mu}_t)) \]
   \[ \Sigma_t = (I - K_t H_t) \Sigma_t \]
   \[ K_t = \Sigma_t H_t^T (H_t \Sigma_t H_t^T + R)^{-1} \]
   with \( H_t = \frac{\partial h(\hat{\mu}_t)}{\partial x_t} \)

Example

Motion Function and its derivative
\[ g(x, u) = \begin{pmatrix} x + (\cos(\psi) \dot{x} - \sin(\psi) \dot{y}) \Delta t \\ y + (\sin(\psi) \dot{x} + \cos(\psi) \dot{y}) \Delta t \\ \psi + \dot{\psi} \Delta t \end{pmatrix} \]
\[ G = \frac{\partial g(x, u)}{\partial x} = \begin{pmatrix} 1 & 0 & (-\sin(\psi) \dot{x} - \cos(\psi) \dot{y}) \Delta t \\ 0 & 1 & (\cos(\psi) \dot{x} + \sin(\psi) \dot{y}) \Delta t \\ 0 & 0 & 1 \end{pmatrix} \]

Example

Observation Function (\( \rightarrow \) Sheet 2)
\[ h(x) = \ldots \]
\[ H = \frac{\partial h(x)}{\partial x} = \ldots \]
Example

- Dead reckoning (no observations)
- Large process noise in x+y

Example

- Now with observations (limited visibility)
- Assume robot knows correct starting pose

Example

- What if the initial pose (x+y) is wrong?

Example

- What if the initial pose (x+y+yaw) is wrong?

Example

- If we are aware of a bad initial guess, we set the initial sigma to a large value (large uncertainty)
Summary

- Observations and actions are inherently noisy
- Knowledge about state is inherently uncertain
- Probability theory
- Probabilistic sensor and motion models
- Bayes Filter, Histogram Filter, Kalman Filter, Examples
**Organization - Exam**

- Oral exams *in teams* (2-3 students)
- At least 15 minutes per student → individual grades
- Questions will address
  - Material from the lecture
  - Material from the exercise sheets
  - Your mini-project

**DC Motors**

- Maybe you built one in school
- Stationary permanent magnet
- Electromagnet induces torque
- Split ring switches direction of current

**Brushless Motors**

- Used in most quadrocopters
- Permanent magnets on the axis
- Electromagnets on the outside
- Requires motor controller to switch currents → Does not require brushes (less maintenance)
### Pulse Width Modulation (PWM)

- Protocol used to control motor speed
- Remote controls typically output PWM

![PWM Diagram](image)

### I2C Protocol

- Serial data line (SDA) + serial clock line (SCL)
- All devices connected in parallel
- 7-10 bit address, 100-3400 kbit/s speed
- Used by Mikrocopter for motor control

![I2C Diagram](image)

### Control Architecture

![Control Architecture Diagram](image)

### Kinematics and Dynamics

- **Kinematics**
  - Integrate acceleration to get velocity
  - Integrate velocity to get position
- **Dynamics**
  - Actuators induce forces and torques
  - Forces induce linear acceleration
  - Torques induce angular acceleration
- What types of forces do you know?
- What types of torques do you know?

### Example: 1D Kinematics

- **State** \( \mathbf{x} = (x \ x' \ x'')^T \in \mathbb{R}^3 \)
- **Action** \( u \in \mathbb{R} \)
- **Process model**
  \[
  \mathbf{x}_t = \begin{pmatrix}
  1 & \Delta t & 0 \\
  0 & 1 & \Delta t \\
  0 & 0 & 1 \\
  \end{pmatrix} \mathbf{x}_{t-1} + \begin{pmatrix}
  0 \\
  0 \\
  1 \\
  \end{pmatrix} u_t
  \]
- **Kalman filter**
- **How many states do we need for 3D?**

### Dynamics - Essential Equations

- **Force (Kraft)**
  \[ m\ddot{x} = \sum_i F_i \]
- **Torque (Drehmoment)**
  \[ J\ddot{\alpha} = \sum_i \tau_i \]
Forces

- Gravity \( F_{\text{grav}} = mg \)
- Friction
  - Stiction (static friction) \( F_{\text{stiction}} = c_s \text{sign} \dot{x} \)
  - Damping (viscous friction) \( F_{\text{damping}} = D \dot{x} \)
- Spring \( F_{\text{spring}} = K(x - x_{eq}) \)
- Magnetic force
- …

Example: Spring-Damper System

- Combination of spring and damper
- Forces \( F = F_{\text{damping}} + F_{\text{spring}} \)
- Resulting dynamics \( m \ddot{x} = D \dot{x} + K(x - x_{eq}) \)

Torques

- Definition \( \tau = F \times r \)
- Torques sum up \( \tau_{\text{net}} = \sum \tau_i \)
- Torque results in angular acceleration \( \tau = J \alpha \) (with \( \alpha = \frac{\text{d} \omega}{\text{d} t} \), \( J \) moment of inertia)
- Friction same as before...

Dynamics of a Quadrocopter

- Each propeller induces force and torque by accelerating air
- Gravity pulls quadrocopter downwards

Vertical Acceleration

- Thrust \( F_{\text{thrust}} = F_1 + F_2 + F_3 + F_4 \)

Vertical and Horizontal Acceleration

- Thrust \( F_{\text{thrust}} = F_1 + F_2 + F_3 + F_4 \)
Vertical and Horizontal Acceleration

- **Thrust** \( F_{\text{thrust}} = F_1 + F_2 + F_3 + F_4 \)
- **Acceleration** \( \ddot{x}_{\text{global}} = (R_{RPY} F_{\text{thrust}} - F_{\text{grav}})/m \)

### Yaw

- Each propeller induces torque due to rotation and the interaction with the air
- Induced torque \( \tau = \tau_1 - \tau_2 + \tau_3 - \tau_4 \)
- Induced angular acceleration

### Assumptions of Cascaded Control

- Dynamics of inner loops is so fast that it is not visible from outer loops
- Dynamics of outer loops is so slow that it appears as static to the inner loops

---

**Pitch (and Roll)**

- Attitude changes when opposite motors generate unequal thrust
- Induced torque \( \tau = (F_1 - F_3) \times r \)
- Induced angular acceleration

---

**Cascaded Control Example**

- Motor control happens on motor boards (controls every motor tick)
- Attitude control implemented on microcontroller with hard real-time (at 1000 Hz)
- Position control (at 10 – 250 Hz)
- Trajectory (waypoint) control (at 0.1 – 1 Hz)
Feedback Control - Generic Idea

- Desired value 35°

Feedback Control - Example

- Controller: $u_t = K(x_{des} - z)$
- Plant: $x_t = x_{t-1} + u_t + \epsilon_t$
- Measurement: $z_t = x_t + \delta_t$

Measurement Noise

- What effect has noise in the measurements?
- Poor performance for K=1
- How can we fix this?
Proper Control with Measurement Noise

- Lower the gain... (K=0.15)

What do High Gains do?

- High gains are always problematic (K=2.15)

What happens if sign is messed up?

- Check K=-0.5

Saturation

- In practice, often the set of admissible controls \( u \) is bounded
- This is called (control) saturation

Delays

- In practice most systems have delays
- Can lead to overshoots/oscillations/de-stabilization

One solution: lower gains (why is this bad?)
Dr. Jürgen Sturm, Computer Vision Group, TUM

**Delays**

- What is the total dead time of this system?

![Diagram of Delays]

- Can we distinguish delays in the measurement from delays in actuation? No!

**Smith Predictor**

- Plant model is available
- 5 seconds delay
- Results in perfect compensation
- Why is this unrealistic in practice?

![Diagram of Smith Predictor]

- Time delay (and plant model) is often not known accurately (or changes over time)
- What happens if time delay is overestimated?
### Smith Predictor

- Time delay (and plant model) is often not known accurately (or changes over time)
- What happens if time delay is underestimated?

### Rigid Body Kinematics

- Consider a rigid body
- Free floating in 1D space, no gravity
- How does this system evolve over time?

### Position Control

- Consider a rigid body
- Free floating in 1D space, no gravity
- How does this system evolve over time?
- Example: \( x_0 = 0, \dot{x}_0 = 0 \)

- In each time instant, we can apply a force \( F \)
- Results in acceleration \( \ddot{x} = F/m \)
- Desired position \( x_{des} = 1 \)
**P Control**

- What happens for this control law?
  \[ u_t = K(x_{\text{des}} - x_{t-1}) \]
- This is called proportional control

**PD Control**

- What happens for this control law?
  \[ u_t = K_P(x_{\text{des}} - x_{t-1}) + K_D(\dot{x}_{\text{des}} - \dot{x}_{t-1}) \]
- Proportional-Derivative control

**PD Control**

- What happens when we add gravity?

**PD Control**

- What happens for this control law?
  \[ u_t = K_P(x_{\text{des}} - x_{t-1}) + K_D(\dot{x}_{\text{des}} - \dot{x}_{t-1}) \]
- What if we set higher gains?

**PD Control**

- What happens for this control law?
  \[ u_t = K_P(x_{\text{des}} - x_{t-1}) + K_D(\dot{x}_{\text{des}} - \dot{x}_{t-1}) \]
- What if we set lower gains?
Gravity compensation

- Add as an additional term in the control law
  \[ u_t = K_P(x_{\text{des}} - x_{t-1}) + K_D(\dot{x}_{\text{des}} - \dot{x}_{t-1}) + F_{\text{grav}} \]
- Any known (inverse) dynamics can be included

PD Control

- What happens when we have systematic errors? (noise with non-zero mean)
- Example: unbalanced quadrocopter, wind, ...
- Does the robot ever reach its desired location?

PID Control

- Idea: Estimate the system error (bias) by integrating the error
  \[ u_t = K_P(x_{\text{des}} - x_t) + K_D(\dot{x}_{\text{des}} - \dot{x}_t) + K_I \int x_{\text{des}} - x_t dt \]
- Proportional+Derivative+Integral Control

Example: Wind-up effect

- Quadrocopter gets stuck in a tree → does not reach steady state
- What is the effect on the I-term?

De-coupled Control

- So far, we considered only single-input, single-output systems (SISO)
- Real systems have multiple inputs + outputs
- MIMO (multiple-input, multiple-output)
- In practice, control is often de-coupled
How to Choose the Coefficients?

- Gains too large: overshooting, oscillations
- Gains too small: long time to converge
- Heuristic methods exist
- In practice, often tuned manually

Example: Ardrone

Cascaded control
- Inner loop runs on embedded PC and stabilizes flight
- Outer loop runs externally and implements position control

Ardrone: Inner Control Loop

- Plant input: motor torques
  \[ \mathbf{u}_{\text{inner}} = (\tau_1, \tau_2, \tau_3, \tau_4)^T \]
- Plant output: roll, pitch, yaw rate, z velocity
  \[ \mathbf{x}_{\text{inner}} = (\omega_x, \omega_y, \omega_z, \dot{z})^T \]

  - attitude (measured using gyro + accelerometer)
  - z velocity (measured using ultrasonic distance sensor + attitude)

Ardrone: Outer Control Loop

- Outer loop sees inner loop as a plant (black box)
- Plant input: roll, pitch, yaw rate, z velocity
  \[ \mathbf{u}_{\text{outer}} = (\omega_x, \omega_y, \omega_z, \dot{z})^T \]
- Plant output:
  \[ \mathbf{x}_{\text{outer}} = (x, y, z, \psi)^T \]

Mechanical Equivalent

- PD Control is equivalent to adding spring-dampers between the desired values and the current position

PID Control – Summary

- PID is the most used control technique in practice
- P control \( \rightarrow \) simple proportional control, often enough
- PI control \( \rightarrow \) can compensate for bias (e.g., wind)
- PD control \( \rightarrow \) can be used to reduce overshoot (e.g., when acceleration is controlled)
- PID control \( \rightarrow \) all of the above
Optimal Control

What other control techniques do exist?
- Linear-quadratic regulator (LQR)
- Reinforcement learning
- Inverse reinforcement learning
- ... and many more

Linear Quadratic Regulator

Given:
- Discrete-time linear system
  \[ x_{k+1} = Ax_k + Bu_k \]
- Quadratic cost function
  \[ J = \sum_{k=0}^{\infty} (x_k^T Q x_k + u_k^T R u_k) \]

Goal: Find the controller with the lowest cost → LQR control

Reinforcement Learning

- In principle, any measure can be used
- Define reward for each state-action pair
  \[ R(x_t, u_t) \]
  - Find the policy (controller) that maximizes the expected future reward
  - Compute the expected future reward based on
    - Known process model
    - Learned process model (from demonstrations)

Inverse Reinforcement Learning

- Parameterized reward function
- Learn these parameters from expert demonstrations and refine
- Example: [Abbeel and Ng, ICML 2010]

Interesting Papers at ICRA 2012

- Flying robots are a hot topic in the robotics community
- 4 (out of 27) sessions on flying robots, 4 sessions on localization and mapping
- Robots: quadrocopters, nano quadrocopters, fixed-wing airplanes
- Sensors: monocular cameras, Kinect, motion capture, laser-scanners
**Autonomous Indoor 3D Exploration with a Micro-Aerial Vehicle**  
Shaojie Shen, Nathan Michael, and Vijay Kumar

- Map a previously unknown building
- Find good exploration frontiers in partial map

**Decentralized Formation Control with Variable Shapes for Aerial Robots**  
Matthew Turpin, Nathan Michael, and Vijay Kumar

- Move in formation (e.g., to traverse a window)
- Avoid collisions
- Dynamic role switching

**Versatile Distributed Pose Estimation and Sensor Self-Calibration for an Autonomous MAV**  
Stephan Weiss, Markus W. Achtelik, Margarita Chi, Roland Siegwart

- IMU, camera
- EKF for pose, velocity, sensor bias, scale, inter-sensor calibration

**On-board Velocity Estimation and Closed-loop Control of a Quadrotor UAV based on Optical Flow**  
Volker Grabe, Heinrich H. Bülthoff, and Paolo Robuffo Giordano

- Ego-motion from optical flow using homography constraint
- Use for velocity control

**Autonomous Landing of a VTOL UAV on a Moving Platform Using Image-based Visual Servoing**  
Daewon Lee, Tyler Ryan and H. Jin. Kim

- Tracking and landing on a moving platform
- Switch between tracking and landing behavior

**Resonant Wireless Power Transfer to Ground Sensors from a UAV**  
Brent Griffin and Carrick Detweiler

- Quadrocopter transfers power to light a LED
Using Depth in Visual Simultaneous Localisation and Mapping
Sebastian A. Scherer, Daniel Dube and Andreas Zell

- Combine PTAM with Kinect
- Monocular SLAM: scale drift
- Kinect: has small maximum range

ICRA Papers

- Will put them in our paper repository
- Remember password (or ask by mail)
- See course website
Organization: Exam

- Registration deadline: June 30
- Course ends: July 19
- Examination dates: t.b.a. (mid August)
  - Oral team exam
  - Sign up for a time slot starting from Mid July
  - List will be placed on blackboard in front of our secretary

Motivation

- Quick geometry recap
- Image filters
- 2D image alignment
- Corner detectors
- Kanade-Lucas-Tomasi tracker
- 3D motion estimation

Angular and linear velocities

- Linear velocity \( \mathbf{v} = (v_x, v_y, v_z)^T \in \mathbb{R}^3 \)
- Angular velocity \( \omega = (\omega_x, \omega_y, \omega_z)^T \in \mathbb{R}^3 \)
- Linear and angular velocity together form a twist \( \xi = (v^T, \omega^T)^T \)

Angular and linear velocities

- Linear velocity \( \mathbf{v} = (v_x, v_y, v_z)^T \in \mathbb{R}^3 \)
- Angular velocity \( \omega = (\omega_x, \omega_y, \omega_z)^T \in \mathbb{R}^3 \)
- Now consider a 3D point \( \mathbf{p} \in \mathbb{R}^3 \) of a rigid body moving with twist \( \xi = (v^T, \omega^T)^T \)
Angular and linear velocities

- Linear velocity $\mathbf{v} = (v_x, v_y, v_z)^T \in \mathbb{R}^3$
- Angular velocity $\mathbf{\omega} = (\omega_x, \omega_y, \omega_z)^T \in \mathbb{R}^3$
- Now consider a 3D point $\mathbf{p} \in \mathbb{R}^3$ of a rigid body moving with twist $\mathbf{\xi} = (\mathbf{v}^T, \mathbf{\omega}^T)^T$
- What is the velocity $\dot{\mathbf{p}}$ at point $\mathbf{p}$?

\[
\mathbf{p}(t) = R(t)\mathbf{p}(0) + t(t) = \exp([\mathbf{\omega}]_x t)\mathbf{p}(0) + \mathbf{v} t
\]

\[
\dot{\mathbf{p}}(t) = [\mathbf{\omega}]_x \mathbf{p}(0) + \mathbf{v}
\]

Recap: Perspective Projection

- 3D point $\mathbf{p}$ (in the camera frame)
- 2D point $\mathbf{x}$ (on the image plane)
- Pin-hole camera model

\[
\mathbf{x} = \lambda \mathbf{\bar{x}} = \mathbf{p}
\]

- Remember, $\mathbf{\bar{x}}$ is homogeneous, need to normalize

\[
\mathbf{\bar{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} \Rightarrow \mathbf{x} = \begin{pmatrix} \tilde{x} / \tilde{z} \\ \tilde{y} / \tilde{z} \end{pmatrix}
\]
Camera Intrinsics

- So far, 2D point is given in meters on image plane.
- But: we want 2D point to be measured in pixels (as the sensor does).

