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

Computer Vision Group
Prof. Daniel Cremers

TUM Technische Universität München

Visual Navigation for Flying Robots

Welcome

Dr. Jürgen Sturm

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

Who are we?

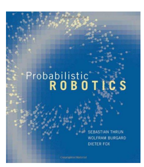
- Computer Vision group:
  1 Professor, 2 Postdocs, 7 PhD students
- Research topics:
  Optical flow and motion estimation, 3D reconstruction, image segmentation, convex optimization
- My research goal:
  Apply solutions from computer vision to real-world problems in robotics.

Goal of this Course

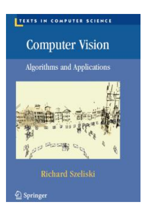
- Provide an overview on problems/approaches for autonomous quadrocopters
- Strong focus on vision as the main sensor
- Areas covered: Mobile Robotics and Computer Vision
- Hands-on experience in lab course

Course Material

- Probabilistic Robotics. Sebastian Thrun, Wolfram Burgard and Dieter Fox. MIT Press, 2005.



- Computer Vision: Algorithms and Applications. Richard Szeliski. Springer, 2010.
  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

Basics on mobile robotics (1–4)

Camera-based localization and mapping (5–9)

Advanced topics (10–12)

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

Team Name				
Student Name				
Student Name				
Student Name				

Team Name				
Student Name				
Student Name				
Student Name				

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)

	Red	Green	Blue
Thu 2pm – 3pm			
Thu 3pm – 4pm			
Thu 4pm – 5pm			

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

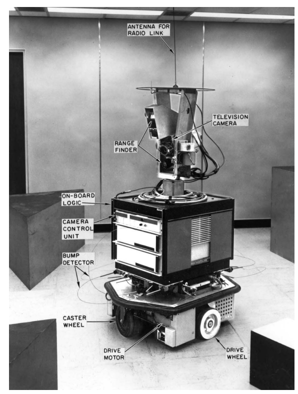
General background

- 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)

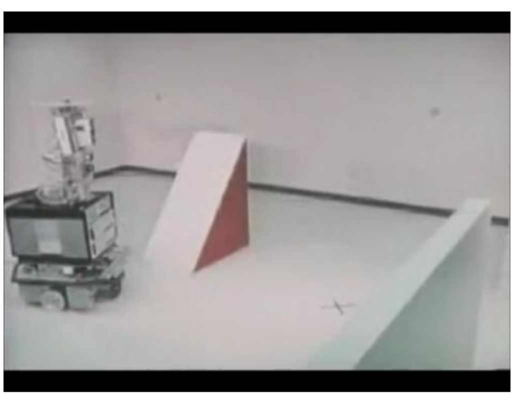
History

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)



Shakey the Robot (1966-1972)



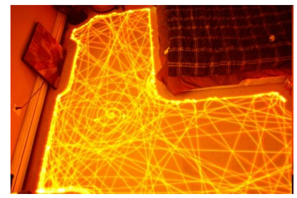
Stanford Cart (1961-80)

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)

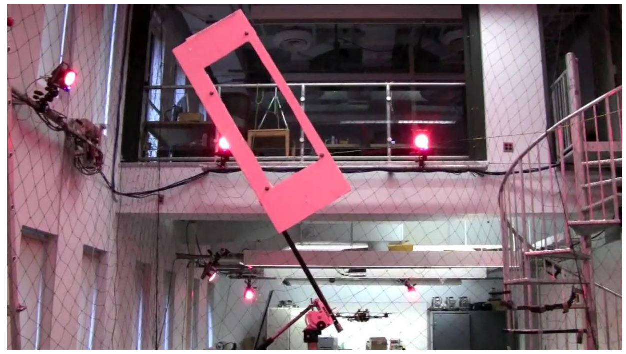
Kiva Robotics (2007)

- Pick, pack and ship automation

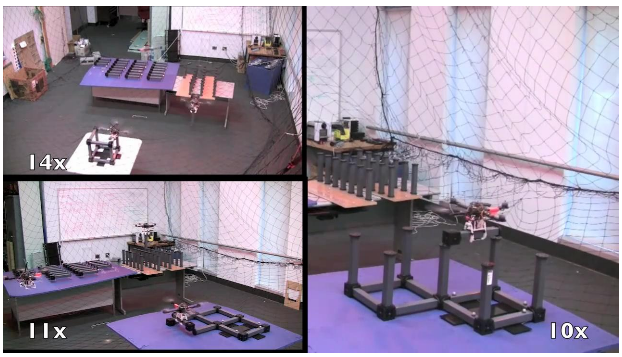
Fork Lift Robots (2010)