Image Plane

- Pixel coordinates $\mathbf{x} \in \Omega$
- Image plane $\Omega \subset \mathbb{R}^2$

Example:
- Discrete case $\mathbf{x} \in [0, W) \times [0, H) \subset \mathbb{N}_0^2$ (default in this course)
- Continuous case $\mathbf{x} \in [0, 1] \times [0, 1] \subset \mathbb{R}^2$

Image Functions

- Realistically, the image function is only defined on a rectangle and has finite range $f : [0, W - 1] \times [0, H - 1] \mapsto [0, 1]$
- Image can be represented as a matrix
- Alternative notations
- $F_{ij}$, $f(i, j)$, $f(x, y)$, $f(\mathbf{x})$, $f(\mathbf{x})$...
Light intensity is sampled by CCD/CMOS sensor on a regular grid.

Electric charge of each cell is quantized and gamma compressed (for historical reasons)

\[ V = B^\gamma \text{ with } \gamma = 2.2 \]

CRTs / monitors do the inverse \( B = V^\gamma \)

Almost all images are gamma compressed → Double brightness results only in a 37% higher intensity value (!)

Most CMOS sensors have a rolling shutter

Rows are read out sequentially

Sensitive to camera and object motion

Can we correct for this?

We want to remove unwanted sources of variation, and keep the information relevant for whatever task we need to solve

\[ f(i, j) \rightarrow g(i, j) \]

Example tasks: de-noising, (de-)blurring, computing derivatives, edge detection, ...

We are often interested in spatially invariant operations

\[ g(i, j) = f * h = \sum_{k,l} h(i-k, j-l) f(k, l) \]

Example

\[ \begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \ast \begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array} = ? \]
**Spatially Invariant Filtering**

- We are often interested in spatially invariant operations
  \[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l)f(k, l) \]
- Example

**Important Filters**

- Impulses
- Shifts
- Blur
  - Gaussian
  - Bilateral filter
  - Motion blur
- Edges
  - Finite difference filter
  - Derivative filter
  - Oriented filters
  - Gabor filter
-...

**Impulse**

\[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l)f(k, l) \]

**Image shift (translation)**

\[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l)f(k, l) \]

**Image rotation**

\[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l)f(k, l) \]

Image rotation is a linear operator (why?), but not a spatially invariant operation (why?). There is no convolution.
Rectangular Filter

\[ g(i, j) = f * h = \sum_{k,l} h(i-k,j-l)f(k,l) \]

Gaussian Blur

- Gaussian distribution
  \[ g_\sigma(i, j) = \frac{1}{2\pi\sigma^2} \exp \left( -\frac{r^2 + j^2}{2\sigma^2} \right) \]
- Example of resulting kernel

Image Gradient

- The image gradient \( \nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^T \) points in the direction of increasing intensity (steepest ascend)
Image Gradient

- The image gradient $\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^T$ points in the direction of increasing intensity (steepest ascend)

$$\nabla f = \left( \frac{\partial f}{\partial x}, 0 \right)^T \quad \nabla f = \left( 0, \frac{\partial f}{\partial y} \right)^T \quad \nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^T$$

- Gradient direction (related to edge orientation)
  $$\theta = \arctan \left( \frac{\partial f}{\partial y}, \frac{\partial f}{\partial x} \right)$$

- Gradient magnitude (edge strength)
  $$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

How can we differentiate a digital image $f(x,y)$?
- Option 1: Reconstruct a continuous image, then take gradient
- Option 2: Take discrete derivative (finite difference filter)
- Option 3: Convolve with derived Gaussian (derivative filter)

Finite difference

- First-order central difference
  $$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x-1,y)}{2}$$
  Corresponding convolution kernel:

- Second-order Derivative
  $$\frac{\partial^2 f(x)}{\partial x^2} \approx f(x+1) - 2f(x) + f(x-1)$$
  Corresponding convolution kernel:
(Dense) Motion Estimation

- 2D motion
- 3D motion

Problem Statement

- **Given:** two camera images $f_0, f_1$
- **Goal:** estimate the camera motion $u$

For the moment, let’s assume that the camera only moves in the $xy$-plane, i.e., $u = (u \ v)^T$

Extension to 3D follows

Error Metrics for Image Comparison

- Sum of Squared Differences (SSD)
  \[
  E_{SSD}(u) = \sum_i (f_1(x_i + u) - f_0(x_i))^2 = \sum_i e_i^2
  \]
  with displacement $u = (u \ v)^T$
  and residual errors $e_i = f_1(x_i + u) - f_0(x_i)$
Robust Error Metrics

- SSD metric is sensitive to outliers
- Solution: apply a (more) robust error metric

\[ E_{\text{SSD}}(\mathbf{u}) = \sum_i \rho(f_1(x_i + \mathbf{u}) - f_0(x_i)) = \sum_i \rho(e_i) \]

Robust Error Metrics

- Sum of Absolute Differences
  \[ \rho_{\text{SAD}}(e) = |e| \]
- Sum of truncated errors
  \[ \rho_{\text{trunc}}(e) = \begin{cases} e^2 & \text{if } |e| < b \\ b^2 & \text{otherwise} \end{cases} \]
- Geman-McClure function (Huber norm)
  \[ \rho_{\text{Huber}}(e) = \frac{e^2}{1 + b^2} \]

Windowed SSD

- Images (and image patches) have finite size
- Standard SSD has a bias towards smaller overlaps (less error terms)
- Solution: divide by the overlap area
- Root mean square error

\[ E_{\text{RMS}}(\mathbf{u}) = \sqrt{E_{\text{SSD}} / A} \]

Exposure Differences

- Images might be taken with different exposure (auto shutter, white balance, ...)
- Bias and gain model
  \[ f_1(x + \mathbf{u}) = (1 + \alpha)f_0(x) + \beta \]
- With SSD we get
  \[ E_{\text{BG}}(\mathbf{u}) = \sum_i (f_1(x_i + \mathbf{u}) - (1 + \alpha)f_0(x_i) + \beta)^2 \]
  \[ = \sum_i \alpha f_0(x) + \beta - e_i^2 \]

Cross-Correlation

- Maximize the product (instead of minimizing the differences)
  \[ E_{\text{CC}}(\mathbf{u}) = - \sum_i f_0(x_i)f_1(x_i + \mathbf{u}) \]
- Normalized cross-correlation (between -1..1)
  \[ E_{\text{NCC}}(\mathbf{u}) = \frac{\sum_i (f_0(x_i) - \text{mean}_0)(f_1(x_i + \mathbf{u}) - \text{mean}_1)}{\sqrt{\text{var}_0\text{var}_1}} \]
General Idea

1. Define an error metric $E(u)$ that defines how well the two images match given a motion vector.
2. Find the motion vector with the lowest error:
   $$ u^* = \arg \min_u E(u) $$

Finding the minimum

- Full search (e.g., ±16 pixels)
- Gradient descent
- Hierarchical motion estimation

Hierarchical motion estimation

- Construct image pyramid
- Estimate motion on coarse level
- Use as initialization for next finer level

Gradient Descent

- Perform gradient descent on the SSD energy function (Lucas and Kanade, 1981)
- Taylor expansion of energy function:
  $$ E_{LKS-SSD}(u + \Delta u) = \sum_i (f_i(x_i + u + \Delta u) - f_0(x_i))^2 $$
  $$ \approx \sum_i (f_i(x_i + u) + J_1(x + u)\Delta u - f_0(x_i))^2 $$
  $$(a + b\Delta u)^2 \approx a^2 + 2ab + b^2 \Delta u$$
  $$ = \sum_i (J_1(x + u)\Delta u + e_i)^2 $$
  with $J_1(x + u) = \nabla f_1(x + u) = \left( \frac{\partial f_1}{\partial x}, \frac{\partial f_1}{\partial y} \right)(x_i + u)$

Least Squares Problem

- Goal: Minimize
  $$ E(u + \Delta u) = \sum_i (J_1(x_i + u)\Delta u + e_i)^2 $$
- Solution: Compute derivative (and set to zero)
  $$ \frac{\partial E(u + \Delta u)}{\partial \Delta u} = 2A\Delta u + 2b $$
  with $A = \sum_i J_1^T(x_i + u)J_1(x + u)$
  and $b = \sum_i e_iJ_1^T(x_i + u)$

Least Squares Problem

1. Compute $A,b$ from image gradients using
   $$ A = \left( \sum f_x^2 \sum f_x f_y \right) \quad b = \left( \sum f_x f_x \sum f_y f_y \right) $$
   with $f_x = \frac{\partial f_1(x)}{\partial x}, f_y = \frac{\partial f_1(x)}{\partial y}$
   and $f_t = \frac{\partial f_1(x)}{\partial t} \approx f_1(x) - f_0(x)$

2. Solve $A\Delta u = -b$
   $$ \Rightarrow \Delta u = -A^{-1}b $$

All of these computations are super fast!
Covariance of the Estimated Motion

- Assuming (small) Gaussian noise in the images
  \[ f_{\text{obs}}(x_i) = f_{\text{true}}(x_i) + \epsilon_i \]
  with \( \epsilon_i \sim \mathcal{N}(0, \sigma^2) \)

- ... results in uncertainty in the motion estimate with covariance (e.g., useful for Kalman filter)
  \[ \Sigma_u = \sigma^2 A^{-1} \]

Optical Computer Mouse (since 1999)

- E.g., ADNS3080 from Agilent Technologies, 2005
  - 6400 fps
  - 30x30 pixels
  - 4 USD

Image Patches

- Sometimes we are interested in the motion of a small image patches
- **Problem**: some patches are easier to track than others
- What patches are easy/difficult to track?
- How can we recognize “good” patches?

Corner Detection

- Let’s look at the shape of the energy functional
- **Idea**: Inspect eigenvalues \( \lambda_1, \lambda_2 \) of Hessian \( A \)
  - \( \lambda_1, \lambda_2 \) small \( \rightarrow \) no point of interest
  - \( \lambda_1 \) large, \( \lambda_2 \) small \( \rightarrow \) edge
  - \( \lambda_1, \lambda_2 \) large \( \rightarrow \) corner
- Harris detector (does not need eigenvalues)
  \[ \lambda_1 \lambda_2 > \kappa (\lambda_1 + \lambda_2)^2 \leftrightarrow \det(A) > \kappa \text{trace}^2(A) \]
- Shi-Tomasi (or Kanade-Lucas)
  \[ \min(\lambda_1, \lambda_2) > \kappa \]
Corner Detection

1. For all pixels, compute corner strength
2. Non-maximal suppression (E.g., sort by strength, strong corner suppresses weaker corners in circle of radius $r$)

Other Detectors

- Förstner detector (localize corner with sub-pixel accuracy)
- FAST corners (learn decision tree, minimize number of tests $\rightarrow$ super fast)
- Difference of Gaussians / DoG (scale-invariant detector)
- ...

Kanade-Lucas-Tomasi (KLT) Tracker

- Algorithm
  1. Find (Shi-Tomasi) corners in first frame and initialize tracks
  2. Track from frame to frame
  3. Delete track if error exceeds threshold
  4. Initialize additional tracks when necessary
  5. Repeat step 2-4
- KLT tracker is highly efficient (real-time on CPU) but provides only sparse motion vectors
- Dense optical flow methods require GPU

3D Motion Estimation

(How) Can we recover the camera motion from the estimated flow field?

- Research paper: Grabe et al., ICRA 2012

Approach [Grabe et al., ICRA’12]

- Compute optical flow
- Estimate homography between images
- Extract angular and (scaled) linear velocity
- Additionally employ information from IMU
Assumptions

1. The quadrocopter moves slowly relative to the sampling rate → limited search radius

2. The environment is planar with normal \( N \) → image transformation is a homography

Apparent Velocity of a Point

- Stationary 3D point feature, given in camera frame
  \[ p \in \mathbb{R}^3 \]
- Moving camera with twist
  \[ \xi = (v^T, \omega^T)^T \in \mathbb{R}^6 \]
- Apparent velocity of the point in camera frame
  \[ \hat{p} = [\omega]_x p + v \]

Continuous Homography Matrix

- Assumption: All feature points are located on a plane
  \[ N^T p = d \]
  with plane normal \( N \in \mathbb{R}^3 \) and distance \( d \in \mathbb{R} \)

Continuous Homography Constraint

- The camera observes point \( p \in \mathbb{R}^3 \) at pixel \( x \in \mathbb{R}^2 \) (assuming \( K = f \) for simplicity)
  \[ \bar{x} = \lambda \bar{x} = p \]
- The KLT tracker estimates the motion \( u \) of the feature track in the image
- Constraint: \( u = \dot{x} \)
### Continuous Homography Constraint

- Combining these formulas gives us
  \[
  \hat{x} + \lambda \hat{u} = H p \\
  \lambda \hat{u} = H p - \hat{x} \\
  \hat{u} = H x - \frac{1}{\lambda} \hat{x}
  \]
- Multiply both sides with \([\hat{x}]_x\) gives us
  \[
  [\hat{x}]_x \hat{u} = [\hat{x}]_x H x - [\hat{x}]_x \frac{1}{\lambda} \hat{x} = 0
  \]
  \[
  \Rightarrow [\hat{x}]_x \hat{u} = [\hat{x}]_x H x
  \]

### Approach

- **Result:** For all observed motions in the image, the continuous homography constraint holds
  \[
  [\hat{x}]_x \hat{u} = [\hat{x}]_x H x
  \]
- How can we use this to estimate the camera motion?!

### Step 1: Estimate \(H\)

- Linear set of equations
  \[
  \begin{pmatrix}
  M_1^T \\
  M_2^T \\
  \vdots \\
  \end{pmatrix}
  h =
  \begin{pmatrix}
  [\hat{x}]_x \hat{u}_1^T \\
  [\hat{x}]_x \hat{u}_2^T \\
  \vdots \\
  \end{pmatrix}
  \]
- Solve for \(h\) using least squares
  \[
  Ah = b \\
  \Rightarrow h = (A^T A)^{-1} A^T b
  \]

### Step 2: Recover camera motion

Grabe et al. investigated three alternatives:

1. Recover \((\omega, \nu, d, N)\) from \(H = [\omega]_x + \nu \frac{1}{d} N^T\) using the 8-point algorithm (not yet explained)
2. Use angular velocity \(\omega\) from IMU to de-rotate observed feature tracks beforehand, then:

\[
H = \nu \frac{1}{d} N^T
\]
3. Additionally use gravity vector from IMU as plane normal \(N = N_{\text{IMU}}\), then

\[
\frac{\nu}{d} = H (N^T N)^{-1}
\]
Evaluation

- Comparison of estimated velocities with ground truth from motion capture system

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Norm error</th>
<th>Std. deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure vision</td>
<td>0.134</td>
<td>0.104</td>
</tr>
<tr>
<td>Ang. vel. known</td>
<td>0.117</td>
<td>0.089</td>
</tr>
<tr>
<td>Normal known</td>
<td>0.115</td>
<td>0.088</td>
</tr>
</tbody>
</table>

- Comparison of actual velocity with desired velocity (closed-loop control)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Norm error</th>
<th>Std. deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure vision</td>
<td>0.134</td>
<td>0.094</td>
</tr>
<tr>
<td>Ang. vel. known</td>
<td>0.117</td>
<td>0.093</td>
</tr>
<tr>
<td>Normal known</td>
<td>0.113</td>
<td>0.088</td>
</tr>
</tbody>
</table>

Visual Velocity Control

- All computations are carried out on-board (18fps)

Landing on a Moving Platform

- Similar approach, but with offboard computation

Commercial Solutions

- Helicommand 3D from Robbe
  2(?) cameras, IMU, air pressure sensor, 450 EUR
- Parrot Mainboard + Navigation board
  1 camera, IMU, ultrasound sensor, 210 USD

Lessons Learned Today

- How to estimate the translational motion from camera images
- Which image patches are easier to track than others
- How to estimate 3D motion from multiple feature tracks (and IMU data)

A Few Ideas for Your Mini-Project

- Person following (colored shirt or wearing a marker)
- Flying camera for taking group pictures (possibly using the OpenCV face detector)
- Fly through a hula hoop (brightly colored, white background)
- Navigate through a door (brightly colored)
- Navigate from one room to another (using ground markers)
- Avoid obstacles using optical flow
- Landing on a moving platform
- Your own idea here – be creative!
- ...
Joggobot

- Follows a person wearing a visual marker

[http://exertiongameslab.org/projects/joggobot]
### VISNAV Oral Team Exam

<table>
<thead>
<tr>
<th>Date and Time</th>
<th>Student Name</th>
<th>Student Name</th>
<th>Student Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tue, Aug. 9, 10am</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tue, Aug. 9, 11am</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tue, Aug. 9, 2pm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tue, Aug. 9, 3pm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tue, Aug. 9, 4pm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu, Aug. 14, 10am</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu, Aug. 14, 11am</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu, Aug. 14, 2pm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu, Aug. 14, 3pm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu, Aug. 14, 4pm</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### The SLAM Problem

**Given:**
- The robot’s controls \( u_1:t = \langle u_1, u_2, \ldots, u_t \rangle \)
- (Relative) observations \( z_{1:t} = \langle z_1, z_2, \ldots, z_t \rangle \)

**Wanted:**
- Map of features \( m = \langle m_1, m_2, \ldots, m_k \rangle \)
- Trajectory of the robot \( x_{1:t} = \langle x_1, x_2, \ldots, x_t \rangle \)

### SLAM Applications

SLAM is central to a range of indoor, outdoor, in-air and underwater applications for both unmanned and autonomous vehicles.

**Examples**
- At home: vacuum cleaner, lawn mower
- Air: inspection, transportation, surveillance
- Underwater: reef/environmental monitoring
- Underground: search and rescue
- Space: terrain mapping, navigation
SLAM with Ceiling Camera  
(Samsung Hauzen RE70V, 2008)

Localization, Path planning, Coverage  
(Neato XV11, $300)

Agenda for Today

- This week: focus on monocular vision
  - Feature detection, descriptors and matching
  - Epipolar geometry
  - Robust estimation (RANSAC)
  - Examples (PTAM, Photo Tourism)
- Next week: focus on optimization (bundle adjustment), stereo cameras, Kinect
- In two weeks: map representations, mapping and (dense) 3D reconstruction

SLAM with Laser + Line camera  
(Neato XV 11, 2010)

SLAM vs. SfM

- In Robotics: Simultaneous Localization and Mapping (SLAM)
  - Laser scanner, ultrasound, monocular/stereo camera
  - Typically in combination with an odometry sensor
  - Typically pre-calibrated sensors
- In Computer Vision: Structure from Motion (SfM), sometimes: Structure and Motion
  - Monocular/stereo camera
  - Sometimes uncalibrated sensors (e.g., Flick images)

How Do We Build a Panorama Map?

- We need to match (align) images
- Global methods sensitive to occlusion, lighting, parallax effects
- How would you do it by eye?
Matching with Features

- Detect features in both images

- Find corresponding pairs

- Use these pairs to align images

---

Problem 1:
We need to detect the same point independently in both images

→ We need a reliable detector

---

Problem 2:
For each point correctly recognize the corresponding one

→ We need a reliable and distinctive descriptor

---

Ideal Feature Detector

- Always finds the same point on an object, regardless of changes to the image

- Insensitive (invariant) to changes in:
  - Scale
  - Lightning
  - Perspective imaging
  - Partial occlusion

→ We need a reliable detector
Harris Detector

- Rotation invariance?

- Remember from last week

  \[ A = \left( \frac{\sum f_x^2}{\sum f_x f_y} \right) R = \frac{\lambda_1 \lambda_2 - \kappa (\lambda_1 + \lambda_2)^2}{\sum f_y^2} \]

  Ellipse rotates but its shape (i.e. eigenvalues) remains the same

  \( \rightarrow \) Corner response \( R \) is invariant to rotation

Harris Detector

- Partial invariance to additive and multiplicative intensity changes
  - Only derivatives are used \( \rightarrow \) invariance to intensity shift \( I \rightarrow I + b \)
  - Intensity scale \( I \rightarrow aI \): Because of fixed intensity threshold on local maxima, only partial invariance

Harris Detector

- Rotation invariance?

- Invariance to intensity change?

- Invariant to scaling?

\( R \) (image coordinate)
Harris Detector

- Not invariant to image scale

Difference Of Gaussians (DoG)

- Alternative corner detector that is additionally invariant to scale change

- Approach:
  - Run linear filter (diff. of two Gaussians, $\sigma_1 = 2\sigma_2$)
  - Do this at different scales
  - Search for a maximum both in space and scale

Example: Difference of Gaussians

SIFT Detector

Example

1. Input image 233x189 pixel
2. 832 candidates DoG minima/maxima (visualization indicate scale, orient., location)
3. 536 keypoints remain after thresholding on minimum contrast and principal curvature
Feature Matching

- Now, we know how to find repeatable corners
- Next question: How can we match them?

Template Convolution

Invariances
- Scaling: No
- Rotation: No (maybe rotate template?)
- Illumination: No (use bias/gain model?)
- Perspective projection: Not really

Scale Invariant Feature Transform (SIFT)

Approach
1. Find SIFT corners (position + scale)
2. Find dominant orientation and de-rotate patch
3. Extract SIFT descriptor (histograms over gradient directions)

Select Dominant Orientation

- Create a histogram of local gradient directions computed at selected scale (36 bins)
- Assign canonical orientation at peak of smoothed histogram
- Each key now specifies stable 2D coordinates (x, y, scale, orientation)
**SIFT Descriptor**
- Compute image gradients over 16x16 window (green), weight with Gaussian kernel (blue)
- Create 4x4 arrays of orientation histograms, each consisting of 8 bins
- In total, SIFT descriptor has 128 dimensions

**Feature Matching**
Given features in $I_p$, how to find best match in $I_q$?
- Define distance function that compares two features
- Test all the features in $I_p$, find the one with the minimal distance

**Feature Distance**
How to define the difference between features?
- Simple approach is Euclidean distance (or SSD)
  \[ d(d_1, d_2) = \|d_1 - d_2\| \]

**Feature Distance**
How to define the difference between features?
- Simple approach is Euclidean distance (or SSD)
  \[ d(d_1, d_2) = \|d_1 - d_2\| \]
- Problem: can give good scores to ambiguous (bad) matches

**Feature Distance**
How to define the difference between features?
- Better approach
  \[ d(d_1, d_2) = \|d_1 - d_2\|/\|d_1 - d_2'\| \]
  with $d_2'$ best matching feature from $I_2$
  $d_2'$ second best matching feature from $I_2$
- Gives small values for ambiguous matches

**Efficient Matching**
For feature matching, we need to answer a large number of nearest neighbor queries
- Exhaustive search $O(n^2)$
Efficient Matching
For feature matching, we need to answer a large number of nearest neighbor queries
- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found
- Approximate search
  - Locality sensitive hashing
  - Approximate nearest neighbor
Efficient Matching

For feature matching, we need to answer a large number of nearest neighbor queries
- Exhaustive search \( O(n^2) \)
- Indexing (k-d tree)
- Approximate search
  - Locality sensitive hashing
  - Approximate nearest neighbor

Other Descriptors

- SIFT (Scale Invariant Feature Transform) [Lowe, 2004]
- SURF (Speeded Up Robust Feature) [Bay et al., 2008]
- BRIEF (Binary robust independent elementary features) [Calonder et al., 2010]
- ORB (Oriented FAST and Rotated Brief) [Rublee et al, 2011]
- ...

Example: RGB-D SLAM [Engelhard et al., 2011; Endres et al. 2012]
- Feature descriptor: SURF
- Feature matching: FLANN (approximate nearest neighbor)

Structure From Motion (SfM)

- Now we can compute point correspondences
- What can we use them for?

Four Important SfM Problems

- Camera calibration
  Known 3D points, observe corresponding 2D points, compute camera pose
- Point triangulation
  Known camera poses, observe 2D point correspondences, compute 3D point
- Motion estimation (epipolar geometry)
  Observe 2D point correspondences, compute camera pose (up to scale)
- Bundle adjustment (next week!)
  Observe 2D point correspondences, compute camera pose and 3D points (up to scale)
Camera Calibration

- **Given:** \( n \) 2D/3D correspondences \( x_i \leftrightarrow p_i \)
- **Wanted:** \( M = KRt \) such that \( x_i = Mp_i \)

The algorithm has two parts:
1. Compute \( M \in \mathbb{R}^{3 \times 4} \)
2. Decompose \( M \) into \( K, R, t \) via QR decomposition

**Step 1: Estimate M**

- Re-arranged in matrix form
  \[
  \begin{pmatrix}
  X & Y & Z & 1 & 0 & 0 & 0 & -xX & -xY & -xZ & -x \\
  0 & 0 & 0 & X & Y & Z & 1 & -yX & -yY & -yZ & -y
  \end{pmatrix}
  m = 0
  \]

- With \( m = (m_{11}, m_{12}, \ldots, m_{34}) \in \mathbb{R}^{12} \)
- Concatenate equations for \( n \geq 6 \) correspondences
  \( Am = 0 \)
- Wanted vector \( m \) is in the null space of \( A \)
  
  Initial solution using SVD (vector with least singular value), refine using non-linear min.

**Step 2: Recover K,R,t**

- Remember \( M = KRt \)
- The first 3x3 submatrix is the product of an upper triangular and orthogonal (rot.) matrix

  \[
  K = \begin{pmatrix}
  f_x & s & c_x \\
  0 & f_y & c_y \\
  0 & 0 & 1
  \end{pmatrix}
  \]

**Procedure:**
1. Factor \( M \) into \( KR \) using QR decomposition
2. Compute translation as \( t = K^{-1}(p_{14}, p_{24}, p_{34})^T \)

**Example: ARToolkit Markers (1999)**

1. Threshold image
2. Detect edges and fit lines
3. Intersect lines to obtain corners
4. Estimate projection matrix \( M \)
5. Extract camera pose \( R, t \) (assume \( K \) is known)

The final error between measured and projected points is typically less than 0.02 pixels

**Triangulation**

- **Given:** cameras \( \{M_j = K_j(R_j t_j)\} \)
  point correspondence \( x_0, x_1 \)
- **Wanted:** Corresponding 3D point \( p \)
**Triangulation**

- Where do we expect to see \( p = (X \ Y \ Z \ W)^T \)?
- Minimize the residuals (e.g., using least squares)

\[
\hat{\mathbf{x}} = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}W}{m_{11}X + m_{12}Y + m_{13}Z + m_{14}W}, \quad \hat{\mathbf{y}} = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}W}{m_{11}X + m_{12}Y + m_{13}Z + m_{14}W}
\]

- Minimize the residuals (e.g., using least squares)

\[
p^* = \arg \min_p \sum_j d(x_j, \hat{x}_j)^2
\]

---

**Epipolar Geometry**

- Consider two cameras that observe a 3D world point
- The line connecting both camera centers is called the **baseline**
- Left line in left camera frame \( p_1 = d_1 \hat{x}_1 \)
- Right line in right camera frame \( p_2 = d_2 \hat{x}_2 \)
where \( \hat{x}_j = K^{-1}x_j \) are the (local) ray directions

\[
d_2\hat{x}_2 = Rd_1\hat{x}_1 + t \\
d_2[t]x_2 = d_1[t]x_1 + [t]t
\]

Intersection of both lines

\[
\begin{align*}
0 &= \hat{x}_1^T[R\hat{x}_1] \\
0 &= \hat{x}_2^T[R\hat{x}_1]
\end{align*}
\]

**Epipolar Geometry**

- Given the image of a point in one view, what can we say about its position in another?
- A point in one image “generates” a line in another image (called the **epipolar line**)

**Epipolar Geometry**

- Left line in **right** camera frame \( p_1' = Rd_1\hat{x}_1 + t \)
- Right line in right camera frame \( p_2' = d_2\hat{x}_2 \)
where \( \hat{x}_j = K^{-1}x_j \) are the (local) ray directions
- Intersection of both lines

\[
\begin{align*}
d_2\hat{x}_2 &= Rd_1\hat{x}_1 + t \\
d_2[t]x_2 &= d_1[t]x_1 + [t]t
\end{align*}
\]

\[
\begin{align*}
0 &= \hat{x}_1^T[R\hat{x}_1] \\
0 &= \hat{x}_2^T[R\hat{x}_1]
\end{align*}
\]

this is called the **epipolar constraint**
Epipolar Geometry

Note: The epipolar constraint holds for every pair of corresponding points \( x_1, x_2 \)

\[ \hat{x}_2^T E \hat{x}_1 = 0 \]

where \( E \) is called the essential matrix

\[ E = [t]_\times R \in \mathbb{R}^{3 \times 3} \]

8-Point Algorithm: General Idea

1. Estimate the essential matrix \( E \) from at least eight point correspondences
2. Recover the relative pose \( R, t \) from \( E \) (up to scale)