Quadrocopters (2001-)

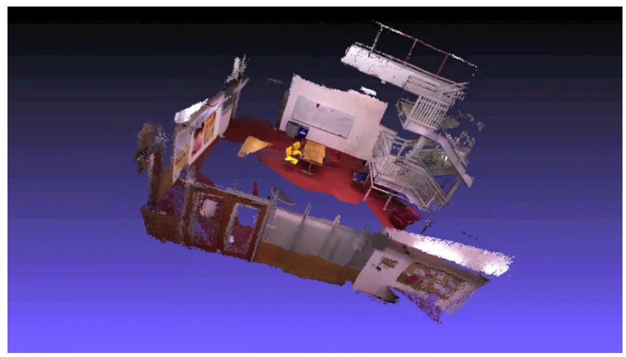
Aggressive Maneuvers (2010)



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 **Used in the lab course**
  - …
- Community/open-source projects
  - Mikrokopter
  - Paparazzi
  - …

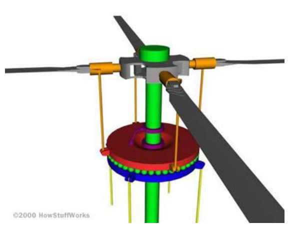
For more, see http://multicopter.org/wiki/Multicopter_Table

Flying Principles

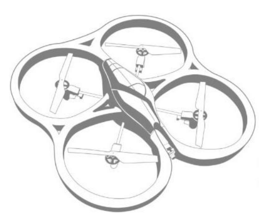
- Fixed-wing airplanes
  - generate lift through forward airspeed and the shape of the wings
  - controlled by flaps
- Helicopters/rotorcrafts
  - main rotor for lift, tail rotor to compensate for torque
  - controlled by adjusting rotor pitch
- Quadrocopter/quadrotor
  - four rotors generate lift
  - controlled by changing the speeds of rotation

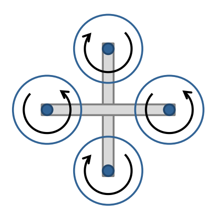
Helicopter

- Swash plate adjusts pitch of propeller cyclically, controls pitch and roll
- Yaw is controlled by tail rotor



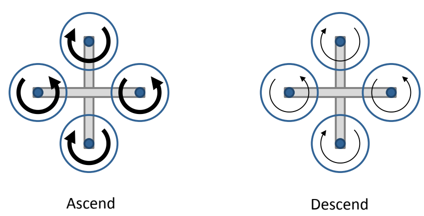
Quadrocopter



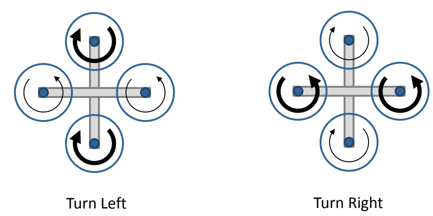


Keep position:
- Torques of all four rotors sum to zero
- Thrust compensates for earth gravity

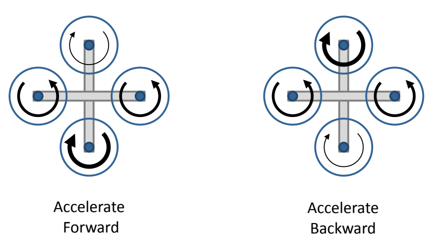
Quadrocopter: Basic Motions



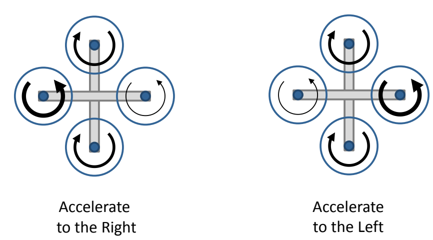
Quadrocopter: Basic Motions



Quadrocopter: Basic Motions



Quadrocopter: Basic Motions



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?

Robot Ethics

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?