Step 1: Estimate \( E \)

- Epipolar constraint \( \hat{x}_2^T E \hat{x}_1 = 0 \)
- Written out (with \( x_j = (x_j, y_j, 1)^T \))
  \[ x_1 x_2 e_{11} + y_1 x_2 e_{12} + x_2 e_{13} + x_1 y_2 e_{21} + y_1 y_2 e_{22} + y_2 e_{23} + x_1 e_{31} + y_1 e_{32} + 1 e_{33} = 0 \]
- Stack the elements into two vectors
  \[ z = (x_1 x_2 y_1 y_2 \ldots 1)^T \quad e = (e_{11} e_{12} \ldots e_{33})^T \]
  \[ z^T e = 0 \]

Normalized 8-Point Algorithm

[Hartley 1997]

- Noise in the point observations is unequally distributed in the constraints, e.g.,
  - double noise
  \[ x_1 x_2 e_{11} + y_1 x_2 e_{12} + x_2 e_{13} + x_1 y_2 e_{21} + y_1 y_2 e_{22} + y_2 e_{23} + x_1 e_{31} + y_1 e_{32} + 1 e_{33} = 0 \]
  - normal noise
- Estimation is sensitive to scaling
- Normalize all points to have zero mean and unit variance

Step 2: Recover \( R, t \)

- Note: The absolute distance between the two cameras can never be recovered from pure images measurements alone!!!
- Illustration

- We can only recover the translation \( t \) up to scale
Step 2a: Recover $t$

- Remember: $E = [t]_x R$
- Therefore, $t^T$ is in the null space of $E$
  \[ t^T E = \underbrace{t^T [t]_x}_{=0} R = 0 \]
  \[
  E = [\hat{t}]_x R = U \Sigma V^T
  = \begin{pmatrix} u_0 & u_1 & t \end{pmatrix} \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} v_0^T & v_1^T & v_2^T \end{pmatrix}
  \]
  \[ S = U \Sigma V^T \]
  \[ R_{00}' U^T R = V^T \]
  \[ R = U R_{00}' V^T \]

→ Recover $\hat{t}$ (up to scale) using SVD

Step 2b: Recover $R$

Remember, the cross-product $[\hat{t}]_x$

- ... projects a vector onto a set of orthogonal basis vectors including $\hat{t}$
- ... zeros out the $\hat{t}$ component
- ... rotates the other two by 90°

\[
[\hat{t}]_x = S Z R_{00}' S^T
= \begin{pmatrix} s_0 & s_1 & \hat{t} \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ \hat{t} \end{pmatrix}
\]

Summary: 8-Point Algorithm

Given: Image pair

Find: Camera motion $R, t$ (up to scale)

- Compute correspondences
- Compute essential matrix
- Extract camera motion

How To Deal With Outliers?

Problem: No matter how good the feature descriptor/matcher is, there is always a chance for bad point correspondences (=outliers)

Robust Estimation

Example: Fit a line to 2D data containing outliers

There are two problems

1. Fit the line to the data $\arg \min_i \sum_i d_i^2$
2. Classify the data into inliers (valid points) and outliers (using some threshold)
RANdom SAmple Consensus (RANSAC)  
[Fischler and Bolles, 1981]

**Goal:** Robustly fit a model to a data set $S$ which contains outliers

**Algorithm:**
1. Randomly select a (minimal) subset of data points and instantiate the model from it
2. Using this model, classify the all data points as inliers or outliers
3. Repeat 1&2 for $N$ iterations
4. Select the largest inlier set, and re-estimate the model from all points in this set

---

**How Many Samples?**

- For probability $p$ of having no outliers, we need to sample $N = \frac{\log(1 - p)}{\log(1 - (1 - \epsilon)^s)}$ subsets
  for subset size $s$ and outlier ratio $\epsilon$
- E.g., for $p=0.95$:

<table>
<thead>
<tr>
<th>Sample size</th>
<th>$5%$</th>
<th>$10%$</th>
<th>$20%$</th>
<th>$25%$</th>
<th>$30%$</th>
<th>$40%$</th>
<th>$50%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>17</td>
<td>38</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4</td>
<td>10</td>
<td>16</td>
<td>24</td>
<td>63</td>
<td>191</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>5</td>
<td>13</td>
<td>21</td>
<td>35</td>
<td>108</td>
<td>382</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>6</td>
<td>17</td>
<td>29</td>
<td>51</td>
<td>177</td>
<td>766</td>
</tr>
</tbody>
</table>

---

**PTAM (2007)**

- Architecture optimized for dual cores
- Tracking thread runs in real-time (30Hz)
- Mapping thread is not real-time

---

**Two Examples**

- **PTAM**
- **Photo Tourism**
  [http://phototour.cs.washington.edu/Photo_Tourism.pdf](http://phototour.cs.washington.edu/Photo_Tourism.pdf)
PTAM – Feature Tracking

- Generate 8x8 matching template (warped from key frame to current pose estimate)
- Search a fixed radius around projected position
  - Using SSD
  - Only search at FAST corner points

Dr. Jürgen Sturm, Computer Vision Group, TUM
Visual Navigation for Flying Robots
85

PTAM – Mapping Thread

- Tracking thread
- Mapping thread

PTAM – Example Timings

- Tracking thread
  - Total: 19.2 ms
  - Key frame preparation: 2.2 ms
  - Feature projection: 3.5 ms
  - Patch search: 9.8 ms
  - Iterative pose update: 3.7 ms

- Mapping thread
  - Key frames: 2-49, 50-99, 100-149
    - Local Bundle Adjustment: 170 ms, 170 ms, 440 ms
    - Global Bundle Adjustment: 380 ms, 1.7 s, 6.9 s

PTAM Video

- Parallel Tracking and Mapping for Small AR Workspaces
- Extra video results made for ISMAR 2007 conference
- Georg Klein and David Murray
  - Active Vision Laboratory
  - University of Oxford

Photo Tourism (2006)

- Overview

Photo Tourism – Scene Reconstruction

- Processing pipeline
  - Feature detection (SIFT)
  - Pair-wise matching
  - Correspondence estimation (RANSAC)
  - Incremental structure from motion

- Automatically estimate
  - Position, orientation and focal length of all cameras
  - 3D positions of point features
Photo Tourism – Input Images

Dr. Jürgen Sturm, Computer Vision Group, TUM
Visual Navigation for Flying Robots

Photo Tourism – Feature Detection

Dr. Jürgen Sturm, Computer Vision Group, TUM
Visual Navigation for Flying Robots

Photo Tourism – Feature Matching

Dr. Jürgen Sturm, Computer Vision Group, TUM
Visual Navigation for Flying Robots

Incremental Structure From Motion

- To help get good initializations, start with two images only (compute pose, triangulate points)
- Non-linear optimization
- Iteratively add more images

Lessons Learned Today

- ... how to detect and match feature points
- ... how to compute the camera pose and to triangulate points
- ... how to deal with outliers
Visual Navigation for Flying Robots

Bundle Adjustment and Stereo Correspondence

Dr. Jürgen Sturm

Agenda for Today

- Map optimization
  - Graph SLAM
  - Bundle adjustment
- Depth reconstruction
  - Laser triangulation
  - Structured light (Kinect)
  - Stereo cameras

Remember: 3D Transformations

- Representation as a homogeneous matrix
  \[ M = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \in \text{SE}(3) \subset \mathbb{R}^{4 \times 4} \]
  - Pro: easy to concatenate and invert
  - Con: not minimal

- Representation as a twist coordinates
  \[ \xi = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)^T \in \mathbb{R}^6 \]
  - Pro: minimal
  - Con: need to convert to matrix for concatenation and inversion

Remember: Rodrigues’ formula

- Given: Twist coordinates
  \[ \xi = (\omega^T, v^T)^T = (\omega_x, \omega_y, \omega_z, v_x, v_y, v_z)^T \]
  \[ = (t \omega^T, v^T)^T \] with \( \|\omega\| = 1, t = \|\omega\| \)

- Return: Homogeneous transformation
  \[ R = I + [\omega]_\times \sin(t) + [\omega]^2_\times (1 - \cos t) \]
  \[ t = (I - R)[\omega]_\times v + [\omega]^T v \]
  \[ M = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \]

Project Proposal Presentations

- This Thursday
- Don’t forget to put title, team name, team members on first slide
- Pitch has to fit in 5 minutes (+5 minutes discussion)
- 9 x (5+5) = 90 minutes
- Recommendation: use 3-5 slides

Remember: 3D Transformations

- From twist coordinates to twist
  \[ \hat{\xi} = \begin{pmatrix} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \in \text{se}(3) \]

- Exponential map between se(3) and SE(3)
  \[ M = \exp(\hat{\xi}) \]
  \[ \hat{\xi} = \log M \]
  \[ \hat{\xi} = [\log M]^V \]

alternative notation:
Notation

- Camera poses in a minimal representation (e.g., twists)
  \[ \mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n \]
- ... as transformation matrices
  \[ \mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n \]
- ... as rotation matrices and translation vectors
  \[ (\mathbf{R}_1, \mathbf{t}_1), (\mathbf{R}_2, \mathbf{t}_2), \ldots, (\mathbf{R}_n, \mathbf{t}_n) \]

Incremental Motion Estimation

- **Idea**: Estimate camera motion from frame to frame

Motion concatenation (for twists)

\[ \mathbf{\hat{c}}_j = \log (\exp \mathbf{\hat{c}}_i \exp \mathbf{\hat{z}}_{ij}) \]

Motion composition operator (in general)

\[ \mathbf{c}_j = \mathbf{c}_i \oplus \mathbf{z}_{ij} \]
\[ \mathbf{z}_{ij} = \mathbf{c}_j \ominus \mathbf{c}_i \]

Incremental Motion Estimation

- **Idea**: Estimate camera motion from frame to frame

Loop Closures

- **Idea**: Estimate camera motion from frame to frame

**Problem:**
- Estimates are inherently noisy
- Error accumulates over time → drift
Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame

Two ways to compute $c_n$:

- $c_n = c_{n-1} \oplus z_{(n-1)n}$
- $c'_n = c_1 \oplus z_{1n}$

Graph SLAM

- Use a graph to represent the model
- Every node in the graph corresponds to a pose of the robot during mapping
- Every edge between two nodes corresponds to a spatial constraint between them
- **Graph-based SLAM:** Build the graph and find the robot poses that minimize the error introduced by the constraints

Graph SLAM Architecture

- Interleaving process of front-end and back-end
- A consistent map helps to determine new constraints by reducing the search space
**Problem Definition**

- **Given:** Set of observations $z_{ij} \in \mathbb{R}^6$
- **Wanted:** Set of camera poses $c_1, \ldots, c_n \in \mathbb{R}^6$
  - State vector $x = (c_1^T, \ldots, c_n^T)^T \in \mathbb{R}^{6n}$

**Map Error**

- **Real observation** $z_{ij}$
- **Expected observation** $\tilde{z}_{ij} = c_j \odot c_i$
- **Difference between observation and expectation** $e_{ij} = z_{ij} \odot \tilde{z}_{ij}$
- Given the correct map, this difference is the result of sensor noise...

**Error Function**

- **Assumption:** Sensor noise is normally distributed
  
  $$e_{ij} \sim \mathcal{N}(0, \Sigma_{ij})$$
- **Error term for one observation** (proportional to negative loglikelihood)
  
  $$f_{ij}(x) = e_{ij}(x)^\top \Sigma_{ij}^{-1} e_{ij}(x)$$
- **Note:** error is a scalar $f_{ij}(x) \in \mathbb{R}$

**Non-Linear Optimization Techniques**

- Gradient descend
- Gauss-Newton
- Levenberg-Marquardt

**Gauss-Newton Method**

1. Linearize the error function
2. Compute its derivative
3. Set the derivative to zero
4. Solve the linear system
5. Iterate this procedure until convergence
Step 1: Linearize the Error Function

- Error function
  \[ f(x) = \sum_{ij} f_{ij}(x) = \sum_{ij} e_{ij}(x) \Sigma_{ij}^{-1} e_{ij}(x) \]

- Evaluate the error function around the initial guess
  \[ f(x + \Delta x) = \sum_{ij} e_{ij}(x + \Delta x) \Sigma_{ij}^{-1} e_{ij}(x + \Delta x) \]

Linearize the Error Function

- Approximate the error function around an initial guess \( x \) using Taylor expansion
  \[ e_{ij}(x + \Delta x) \approx e_{ij}(x) + J_{ij} \Delta x \]
  with
  \[ J_{ij}(x) = \begin{pmatrix} \frac{\partial e_{ij}(x)}{\partial c_1} & \frac{\partial e_{ij}(x)}{\partial c_2} & \ldots & \frac{\partial e_{ij}(x)}{\partial c_n} \end{pmatrix} \]

Derivatives of the Error Terms

- Does one error function \( e_{ij}(x) \) depend on all state variables in \( x \)?
  - No, \( e_{ij}(x) \) depends only on \( c_i \) and \( c_j \)

- Is there any consequence on the structure of the Jacobian?
  - Yes, it will be non-zero only in the columns corresponding to \( c_i \) and \( c_j \)
  - Jacobian is sparse
  \[ J_{ij}(x) = \begin{pmatrix} 0 & \ldots & 0 & \frac{\partial e_{ij}(x)}{\partial c_i} & \frac{\partial e_{ij}(x)}{\partial c_j} & \ldots & 0 \end{pmatrix} \]
Linearizing the Error Function

\[ f(x) = \sum_{ij} e_{ij}(x)^T \Sigma_{ij}^{-1} e_{ij}(x) \]
\[ \approx c + 2b^T \Delta x + \Delta x^T H \Delta x \]

with \( b^T = \sum_{ij} e_{ij}^T \Sigma_{ij}^{-1} J_{ij} \)
\[ H = \sum_{ij} J_{ij}^T \Sigma_{ij}^{-1} J_{ij} \]

- What is the structure of \( b^T \) and \( H \)?
  (Remember: all \( J_{ij} \)'s are sparse)

Illustration of the Structure

\[ b_{ij}^T = e_{ij}^T \Sigma_{ij}^{-1} J_{ij} \]
Non-zero only at \( c_i \) and \( c_j \)

\[ H_{ij} = J_{ij}^T \Sigma_{ij}^{-1} J_{ij} \]
Non-zero on the main diagonal at \( c_i \) and \( c_j \)

Illustration of the Structure

\[ b = \sum b_{ij} \]
\( b \): dense vector

\[ H = \sum_{ij} H_{ij} \]
\( H \): sparse block structure with main diagonal

(Linear) Least Squares Minimization

1. Linearize error function
   \[ f(x + \Delta x) \approx c + 2b^T \Delta x + \Delta x^T H \Delta x \]
2. Compute the derivative
   \[ \frac{df(x + \Delta x)}{d\Delta x} = 2b + 2H \Delta x \]
3. Set derivative to zero
   \[ H \Delta x = -b \]
4. Solve this linear system of equations, e.g.,
   \[ \Delta x = -H^{-1}b \]
Gauss-Newton Method

**Problem:** $f(x)$ is non-linear!

**Algorithm:** Repeat until convergence
1. Compute the terms of the linear system
   \[ b^T = \sum_{ij} e_{ij}^T \Sigma_{ij}^{-1} J_{ij} \]
   \[ H = \sum_{ij} J_{ij}^T \Sigma_{ij}^{-1} J_{ij} \]
2. Solve the linear system to get new increment
   \[ H \Delta x = -b \]
3. Update previous estimate
   \[ x \leftarrow x + \Delta x \]

Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots

Sparsity of the Hessian

- The Hessian is
  - positive semi-definit
  - symmetric
  - sparse
- This allows the use of efficient solvers
  - Sparse Cholesky decomposition (~100M matrix elements)
  - Preconditioned conjugate gradients (~1.000M matrix elements)
  - ... many others

Example in 1D

- Two camera poses $c_1, c_2 \in \mathbb{R}$
- State vector $x = (c_1, c_2)^T \in \mathbb{R}^2$
- One (distance) observation $z_{12} \in \mathbb{R}$
- Initial guess $c_1 = c_2 = 0$
- Observation $z_{12} = 1$
- Sensor noise $\Sigma_{12} = 0.5$

\[ z_{12} \leftarrow c_1 \]

\[ z_{12} \leftarrow c_2 \]

What Went Wrong?

- The constraint only specifies a relative constraint between two nodes
- Any poses for the nodes would be fine as long as their relative coordinates fit
- One node needs to be fixed
  - Option 1: Remove one row/column corresponding to the fixed pose
  - Option 2: Add to $H$, a linear constraint $1 \cdot \Delta c_1 = 0$
  - Option 3: Add the identity matrix to $H$ (Levenberg-Marquardt)

Fixing One Node

- The constraint only specifies a relative constraint between two nodes
- Any poses for the nodes would be fine as long as their relative coordinates fit
- One node needs to be fixed (here: Option 2)

\[ H = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \]

\[ \Delta x = -H^{-1}b \]

\[ \Delta x = (0 \ 1)^T \]
Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor
  \[(H + \lambda I)\Delta x = -b\]
  \[(J^T J + \lambda I)\Delta x = -J^T e\]

- What is the effect of this damping factor?
  - Small \(\lambda\)?
  - Large \(\lambda\)?

Non-Linear Minimization

- One of the state-of-the-art solution to compute the maximum likelihood estimate
- Various open-source implementations available
  - g2o [Kuemmerle et al., 2011]
  - sba [Lourakis and Argyros, 2009]
  - iSAM [Kaess et al., 2008]
- Other extensions:
  - Robust error functions
  - Alternative parameterizations

Error Function

- Camera pose \(c_i \in \mathbb{R}^6\)
- Feature point \(p_j \in \mathbb{R}^3\)
- Observed feature location \(z_{ij} \in \mathbb{R}^2\)
- Expected feature location \(g(c_i, p_j) = R_i^T (t_i - p_j)\)
  \[h(c_i, p_j) = g_{x,y}(c_i, p_j)/g_z(c_i, p_j)\]

Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor
  \[(H + \lambda I)\Delta x = -b\]
  \[(J^T J + \lambda I)\Delta x = -J^T e\]

- What is the effect of this damping factor?
  - Small \(\lambda\) \(\rightarrow\) same as least squares
  - Large \(\lambda\) \(\rightarrow\) steepest descent (with small step size)

Algorithm

- If error decreases, accept \(\Delta x\) and reduce \(\lambda\)
- If error increases, reject \(\Delta x\) and increase \(\lambda\)

Bundle Adjustment

- **Graph SLAM:** Optimize (only) the camera poses
  \[x = (c_1^T, \ldots, c_n^T)^T \in \mathbb{R}^{6n}\]

- **Bundle Adjustment:** Optimize both 6DOF camera poses and 3D (feature) points
  \[x = (c_1^T, \ldots, c_n^T, p_1^T, \ldots, p_m^T)^T \in \mathbb{R}^{6n + 3m}\]

- Typically \(m \gg n\) (why?)

Error Function

- Difference between observation and expectation
  \[e_{ij} = z_{ij} - h(c_i, p_j)\]

- Error function
  \[f(c, p) = \sum_{ij} e_{ij}^T \Sigma^{-1} e_{ij}\]

- Covariance \(\Sigma\) is often chosen isotropic and on the order of one pixel
Illustration of the Structure
- Each camera sees several points
- Each point is seen by several cameras
- Cameras are independent of each other (given the points), same for the points

Primary Structure
- Characteristic structure
  \[
  \begin{pmatrix}
  J_c^T J_c & J_c^T J_p \\
  J_p^T J_c & J_p^T J_p
  \end{pmatrix}
  \begin{pmatrix}
  \Delta c \\
  \Delta p
  \end{pmatrix}
  =
  \begin{pmatrix}
  -J_c^T e_c \\
  -J_p^T e_p
  \end{pmatrix}
  \]

- Insight: \( H_{cc} \) and \( H_{pp} \) are block-diagonal (because each constraint depends only on one camera and one point)

  \[
  \begin{pmatrix}
  J_c^T e_c \\
  J_p^T e_p
  \end{pmatrix}
  \]

  This can be efficiently solved using the Schur Complement

Example Hessian (Lourakis and Argyros, 2009)

\[
H =
\]

Schur Complement
- Given: Linear system
  \[
  \begin{pmatrix}
  A & B \\
  C & D
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  =
  \begin{pmatrix}
  a \\
  b
  \end{pmatrix}
  \]

  If \( D \) is invertible, then (using Gauss elimination)

  \[
  (A - BD^{-1}C)x = a - BD^{-1}b
  \]

  \[
  y = D^{-1}(b - Cx)
  \]

  Reduced complexity, i.e., invert one \( p \times p \) and \( p \times p \) matrix instead of one \( (p+q) \times (p+q) \) matrix

From Sparse Maps to Dense Maps
- So far, we only looked at sparse 3D maps
- We know where the (sparse) cameras are
- We know where the (sparse) 3D feature points are
- How can we turn these models into volumetric 3D models?
From Sparse Maps to Dense Maps

- **Today:** Estimation of depth dense images (stereo cameras, laser triangulation, structured light/Kinect)
- **Next week:** Dense map representations and data fusion

Stereo Correspondence Constraints

- Given a point in the left image, where can the corresponding point be in the right image?

Epipolar Plane

- All epipolar lines intersect at the epipoles
- An epipolar plane intersects the left and right image planes in epipolar lines

Human Stereo Vision

Reminder: Epipolar Geometry

- A point in one image “generates” a line in another image (called the **epipolar line**)
- Epipolar constraint \( \mathbf{x}_2^T E \mathbf{x}_1 = 0 \)

Epipolar Constraint

- This is useful because it reduces the correspondence problem to a 1D search along an epipolar line
Examples: Converging and Parallel Cameras

**Rectification**
- In practice, it is convenient if the image scanlines (rows) are the epipolar lines
- Reproject image planes onto a common plane parallel to the baseline (two 3x3 homographies)
- Afterwards pixel motion is horizontal

**Basic Stereo Algorithm**
- For each pixel in the left image
  - Compare with every pixel on the same epipolar line in the right image
  - Pick pixel with minimum matching cost (noisy)
  - Better: match small blocks/patches (SSD, SAD, NCC)

**Block Matching Algorithm**
- **Input:** Two images and camera calibrations
- **Output:** Disparity (or depth) image
- **Algorithm:**
  1. Geometry correction (undistortion and rectification)
  2. Matching cost computation along search window
  3. Extrema extraction (at sub-pixel accuracy)
  4. Post-filtering (clean up noise)
Example

- Input
- Output

Problems with Stereo

- Block matching typically fails in regions with low texture
  - Global optimization/regularization (speciality of our research group)
  - Additional texture projection

Laser Triangulation

Idea:

- Well-defined light pattern (e.g., point or line) projected on scene
- Observed by a line/matrix camera or a position-sensitive device (PSD)
- Simple triangulation to compute distance

What is the Influence of the Block Size?

- Common choices are 5x5 .. 11x11
- Smaller neighborhood: more details
- Larger neighborhood: less noise
- Suppress pixels with low confidence (e.g., check ratio best match vs. 2nd best match)
Example: Neato XV-11

- Specs: 360deg, 10Hz, 30 USD

How Does the Data Look Like?

Laser Triangulation

- Stripe laser + 2D camera
- Often used on conveyer belts (volume sensing)
- Large baseline gives better depth resolution but more occlusions → use two cameras

Structured Light

- Multiple stripes / 2D pattern
- Data association more difficult
- Coding schemes
  - Temporal: Coded light
  - Wavelength: Color
  - Spatial: Pattern (e.g., diffraction patterns)
Sensor Principle of Kinect

- Kinect projects a diffraction pattern (speckles) in near-infrared light
- CMOS IR camera observes the scene

Infrared pattern projector

Color camera

“stereo” Baseline

Infrared image (with distorted pattern)

Standard block matcher (9x9)

Disparity image

Depth image (color encodes distance from camera)

Example Data

- Kinect provides color (RGB) and depth (D) video
- This allows for novel approaches for (robot) perception

Sensor Principle of Kinect

- Pattern is memorized at a known depth
- For each pixel in the IR image
  - Extract 9x9 template from memorized pattern
  - Correlate with current IR image over 64 pixels and search for the maximum
  - Interpolate maximum to obtain sub-pixel accuracy (1/8 pixel)
- Calculate depth by triangulation

Technical Specs

- Infrared camera has 640x480 @ 30 Hz
- Depth correlation runs on FPGA
- 11-bit depth image
- 0.8m – 5m range
- Depth sensing does not work in direct sunlight (why?)
- RGB camera has 640x480 @ 30 Hz
- Bayer color filter
- Four 16-bit microphones with DSP for beam forming @ 16kHz
- Requires 12V (for motor), weighs 500 grams
- Human pose recognition runs on Xbox CPU and uses only 10-15% processing power @ 30 Hz

History

- 2005: Developed by PrimeSense (Israel)
- 2006: Offer to Nintendo and Microsoft, both companies declined
- 2007: Alex Kidman becomes new incubation director at Microsoft, decides to explore PrimeSense device. Johnny Lee assembles a team to investigate technology and develop game concepts
- 2008: The group around Prof. Andrew Blake and Jamie Shotton (Microsoft Research) develops pose recognition
- 2009: The group around Prof. Dieter Fox (Intel Labs / Univ. of Washington) works on RGB-D mapping and RGB-D object recognition
- Nov 4, 2010: Official market launch
- Nov 10, 2010: First open-source driver available
- 2011: First programming competitions (ROS 3D, PrimeSense), First workshops (RSS, Euron)
- 2012: First special issues (JVCI, T-SMC)
Impact of the Kinect Sensor

- Sold >18M units, >8M in first 60 days (Guiness: “fastest selling consumer electronics device”)
- Has become a “standard” sensor in robotics

Kinect: Applications

Open Research Questions

- How can RGB-D sensing facilitate in solving hard perception problems in robotics?
  - Interest points and feature descriptors?
  - Simultaneous localization and mapping?
  - Collision avoidance and visual navigation?
  - Object recognition and localization?
  - Human-robot interaction?
  - Semantic scene interpretation?
Exercise Sheet 5

Visual Navigation for Flying Robots
Place Recognition, ICP, and Dense Reconstruction
Dr. Jürgen Sturm

Agenda for Today
- **Localization**
  - Visual place recognition
  - Scan matching and Iterative Closest Point
- Mapping with known poses (3D reconstruction)
  - Occupancy grids
  - Octtrees
  - Signed distance field
  - Meshing

Remember: Loop Closures
- Use loop-closures to minimize the drift / minimize the error over all constraints

Loop Closures
How can we detect loop closures efficiently?
1. Compare with all previous images $O(n)$
   (not efficient)

1. Prepare mid-term presentation
2. Proposed structure: 3 slides
   1. Remind people who you are and what you are doing (can be same slide as last time)
   2. Your work/achievements so far (video is a plus)
   3. Your plans for the next two weeks
3. Hand in slides before July 3, 10am

Dr. Jürgen Sturm, Computer Vision Group, TUM
Loop Closures

How can we detect loop closures efficiently?

2. Use motion model and covariance to limit search radius (metric approach)

Appearance-based Place Recognition

Appearance can help to recover the pose estimate where metric approaches might fail

Object/Scene Recognition

- Analogy to documents: The content can be inferred from the frequency of visual words

Bag of Visual Words

- Visual words = (independent) features
Bag of Visual Words

- Visual words = (independent) features
- Construct a dictionary of representative words

Dictionary of visual words (codebook)

Overview

- Feature detection and extraction (e.g., SIFT, …)
- Codewords dictionary
- Image representation (histogram of word occurrences)

Learning the Dictionary

- Clustering, e.g., k-means
- Cluster center = code words
- Descriptor vectors (e.g., SIFT, SURF, …)
Learning the Visual Vocabulary

Example Image Representation

- Build the histogram by assigning each detected feature to the closest entry in the codebook

Object/Scene Recognition

- Compare histogram of new scene with those of known scenes, e.g., using
  - simple histogram intersection
    \[ \text{score}(p, q) = \sum \min(p_i, q_i) \]
  - naïve Bayes
  - more advanced statistical methods

Timing Performance

- Inference: 25 ms for 100k locations
- SURF detection + quantization: 483 ms

Example: FAB-MAP

[Cummins and Newman, 2008]

Summary: Bag of Words

[Fei-Fei and Perona, 2005; Nister and Stewenius, 2006]

- Compact representation of content
- Highly efficient and scalable
- Requires training of a dictionary
- Insensitive to viewpoint changes/image deformations (inherited from feature descriptor)
**Laser-based Motion Estimation**

- So far, we looked at motion estimation (and place recognition) from **visual** sensors
- Today, we cover motion estimation from **range** sensors
  - Laser scanner (laser range finder, ultrasound)
  - Depth cameras (time-of-flight, Kinect ...)

---

**Laser Scanner**

- Measures angles and distances to closest obstacles
  \[ z = (\theta_1, z_1, \ldots, \theta_n, z_n) \in \mathbb{R}^{2n} \]
- Alternative representation: 2D point set (cloud)
  \[ z = (x_1, y_1, \ldots, x_n, y_n)^T \in \mathbb{R}^{2n} \]
- Probabilistic sensor model \( p(z \mid x) \)

---

**Laser-based Motion Estimation**

How can we best align two laser scans?

- Exhaustive search
- Feature extraction (lines, corners, ...)
- Iterative minimization (ICP)

---

**Exhaustive Search**

- Convolve first scan with sensor model
- Sweep second scan over likelihood map, compute correlation and select best pose

---

**Example: Exhaustive Search** [Olson, ’09]

- Multi-resolution correlative scan matching
- Real-time by using GPU
- Remember: SE(2) has 3 DOFs
Does Exhaustive Search Generalize To 3D As Well?

Iterative Closest Point (ICP)

- **Given:** Two corresponding point sets (clouds)
  \[ P = \{ p_1, \ldots, p_n \}, \]
  \[ Q = \{ q_1, \ldots, q_n \}. \]

- **Wanted:** Translation \( t \) and rotation \( R \) that minimize the sum of the squared error
  \[ E(R, t) = \frac{1}{n} \sum_{i=1}^{n} ||p_i - Rq_i - t||^2 \]

  where \( p_i \) and \( q_i \) are corresponding points.

Known Correspondences

- **Note:** If the correct correspondences are known, both rotation and translation can be calculated in closed form.

Known Correspondences

- **Idea:** The center of mass of both point sets has to match
  \[ \bar{p} = \frac{1}{n} \sum_i p_i, \quad \bar{q} = \frac{1}{n} \sum_i q_i. \]

  Subtract the corresponding center of mass from every point.

  Afterwards, the point sets are zero-centered, i.e., we only need to recover the rotation.

Unknown Correspondences

- **Decompose the matrix**
  \[ W = \sum_i (p_i - \bar{p})(q_i - \bar{q})^T = USV^T \]

  using singular value decomposition (SVD).

- **Theorem**
  If \( \text{rank} \ W = 3 \), the optimal solution of \( E(R, t) \) is unique and given by
  \[ R = UV^T, \]
  \[ t = \bar{p} - R\bar{q} \]

  (for proof, see [http://hss.ulb.uni-bonn.de/2006/0912/0912.pdf](http://hss.ulb.uni-bonn.de/2006/0912/0912.pdf), p.34/35)
ICP Algorithm

[Besl & McKay, 92]

- **Algorithm:** Iterate until convergence
  - Find correspondences
  - Solve for $R, t$
  - Converges if starting position is “close enough”

Example: ICP

ICP Variants

Many variants on all stages of ICP have been proposed:
- **Selecting** and **weighting** source points
- **Finding** corresponding points
- Rejecting certain (outlier) correspondences
- Choosing an **error metric**
- **Minimization**

Performance Criteria

- Various aspects of performance
  - Speed
  - Stability (local minima)
  - Tolerance w.r.t. noise and/or outliers
  - **Basin of convergence** (maximum initial misalignment)
- Choice depends on data and application

Selecting Source Points

- Use all points
- Uniform sub-sampling
- Random sampling
- Feature-based sampling
- Normal-space sampling
  - Ensure that samples have normals distributed as uniformly as possible

Spatially Uniform Sampling

- Density of points usually depends on the distance to the sensor → no uniform distribution
- Can lead to a bias in ICP
Feature-based Sampling

Detect interest points (same as with images)
- Decrease the number of correspondences
- Increase efficiency and accuracy
- Requires pre-processing

Example: Normal-Space Sampling

Normal-space sampling can help on mostly-smooth areas with sparse features

Finding Correspondences

Has greatest effect on convergence and speed
- Closest point
- Normal shooting
- Closest compatible point
- Projection
- Speed-up using kd-trees (or oct-trees)

Closest Point Matching

- Find closest point in the other point set
- Distance threshold

Selection and Weighting

- Selection is a form of (binary) weighting
- Instead of re-sampling one can also use weighting
- Weighting strategy depends on the data
- Pre-processing / run-time trade-off

Normal-Space Sampling

Uniform sampling
Normal-space sampling
Normal Shooting

- Project along normal, intersect other mesh
- Slightly better than closest point for smooth meshes, worse for noisy or complex meshes

Closest Compatible Point

- Can improve effectiveness of both the previous variants by only matching to compatible points
- Compatibility based on normals, colors, ...
- In the limit, degenerates to feature matching

Speeding Up Correspondence Search

Finding closest point is most expensive stage of the ICP algorithm
- Build index for one point set (kd-tree)
- Use simpler algorithm (e.g., projection-based matching)

Projection-based Matching

- Slightly worse performance per iteration
- Each iteration is one to two orders of magnitude faster than closest-point
- Requires point-to-plane error metric

Error Metrics

- Point-to-point
- Point-to-plane lets flat regions slide along each other
- Generalized ICP: Assign individual covariance to each data point [Segal, 2009]

Minimization

- Only point-to-point metric has closed form solution(s)
- Other error metrics require non-linear minimization methods
  - Which non-linear minimization methods do you remember?
  - Which robust error metrics do you remember?
Robust Error Metrics

Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots

Example: Real-Time ICP on Range Images

[Rusinkiewicz and Levoy, 2001]

- Real-time scan alignment
- Range images from structure light system (projector and camera, temporal coding)

ICP: Summary

- ICP is a powerful algorithm for calculating the displacement between point clouds
- The overall speed depends most on the choice of matching algorithm
- ICP is (in general) only locally optimal \( \rightarrow \) can get stuck in local minima

Agenda for Today

- Localization
- Visual place recognition
- Scan matching and Iterative Closest Point
- Mapping with known poses (3D reconstruction)
- Occupancy grids
- Octtrees
- Signed distance field
- Meshing

Occupancy Grid

Idea:

- Represent the map \( m \) using a grid
- Each cell is either free or occupied
- \( m = (m_1, \ldots, m_n) \in \{\text{empty, occ}\}^n \)
- Robot maintains a belief \( \text{Bel}(m) \) on map state

Goal: Estimate the belief from sensor observations

\[
\text{Bel}(m) = P(m \mid z_1, \ldots, z_i)
\]
Mapping

- **Goal:** Estimate
  \[ \text{Bel}(m) = P(m \mid z_1, \ldots, z_n) \]
- How can this be computed?

Binary Bayes Filter

- **Goal:** Estimate
  \[ \text{Bel}(m) = P(m \mid z_1, \ldots, z_n) \]
- How can this be computed?
- E.g., using the Bayes Filter from Lecture 3

\[
P(m \mid z_{1:t}) = \left( 1 + \frac{1 - P(m \mid z_t) - P(m \mid z_{1:t-1})}{P(m \mid z_t) - P(m \mid z_{1:t-1})} \right)^{-1}
\]

Clamping Update Policy

- Often, the world is not “fully” static
- Consider an appearing/disappearing obstacle
- To change the state of a cell, the filter needs as many positive (negative) observations
- **Idea:** Clamp the beliefs to min/max values

\[
L'(m \mid z_{1:t}) = \max(\min(L(m \mid z_{1:t}), t_{\text{max}}), t_{\text{min}})
\]

Sensor Model

- For the Bayes filter, we need the inverse sensor model
  \[ p(m \mid z) \]
- Let’s consider an ultrasound sensor
  - Located at (0,0)
  - Measures distance of 2.5m
- How does the inverse sensor model look like?

Inverse sensor model \( p(m \mid z) \) is specific to the sensor used for mapping.
The log-odds representation can be used to increase speed and numerical stability.

\[
L(x) := \log \frac{p(x)}{p(-x)} = \log \frac{p(x)}{1 - p(x)}
\]
Typical Sensor Model for Ultrasound

- Combination of a linear function (in x-direction) and a Gaussian (in y-direction)

- Question: What about a laser scanner?

Example: Updating the Occupancy Grid

Note: The maximum likelihood map is obtained by clipping the occupancy grid map at a threshold of 0.5

Memory Consumption

- Consider we want to map a building with 40x40m at a resolution of 0.05cm
- How much memory do we need?
  \[
  \left(\frac{40}{0.05}\right)^2 = 640,000 \text{ cells} = 4.88\text{mb}
  \]
- And for 3D?
  \[
  \left(\frac{40}{0.05}\right)^3 = 512,000,000 \text{ cells} = 3.8\text{gb}
  \]
- And what about a whole city?

Map Representation by Octtrees

- Tree-based data structure
- Recursive subdivision of space into octants
- Volumes can be allocated as needed
- Multi-resolution
Example: OctoMap
[Wurm et al., 2011]

- Freiburg, building 79
  44 x 18 x 3 m³, 0.05m resolution, 0.7mb on disk

Example: OctoMap
[Wurm et al., 2011]

- Freiburg computer science campus
  292 x 167 x 28 m³, 0.2m resolution, 2mb on disk

Signed Distance Field (SDF)
[Curless and Levoy, 1996]

- Idea: Instead of representing the cell occupancy, represent the distance of each cell to the surface

  - Occupancy grid maps: explicit representation

  - SDF: implicit representation

Signed Distance Field (SDF)
[Curless and Levoy, 1996]

Algorithm:
1. Estimate the signed distance field
2. Extract the surface using interpolation
   (surface is located at zero-crossing)

Weighting Function

- Weight each observation according to its confidence

  - Weight can additionally be influenced by other modalities (reflectance values, ...)

Data Fusion

- Each voxel cell \( x \) in the SDF stores two values
  - Weighted sum of signed distances \( D_t(x) \)
  - Sum of all weights \( W_t(x) \)

- When new range image arrives, update every voxel cell according to
  \[
  D_{t+1}(x) = D_t(x) + w_{t+1}(x)d_{t+1}(x)
  \]
  \[
  W_{t+1}(x) = W_t(x) + w_{t+1}(x)
  \]
Two Nice Properties

- Noise cancels out over multiple measurements
- Zero-crossing can be extracted at sub-voxel accuracy (least squares estimate)

1D Example: \[ x^* = \frac{\sum D_i(x)x}{\sum W_i(x)x} \]

SDF Example

A cross section through a 3D signed distance function of a real scene

Surface with cross-section

SDF

brightness encodes \( D(x)/W(x) \)

Visualizing Signed Distance Fields

Common approaches to iso surface extraction:

1. Ray casting (GPU, fast)
   For each camera pixel, shoot a ray and search for zero crossing
2. Polygonization (CPU, slow)
   E.g., using the marching cubes algorithm
   Advantage: outputs triangle mesh

Ray Casting

- Interpolation reduces artifacts
- Close to surface, gradient represents the surface normal
Marching Cubes

First in 2D, **marching squares**:
- Evaluate each cell separately
- Check which edges are inside/outside
- Generate triangles according to lookup table
- Locate vertices using least squares

![Marching Cubes Diagram](image)

KinectFusion

[Newcombe et al., 2011]
- Projective ICP with point-to-plane metric
- Truncated signed distance function (TSDF)
- Ray Casting

![KinectFusion Diagram](image)

Lessons Learned Today
- How to quickly recognize previously seen places
- How to align point clouds
- How to estimate occupancy maps
- How to reconstruct triangle meshes at sub-voxel accuracy

![Lessons Learned Today Diagram](image)
Visual Navigation for Flying Robots

Motion Planning

Dr. Jürgen Sturm

Motion Planning Problem

What are good performance metrics?

Motivation: Flying Through Forests

Motion Planning Problem

What are good performance metrics?

- Execution speed / path length
- Energy consumption
- Planning speed
- Safety (minimum distance to obstacles)
- Robustness against disturbances
- Probability of success
- ...

in.tum.summer party & career forum

The Department of Informatics would like to invite its students and employees to its summer party and career forum.

July 4, 2012

3 pm – 6 pm Career Forum: Presentations given by Google, Capgemini etc.

3 pm – 6 pm Foosball Tournament

Starting at 5 pm Summer Party: BBQ, live band and lots of fun!

www.in.tum.de/2012summerparty
Motion Planning Examples

Motion planning is sometimes also called the piano mover’s problem

Robot Architecture

- Configuration spaces
- Roadmap construction
- Search algorithms
- Path optimization and re-planning
- Path execution

Configuration Space

- The configuration space (C-space) is the space of all possible configurations
- C-space topology is usually not Cartesian
- C-space is described as a topological manifold

Notation

- Configuration space $C \subseteq \mathbb{R}^d$
- Configuration $q \in C$
- Free space $C_{\text{free}}$
- Obstacle space $C_{\text{obs}}$

- Properties
  - $C_{\text{free}} \cup C_{\text{obs}} = C$
  - $C_{\text{free}} \cap C_{\text{obs}} = \emptyset$
Free Space Example

- What are admissible configurations for the robot? Equiv.: What is the free space?
  - “Point” robot

Example

- What are admissible configurations for the robot? Equiv.: What is the free space?
  - Circular robot

Example

- What are admissible configurations for the robot? Equiv.: What is the free space?
  - Large circular robot

Computing the Free Space

- Free configuration space is obtained by sliding the robot along the edge of the obstacle regions "blowing them up" by the robot radius
- This operation is called the Minowski sum

\[ A \oplus B = \{a + b | a \in A, b \in B\} \]

where \( A, B \subset \mathbb{R}^d \)
Example: Minowski Sum

- Triangular robot and rectangular obstacle

![Diagram: Minowski Sum]

Basic Motion Planning Problem

- **Given**
  - Free space $C_{free}$
  - Initial configuration $q_I$
  - Goal configuration $q_G$

- **Goal:** Find a continuous path

\[ \tau : [0,1] \rightarrow C_{free} \]

with $\tau(0) = q_I$, $\tau(1) = q_G$

Motion Planning Sub-Problems

1. **C-Space discretization**
   (generating a graph / roadmap)
2. Search algorithm
   (Dijkstra’s algorithm, A*, ...)
3. Re-planning
   (D*, ...)
4. Path tracking
   (PID control, potential fields, funnels, ...)

C-Space Discretizations

- Two competing paradigms
  - Combinatorial planning
    (exact planning)
  - Sampling-based planning
    (probabilistic/randomized planning)

Combinatorial Methods

- Mostly developed in the 1980s
- Extremely efficient for low-dimensional problems
- Sometimes difficult to implement
- Usually produce a road map in $C_{free}$
- Assume polygonal environments
Roadmaps

A roadmap is a graph in $C_{\text{free}}$ where
- Each vertex is a configuration $q \in C_{\text{free}}$
- Each edge is a path $\tau : [0, 1] \rightarrow C_{\text{free}}$ for which $\tau(0)$ and $\tau(1)$ are vertices

(Desired) Properties of Roadmaps

- **Accessibility**
  From anywhere in $C_{\text{free}}$, it is easy to compute a path that reaches at least one of the vertices
- **Connectivity-preserving**
  If there exists a path between $q_I$ and $q_G$ in $C_{\text{free}}$, then there must also exist a path in the roadmap

Roadmap Construction

We consider here three combinatorial methods:
- Trapezoidal decomposition
- Shortest path roadmap
- Regular grid
- ... but there are many more!

Afterwards, we consider two sampling-based methods:
- Probabilistic roadmaps (PRMs)
- Rapidly exploring random trees (RRTs)

Example Query

Compute path from $q_I$ to $q_G$
- Identify start and goal trapezoid
- Connect start and goal location to center vertex
- Run search algorithm (e.g., Dijkstra)
Properties of Trapezoidal Decomposition
+ Easy to implement
+ Efficient computation
+ Scales to 3D
- Does not generate shortest path

Shortest-Path Roadmap
- Contains all vertices and edges that optimal paths follow when obstructed
- Imagine pulling a tight string between \( q_I \) and \( q_G \)

Roadmap Construction
- Vertices = all sharp corners (>180deg, red)
- Edges
  1. Two consecutive sharp corners on the same obstacle (light blue)
  2. Bitangent edges (when line connecting two vertices extends into free space, dark blue)

Example Query
+ Easy to construct in 2D
+ Generates shortest paths
- Optimal planning in 3D or more dim. is NP-hard

Approximate Decompositions
- Construct a regular grid
- High memory consumption (and number of tests)
- Any ideas?
Approximate Decompositions

- Construct a regular grid
- Use quadtree/octtree to save memory
- Sometimes difficult to determine status of cell

Sample visual representations of approximate decompositions.

Summary: Combinatorial Planning

- **Pro:** Find a solution when one exists (complete)
- **Con:** Become quickly intractable for higher dimensions
- **Alternative:** Sampling-based planning
  - Weaker guarantees but more efficient

Sampling-based Methods

- Abandon the concept of explicitly characterizing $C_{\text{free}}$ and $C_{\text{obst}}$ and leave the algorithm in the dark when exploring $C_{\text{free}}$
- The only light is provided by a collision-detection algorithm that probes $C$ to see whether some configuration lies in $C_{\text{free}}$
- We will have a look at
  - Probabilistic road maps (PRMs)
  - Rapidly exploring random trees (RRTs)

Probabilistic Roadmaps (PRMs)

[Kavraki et al., 1992]

- **Vertex:** Take random sample from $C$, check whether sample is in $C_{\text{free}}$
- **Edge:** Check whether line-of-sight between two nearby vertices is collision-free
- Options for “nearby”: k-nearest neighbors or all neighbors within specified radius
- Add vertices and edges until roadmap is dense enough

PRM Example

1. Sample vertex
2. Find neighbors
3. Add edges

Step 3: Check edges for collisions, e.g., using discretized line search
**Probabilistic Roadmaps**

- Probabilistic, complete
- Scale well to higher dimensional C-spaces
- Very popular, many extensions

**Rapidly Exploring Random Trees**

- **Idea:** Grow tree from start to goal location

  ![Existing RRT is “grown” as follows...](image)

**Algorithm**

1. Initialize tree with first node \( q_I \)
2. Pick a random target location (every 100th iteration, choose \( q_G \))
3. Find closest vertex in roadmap
4. Extend this vertex towards target location
5. Repeat steps until goal is reached

**Why not pick \( q_G \) every time?**

- This will fail and run into \( C_{obs} \) instead of exploring

**RRT Examples**

- 2-DOF example
- 3-DOF example (2D translation + rotation)
Non-Holonomic Robots

- Some robots cannot move freely on the configuration space manifold
- Example: A car can not move sideways
  - 2-DOF controls (speed and steering)
  - 3-DOF configuration space (2D translation + rotation)

Rapidly Exploring Random Trees

+ Probabilistic. complete
+ Balance between greedy search and exploration
+ Very popular, many extensions
- Metric sensitivity
- Unknown rate of convergence
- Not optimal, not complete

Summary: Sampling-based Planning

- More efficient in most practical problems but offer weaker guarantees
- Probabilistically complete (given enough time it finds a solution if one exists, otherwise, it may run forever)
- Performance degrades in problems with narrow passages

Motion Planning Sub-Problems

1. C-Space discretization
   (generating a graph / roadmap)
2. Search algorithms
   (Dijkstra’s algorithm, A*, ...)
3. Re-planning
   (D*, ...)
4. Path tracking
   (PID control, potential fields, funnels, ...)

Search Algorithms

- Given: Graph G consisting of vertices and edges (with associated costs)
- Wanted: find the best (shortest) path between two vertices
- What search algorithms do you know?
Uninformed Search

- **Breadth-first**
  - Complete
  - Optimal if action costs equal
  - Time and space $O(b^d)$

- **Depth-first**
  - Not complete in infinite spaces
  - Not optimal
  - Time $O(b^d)$
  - Space $O(bd)$
    (can forget explored subtrees)

Informed Search

- **Idea**
  - Select nodes for further expansion based on an evaluation function $f(n)$
  - First explore the node with lowest value
  - What is a good evaluation function?

Informed Search

- **Greedy best-first search**
  - Simply expand the node closest to the goal $f(n) = h(n)$
  - Not optimal, not complete

- **A* search**
  - Combines path cost with estimated goal distance $f(n) = g(n) + h(n)$
  - Optimal and complete if $h(n)$ never overestimates actual cost

Example: Dijkstra’s Algorithm

- Extension of breadth-first with arbitrary (non-negative) costs

What is a Good Heuristic Function?

- Choice is problem/application-specific
- Two popular choices
  - Manhattan distance (neglecting obstacles)
  - Euclidean distance (neglecting obstacles)
- Value iteration / Dijkstra (from the goal backwards)
Comparison Search Algorithms

Problems on A* on Grids

1. The shortest path is often very close to obstacles (cutting corners)
   - Uncertain path execution increases the risk of collisions
   - Uncertainty can come from delocalized robot, imperfect map, or poorly modeled dynamic constraints

2. Trajectories are aligned to grid structure
   - Path looks unnatural
   - Paths are longer than the true shortest path in continuous space

3. When the path turns out to be blocked during traversal, it needs to be re-planned from scratch
   - In unknown or dynamic environments, this can occur very often
   - Replanning in large state spaces is costly
   - Can we re-use (repair) the initial plan?

Let’s look at solutions to these problems...

Map Smoothing

- Problem: Path gets close to obstacles
- Solution: Convolve the map with a kernel (e.g., Gaussian)

\[ f(n) = g(n) \cdot p_{occ}(n) + h(n) \]

- Leads to non-zero probability around obstacles
- Evaluation function

Path Smoothing

- Problem: Paths are aligned to grid structure (because they have to lie in the roadmap)
- Paths look unnatural and are sub-optimal
- Solution: Smooth the path after generation
  - Traverse path and find pairs of nodes with direct line of sight; replace by line segment
  - Refine initial path using non-linear minimization (e.g., optimize for continuity/energy/execution time)

Example: Map Smoothing
Example: Path Smoothing

- Replace pairs of nodes by line segments
- Non-linear optimization

D* Search

- **Problem:** In unknown, partially known or dynamic environments, the planned path may be blocked and we need to **replan**
- Can this be done efficiently, avoiding to replan the entire path?

Main concepts

- **Invert search direction** (from goal to start)
  - Goal does not move, but robot does
  - Map changes (new obstacles) have only local influence close to current robot pose
- **Mark** the changed node and all dependent nodes as unclean (=to be re-evaluated)
- **Find shortest path** to start (using A*) while reusing previous solution

D* Example

- **Situation at start**

  - Breadth-First-Search
  - A*
  - D* Lite

- **After discovery of blocked cell**

  - All other nodes remain unaltered, the shortest path can reuse them.
**D* Search**

- D* is as optimal and complete as A*
- D* and its variants are widely used in practice
- Field D* was running on Mars rovers Spirit and Opportunity

**Real-Time Motion Planning**

- What is the maximum time needed to re-plan in case of an obstacle detection?
- What if the robot has to react quickly to unforeseen, fast moving objects?
- Do we really need to re-plan for every obstacle on the way?

**Layered Motion Planning**

- An approximate global planner computes paths ignoring the kinematic and dynamic vehicle constraints (not real-time)
- An accurate local planner accounts for the constraints and generates feasible local trajectories in real-time (collision avoidance)

---

**Robot Architecture**

- Robot
  - Global Map (SLAM)
  - Local Obstacle Map
  - Localization
  - Actuators
  - Sensors
- Physical World
  - Executive
  - Path Planner
  - Path Tracking
  - Collision Avoidance
  - Position Control
Local Planner

- **Given:** Path to goal (sequence of via points), range scan of the local vicinity, dynamic constraints
- **Wanted:** Collision-free, safe, and fast motion towards the goal (or next via point)
- Typical approaches:
  - Potential fields
  - Dynamic window approach

Navigation with Potential Fields

- Treat robot as a particle under the influence of a potential field
- **Pro:**
  - easy to implement
- **Con:**
  - suffers from local minima
  - no consideration of dynamic constraints

**Dynamic Window Approach**

- Consider a 2D planar robot
  - all possible speeds of the robot $V_a$
  - forward velocity $0.9 \text{m/s}$
  - angular velocity $-90 \text{deg/s}$ to $+90 \text{deg/s}$

- Consider additionally dynamic constraints
  - all possible speeds of the robot $V_a$
  - forward velocity $0.9 \text{m/s}$
  - angular velocity $-90 \text{deg/s}$ to $+90 \text{deg/s}$
  - obstacle-free area $V_a$
  - current robot speed
  - dynamic window $V_d$ (speeds reachable in one time frame)
  - Admissible space $V_a \cap V_n \cap V_d$

- Navigation function (potential field)
  - Maximizes velocity
  - Current robot pose
  - Path from A*
  - Forward velocity $0.5 \text{m/s}$
Dynamic Window Approach
[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Navigation function (potential field)

Maximizes velocity
Rewards alignment to A* path gradient

Current robot pose
Path from A*

Forward velocity

Example: Dynamic Window Approach
[Brock and Khatib, ICRA '99]

Problems of DWAs
- DWAs suffer from local minima (need tuning), e.g., robot does not slow down early enough to enter doorway:
- Can you think of a solution?
- Note: General case requires global planning

Lessons Learned Today
- Motion planning problem and configuration spaces
- Roadmap construction
- Search algorithms and path optimization
- Local planning for path execution

Dr. Jürgen Sturm, Computer Vision Group, TUM
Visual Navigation for Flying Robots

Planning under Uncertainty, Exploration and Coordination

Dr. Jürgen Sturm

Agenda for Today
- Planning under Uncertainty
- Exploration with a single robot
- Coordinated exploration with a team of robots
- Coverage

Agenda For Next Week
- First half: Good practices for experimentation, evaluation and benchmarking
- Second half: Time for your questions on course material

⇒ Prepare your questions (if you have)

Motivation: Planning under Uncertainty
- Consider a robot with range-limited sensors and a feature-poor environment
- Which route should the robot take?

Reminder: Performance Metrics
- Execution speed / path length
- Energy consumption
- Planning speed
- Safety (minimum distance to obstacles)
- Robustness against disturbances
- Probability of success
- ...

Reminder: Belief Distributions
- In general, actions of the robot are not carried out perfectly
- Position estimation ability depends on map
- Let’s look at the belief distributions...
Reminder: Belief Distributions

- Actions increase the uncertainty (in general)
- Observations decrease the uncertainty (always)
- Observations are not always available

Solution 1: Shape The Environment To Decrease Uncertainty

- Plan 1: Take the shortest path
- What is the probability of success of plan 1?

Solution 2: Add (More/Better) Sensors

- Pro: Simple solution, need fewer/no sensors
- Con: Requires task specific design/engineering of both the robot and the environment
- Applications:
  - Docking station
  - Perception-less manipulation (on conveyor belts)
  - ...

Solution 1: Shape The Environment To Decrease Uncertainty

- Assume a robot without sensors
- What is a good navigation plan?

Solution 2: Add (More/Better) Sensors
**Solution 3: POMDPs**

- Partially observable Markov decision process (POMDP)
- Considers uncertainty of the motion model and sensor model
- Finite/infinite time horizon
- Resulting policy is optimal
- One solution technique: Value iteration
- **Problem:** In general (and in practice) computationally intractable (PSPACE-hard)

---

**Continuum of Possible Approaches to Motion Planning**

<table>
<thead>
<tr>
<th>Conventional path planner</th>
<th>POMDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>tractable</td>
<td>intractable</td>
</tr>
</tbody>
</table>

**Remember: Motion Planning**

- Problem: The roadmap does not consider the sensor capabilities of the robot
- Can the robot actually keep position at each vertex?
  - Can it localize at the vertex?
  - Given localization abilities, what is the probability of hitting into an obstacle?
- Can the robot robustly navigate between two vertices?
  - Line of sight is not enough
  - Robot might get lost or hit into an obstacle

---

**Remember: Motion Planning in High-Dimensional Configuration Spaces**

- Problem: The roadmap does not consider the sensor capabilities of the robot
- Can the robot actually keep position at each vertex?
  - Can it localize at the vertex?
  - Given localization abilities, what is the probability of hitting into an obstacle?
- Can the robot robustly navigate between two vertices?
  - Line of sight is not enough
  - Robot might get lost or hit into an obstacle

---

**Motion Planning in Information Space [Roy et al.]**

1. Sample vertices and localization distributions where \( p(x \in C_{\text{obst}}) < \epsilon \)
2. Add edges between points where \( p(x \in C_{\text{obst}}) < \epsilon \) along path
3. Perform graph search
Motion Planning in Information Space

- **Problem:** Posterior distribution depends also on the path taken to the vertex
- **Example**

Planning in Information Spaces

- **Given:** Roadmap
- **Goal:** Find path from start to goal nodes that results in minimum uncertainty at goal
- **Problem:** How can we estimate the belief distribution at the goal (efficiently)?

Example: Belief Roadmap

- **Belief Roadmap**
  [He et al., 2008]
  
  How can we propagate the belief distribution along an edge?
  1. Sample waypoints, use forward simulation to compute full posterior
  2. Linearize model and use Kalman filter

Belief Propagation

- **The posterior distribution depends on the prior distribution**
Planning in Information Spaces
[He et al., 2008]

- The posterior distribution at a vertex depends on the prior distribution (and thus on path to the vertex)
- Need to perform forward simulation (and belief prediction) along each edge for every start state
- Computing minimum cost path of 30 edges: \(\approx 100\) seconds

Summary: Planning Under Uncertainty

- Actions and observations are inherently noisy
- Planners neglecting this are not robust
- Consider the uncertainty during planning to increase robustness

Mission Planning

- Goal: Generate and execute a plan to accomplish a certain (navigation) task
- Example tasks
  - Exploration
  - Coverage
  - Surveillance
  - Tracking
  - ...

Task Planning

- Goal: Generate and execute a high level plan to accomplish a certain task
- Often symbolic reasoning (or hard-coded)
  - Propositional or first-order logic
  - Automated reasoning systems
  - Common programming languages: Prolog, LISP
- Multi-agent systems, communication
- Artificial Intelligence

Exploration and SLAM

- SLAM is typically passive, because it consumes incoming sensor data
- Exploration actively guides the robot to cover the environment with its sensors
- Exploration in combination with SLAM: Acting under pose and map uncertainty
- Uncertainty should/needs to be taken into account when selecting an action
**Exploration**

- By reasoning about control, the mapping process can be made much more effective
- Question: Where to move next?

- This is also called the *next-best-view problem*

**Example**

- Where should the robot go next?

**Information Theory**

- **Entropy** is a general measure for the uncertainty of a probability distribution
- Entropy $= \text{Expected amount of information needed to encode an outcome } X = x$
  
  $$H(X) = E(I(X)) = E(- \log p(X)) = - \sum_{i=1}^{n} p(x_i) \log p(x_i)$$

**Exploration**

- Choose the action that maximizes utility
  
  $$a^* = \arg \max_{a \in A} U(m, a)$$

- Question: How can we define utility?

**Maximizing the Information Gain**

- Pick the action $a$ that maximizes the information gain given a map $m$
  
  $$a^* = \arg \max_{a \in A} IG(m, a)$$

**Example: Binary Random Variable**

- Binary random variable $X \in \{0, 1\}$
- Probability distribution $P(X = 1) = p$
- How many bits do we need to transmit one sample of $p(X)$?
  - For $p=0$?
  - For $p=0.5$?
  - For $p=1$?
Example: Binary Random Variable

- Binary random variable \( X \in \{0,1\} \)
- Probability distribution \( P(X = 1) = p \)
- How many bits do we need to transmit one sample of \( p(X) \)?
- Answer:

Information Theory

- **Information gain** = Uncertainty reduction
  \[ IG(X, Y) = H(X) - H(X \mid Y) \]
- **Conditional entropy**
  \[ H(X \mid Y) = \sum_{i,j} p(x_i, y_j) \log \frac{p(y_j)}{p(x_i, y_j)} \]

Example: Map Entropy

Maximizing the Information Gain

- To compute the information gain one needs to know the observations obtained when carrying out an action
  \[ a^* = \arg \max_{a \in A} IG(m, a) \]
- This quantity is not known! Reason about potential measurements
  \[ a^* = \arg \max_{a \in A} \int IG(m, z)p(z \mid a)dz \]

Exploration Costs

- So far, we did not consider the cost of executing an action (e.g., time, energy, ...)
- **Utility** = uncertainty reduction \(-\) cost
- Select the action with the highest expected utility
  \[ a^* = \arg \max_{a \in A} IG(m, a) - \alpha \cdot E(cost(m, a)) \]
Exploration

- For each location \( <x,y> \)
  - Estimate the number of cells robot can sense (e.g., simulate laser beams using current map)
  - Estimate the cost of getting there

Exploration Actions

- So far, we only considered reduction in map uncertainty
- In general, there are many sources of uncertainty that can be reduced by exploration
  - Map uncertainty (visit unexplored areas)
  - Trajectory uncertainty (loop closing)
  - Localization uncertainty (active re-localization by re-visiting known locations)

Example: Active Loop Closing
[Stachniss et al., 2005]

- Reduce map uncertainty
- Reduce map + path uncertainty

Example: Active Loop Closing
[Stachniss et al., 2005]

- Entropy evolution
**Example: Reduce uncertainty in map, path, and pose** [Stachniss et al., 2005]

The decision-theoretic approach leads to intuitive behaviors: “re-localize before getting lost”

Some animals show a similar behavior (dogs marooned in the tundra of north Russia)

**Multi-Robot Exploration**

*Given:* Team of robots with communication  
*Goal:* Explore the environment as fast as possible

**Complexity**

- Single-robot exploration in known, graph-like environments is in general NP-hard
- Proof: Reduce traveling salesman problem to exploration
- Complexity of multi-robot exploration is exponential in the number of robots

**Motivation: Why Coordinate?**

- Without coordination, two robots might choose the same exploration frontier

**Levels of Coordination**

1. **No exchange of information**
2. **Implicit coordination:** Sharing a joint map  
   - Communication of the individual maps and poses  
   - Central mapping system
3. **Explicit coordination:** Determine better target locations to distribute the robots  
   - Central planner for target point assignment  
   - Minimize expected path cost / information gain / …
**Typical Trajectories**

Implicit coordination: 
![Typical Trajectories Implicit]

Explicit coordination: 
![Typical Trajectories Explicit]

**Coordination Algorithm**

In each time step:
- Determine set of exploration targets 
  \( S = \{s_1, \ldots, s_n\} \)
- Compute for each robot \( i \) and each target \( j \) the expected cost/utility \( C_{ij} \)
- Assign robots to targets using the **Hungarian algorithm**

**Hungarian Algorithm**

- Combinatorial optimization algorithm
- Solves the assignment problem in polynomial time \( O(n^3) \)
- General idea: Algorithm modifies the cost matrix until there is zero cost assignment

**Hungarian Algorithm: Example**

<table>
<thead>
<tr>
<th>targets</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

1. Compute the cost matrix (non-negative)

**Exploration Time**

[Stachniss et al., 2006]

![Exploration Time Graph]

**Exploration Time**

- Time for all robots to visit each target
- Uncoordinated vs. coordinated exploration
- Polynomial time complexity

**Hungarian Algorithm: Example**

<table>
<thead>
<tr>
<th>targets</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

2. Find minimum element in each row
Hungarian Algorithm: Example

3. Subtract minimum from each row element

4. Find minimum element in each column

5. Subtract minimum from each column element

6a. Assign (if possible)

6b. If no assignment is possible:
   - Connect all 0's by lines
   - Find the minimum in all remaining elements and subtract
   - Repeat step 2 – 6

If there are not enough targets:
Copy targets to allow multiple assignments
Example: Segmentation-based Exploration [Wurm et al., IROS 2008]

- Two-layer hierarchical role assignments using Hungarian algorithm (1: rooms, 2: targets in room)
- Reduces exploration time and risk of interferences

Summary: Exploration

- Exploration aims at generating robot motions so that an optimal map is obtained
- Coordination reduces exploration time
- Hungarian algorithm efficiently solves the assignment problem (centralized, 1-step lookahead)
- Challenges (active research):
  - Limited bandwidth and unreliable communication
  - Decentralized planning and task assignment

Coverage Path Planning

- Given: Known environment with obstacles
- Wanted: The shortest trajectory that ensures complete (sensor) coverage

Coverage Path Planning: Applications

- For flying robots
  - Search and rescue
  - Area surveillance
  - Environmental inspection
  - Inspection of buildings (bridges)
- For service robots
  - Lawn mowing
  - Vacuum cleaning
- For manipulation robots
  - Painting
  - Automated farming

Coverage Path Planning

- What is a good coverage strategy?
- What would be a good cost function?
Coverage Path Planning

- What is a good coverage strategy?
- What would be a good cost function?
  - Amount of redundant traversals
  - Number of stops and rotations
  - Execution time
  - Energy consumption
  - Robustness
  - Probability of success
  - ...

Related to the traveling salesman problem (TSP):
“Given a weighted graph, compute a path that visits every vertex once”
- In general NP-complete
- Many approximations exist
- Many approximate (and exact) solvers exist

Coverage of Simple Shapes

- Approximately optimal solution often easy to compute for simple shapes (e.g., trapezoids)

Idea
[Mannadiar and Rekleitis, ICRA 2011]
Coverage Based On Cell Decomposition
[Mannadiar and Rekleitis, ICRA 2011]

Approach:
1. Decompose map into “simple” cells
2. Compute connectivity between cells and build graph
3. Solve coverage problem on reduced graph

Step 1: Boustrophedon Cellular Decomposition
[Mannadiar and Rekleitis, ICRA 2011]
- Similar to trapezoidal decomposition
- Can be computed efficiently

Step 2: Build Reeb Graph
[Mannadiar and Rekleitis, ICRA 2011]
- Vertices = Critical points (that triggered the split)
- Edges = Connectivity between critical points

Step 3: Compute Euler Tour
[Mannadiar and Rekleitis, ICRA 2011]
- Extend graph so that vertices have even order
- Compute Euler tour (linear time)

Resulting Coverage Plan
[Mannadiar and Rekleitis, ICRA 2011]
- Follow the Euler tour
- Use simple coverage strategy for cells
- Note: Cells are visited once or twice

Robotic Cleaning of 3D Surfaces
[Hess et al., IROS 2012]
- Goal: Cover entire surface while minimizing trajectory length in configuration space
- Approach:
  - Discretize 3D environment into patches
  - Build a neighborhood graph
  - Formulate the problem as generalized TSP (GTSP)
Robotic Cleaning of 3D Surfaces
[Hess et al., IROS 2012]

Lessons Learned Today

- How to generate plans that are robust to uncertainty in sensing and locomotion
- How to explore an unknown environment
  - With a single robot
  - With a team of robots
- How to generate plans that fully cover known environments

Video: SFLY Final Project Demo (2012)

sFly
Swarm of Micro Flying Robots
http://www.sfly.org/
Course Evaluation

- Much positive feedback – thank you!!!
- We are also very happy with you as a group. Everybody seemed to be highly motivated!
- Suggestions for improvements (from course evaluation forms)
  - Workload was considered a bit too high → ECTS have been adjusted to 6 credits
  - ROS introduction lab course would be helpful → Will do this next time
- Any further suggestions/comments?

Scientific Research – General Idea

1. Observe phenomena
2. Formulate explanations and theories
3. Test them

Scientific Research in Student Projects

- How can you get involved in scientific research during your study?
Scientific Research in Student Projects

- How can you get involved in scientific research during your study?
  - Bachelor lab course (10 ECTS)
  - Bachelor thesis (15 ECTS)
  - Graduate lab course (10 ECTS)
  - Interdisciplinary project (16 ECTS)
  - Master thesis (30 ECTS)
  - Student research assistant (10 EUR/hour, typically 10 hours/week)

Step 1: Generate the Idea

- Be creative
- Follow your interests / preferences

Examples:
- Research question
- Challenging problem
- Relevant application
- Promising method (e.g., try to transfer method from another field)

Step 1b: Find related work

- There is always related work
- Find related research papers
  - Use Google scholar, paper repositories, ...
  - Navigate the citation network
  - Read survey articles
- Browse through (recent) text books
- Ask your professor, colleagues, ...
- It’s very unlikely that somebody else has already perfectly solved exactly your problem, so don’t worry! Technology evolves very fast...

Step 2: Develop a Solution

- Practitioner
  - Start programming
  - Realize that it is not going to work, start over, ...
  - When it works, formalize it (try to find out why it works and what was missing before)
  - Empirically verify that it works
- Theorist
  - Formalize the problem
  - Find suitable method
  - (Theoretically) prove that it is right
  - (If needed) implement a proof-of-concept

Step 3: Validation

- What are your claims?
- How can you prove them?
  - Theoretical proof (mathematical problem)
  - Experimental validation
    - Qualitative (e.g., video)
    - Quantitative (e.g., many trials, statistical significance)
  - Compare and discuss your results with respect to previous work/approaches

Step 4: Dissemination

- Good solution/expertise alone is not enough
- You need to convince other people in the field

Usual procedure:
1. Write research paper (usually 6-8 pages)  
2. Submit PDF to an international conference or journal  
3. Paper will be peer-reviewed
4. Improve paper (if necessary)
5. Give talk or poster presentation at conference
6. Optionally: Repeat step 1-5 until PhD 😊
Step 5: Refinement

- Discuss your work with
  - Your colleagues
  - Your professor
  - Other colleagues at conferences
- Improve your approach and evaluation
  - Adopt notation to the standard
  - Get additional references/insights
  - Conduct more/additional experiments
- Simplify and generalize your approach
- Collaborate with other people (in other fields)

What are the desired properties of a good scientific experiment?

- Reproducibility / repeatability
  - Document the experimental setup
  - Choose (and motivate) an evaluation criterion
- Experiments should allow you to validate/falsify competing hypotheses

Current trends:

- Make data available for review and criticism
- Same for software (open source)

Challenges

- Reproducibility is sometimes not easy to guarantee
- Any ideas why?

- Randomized components/noise (beat with the law of large numbers/statistical tests)
- Experiment requires special hardware
  - Self-built, unique robot
  - Expensive lab equipment
  - …
- Experiments cost time
- “(Video) Demonstrations will suffice”
- Technology changes fast
Benchmarks

- Effective and affordable way of conducting experiments
- Sample of a task domain
- Well-defined performance measurements
- Widely used in computer vision and robotics
- Which benchmark problems do you know?

Example Benchmark Problems

Computer Vision

- Middlebury datasets (optical flow, stereo, ...)
- Caltech-101, PASCAL (object recognition)
- Stanford bunny (3d reconstruction)

Robotics

- RoboCup competitions (robotic soccer)
- DARPA challenges (autonomous car)
- SLAM datasets

Image Denoising: Lenna Image

- 512x512 pixel standard image for image compression and denoising
- Lena Söderberg, Playboy magazine Nov. 1972
- Scanned by Alex Sawchuck at USC in a hurry for a conference paper

http://www.cs.cmu.edu/~chuck/lennapg/

Object Recognition: Caltech-101

- Pictures of objects belonging to 101 categories
- About 40-800 images per category
- Recognition, classification, categorization

RoboCup Initiative

- Evaluation of full system performance
- Includes perception, planning, control, ...
- Easy to understand, high publicity
- “By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, complying with the official rule of the FIFA, against the winner of the most recent World Cup.”

RoboCup Initiative
SLAM Evaluation

- Intel dataset: laser + odometry [Haehnel, 2004]
- New College dataset: stereo + omni-directional vision + laser + IMU [Smith et al., 2009]
- TUM RGB-D dataset [Sturm et al., 2011/12]
- ...

TUM RGB-D Dataset
[Sturm et al., RSS RGB-D 2011; Sturm et al., IROS 2012]

- RGB-D dataset with ground truth for SLAM evaluation
- Two error metrics proposed (relative and absolute error)
- Online + offline evaluation tools
- Training datasets (fully available)
- Validation datasets (ground truth not publicly available to avoid overfitting)

Recorded Scenes

- Various scenes (handheld/robot-mounted, office, industrial hall, dynamic objects, ...)
- Large variations in camera speed, camera motion, illumination, environment size, ...

Dataset Acquisition

- Motion capture system
  - Camera pose (100 Hz)
- Microsoft Kinect
  - Color images (30 Hz)
  - Depth maps (30 Hz)
  - IMU (500 Hz)
- External video camera (for documentation)

Motion Capture System

- 9 high-speed cameras mounted in room
- Cameras have active illumination and pre-process image (thresholding)
- Cameras track positions of retro-reflective markers

Calibration

Calibration of the overall system is not trivial:
1. Mocap calibration
2. Kinect-mocap calibration
3. Time synchronization
Calibration Step 1: Mocap

- Need at least 2 cameras for position fix
- Need at least 3 markers on object for full pose
- Calibration stick for extrinsic calibration

Example: Raw Image from Mocap

Example: Tracked Object (4 Markers)

Example: Position Triangulation of a Single Marker

Example: Recorded Trajectory
Calibration Step 2: Mocap-Kinect
- Need to find transformation between the markers on the Kinect and the optical center
- Special calibration board visible both by Kinect and mocap system (manually gauged)

Calibration Step 3: Time Synchronization
- Assume a constant time delay between mocap and Kinect messages
- Choose time delay that minimizes reprojection error during checkerboard calibration

Calibration - Validation
- Intrinsic calibration
- Extrinsic calibration color + depth
- Time synchronization color + depth
- Mocap system slowly drifts (need re-calibration every hour)
- Validation experiments to check the quality of calibration
  - 2mm length error on 2m rod across mocap volume
  - 4mm RMSE on checkerboard sequence

Example Sequence: Freiburg1/XYZ
Sequence description (on the website):
“For this sequence, the Kinect was pointed at a typical desk in an office environment. This sequence contains only translatory motions along the principal axes of the Kinect, while the orientation was kept (mostly) fixed. This sequence is well suited for debugging purposes, as it is very simple.”
**What Is a Good Evaluation Metric?**

- Compare camera trajectories
  - Ground truth trajectory \( Q_1, \ldots, Q_n \in \text{SE(3)} \)
  - Estimate camera trajectory \( P_1, \ldots, P_n \in \text{SE(3)} \)
- Two common evaluation metrics
  - Relative pose error (drift per second)
  - Absolute trajectory error (global consistency)

**Relative Pose Error (RPE)**

- Measures the (relative) drift
- Recommended for the evaluation of visual odometry approaches

\[
E_i := \left( Q_i^{-1} Q_{i+\Delta} \right)^{-1} \left( P_i^{-1} P_{i+\Delta} \right)
\]

**Absolute Trajectory Error (ATE)**

- Measures the global error
- Requires pre-aligned trajectories
- Recommended for SLAM evaluation

\[
E_i := Q_i^{-1} S P_i
\]

**Evaluation metrics**

- Average over all time steps
  \[
  \text{RMSE}(E_{1:n}) := \left( \frac{1}{m} \sum_{i=1}^{m} \| \text{trans}(E_i) \|^2 \right)^{1/2}
  \]
- Reference implementations for both evaluation metrics available
- Output: RMSE, Mean, Median (as text)
- Plot (png/pdf, optional)
Summary – TUM RGB-D Benchmark

- Dataset for the evaluation of RGB-D SLAM systems
- Ground-truth camera poses
- Evaluation metrics + tools available

Discussion on Benchmarks

Pro:
- Provide objective measure
- Simplify empirical evaluation
- Stimulate comparison

Con:
- Introduce bias towards approaches that perform well on the benchmark (overfitting)
- Evaluation metrics are not unique (many alternative metrics exist, choice is subjective)

Three Phases of Evolution in Research

- Novel research problem appears (e.g., market launch of Kinect, quadrocopters, ...)
  - Is it possible to do something at all?
  - Proof-of-concept, qualitative evaluation
- Consolidation
  - Problem is formalized
  - Alternative approaches appear
  - Need for quantitative evaluation and comparison
- Settled
  - Benchmarks appear
  - Solid scientific analysis, text books, ...

Final Exam

- Oral exam in teams (2-3 students)
- At least 15 minutes per student → individual grades
- Questions will address
  - Your project
  - Material from the exercise sheets
  - Material from the lecture
Exercise Sheet 6

- Prepare final presentation
- Proposed structure: 4-5 slides
  1. Title slide with names + motivating picture
  2. Approach
  3. Results (video is a plus)
  4. Conclusions (what did you learn in the project?)
  5. Optional: Future work, possible extensions
- Hand in slides before Tue, July 17, 10am (!)