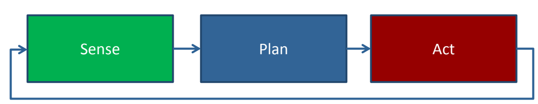
Robot Hardware/Components

- 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

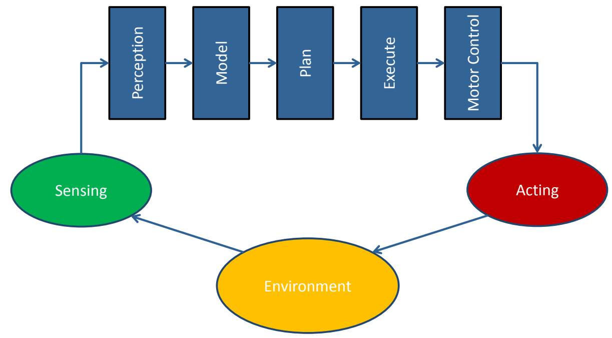


- Inspired by methods from Artificial Intelligence (70’s)
- Focus on automated reasoning and knowledge representation
- STRIPS (Stanford Research Institute Problem Solver): Perfect world model, closed world assumption
- Shakey: Find boxes and move them to designated positions

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

Classical paradigm as horizontal/functional decomposition

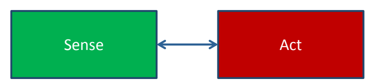


Characteristics of hierarchical paradigm

Good old-fashioned Artificial Intelligence (GOFAI):

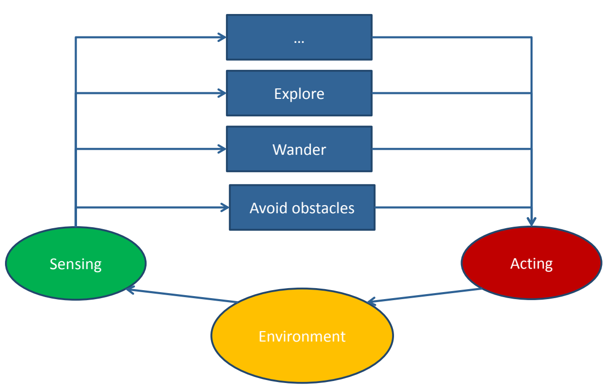
- Symbolic approaches
- Robot perceives the world, plans the next action, acts
- All data is inserted into a single, global world model
- Sequential data processing

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

Reactive Paradigm as Vertical Decomposition



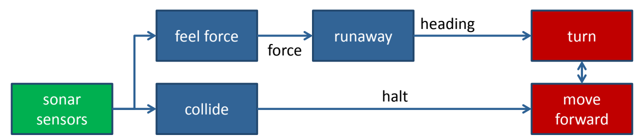
Characteristics of Reactive Paradigm

- 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)

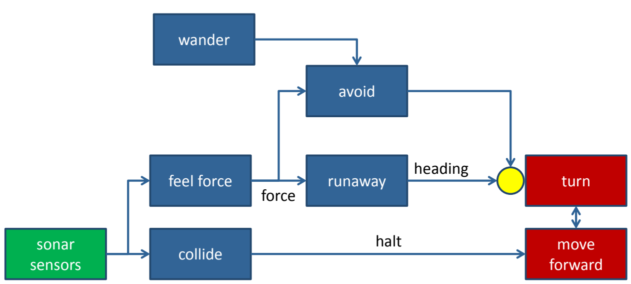
Subsumption Architecture

- 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

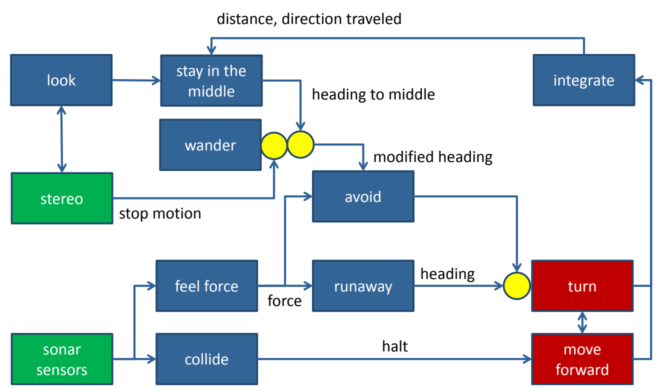
Level 1: Avoid



Level 2: Wander



Level 3: Follow Corridor



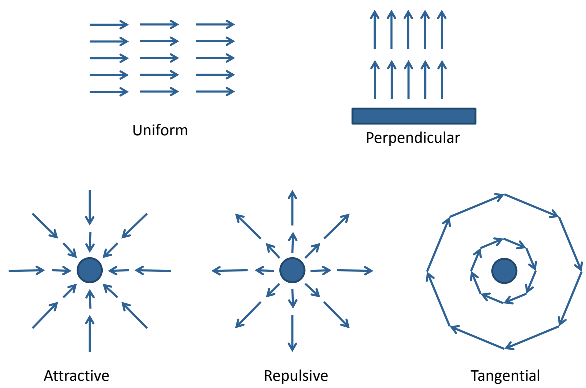
Roomba Robot

- Exercise: Model the behavior of a Roomba robot.

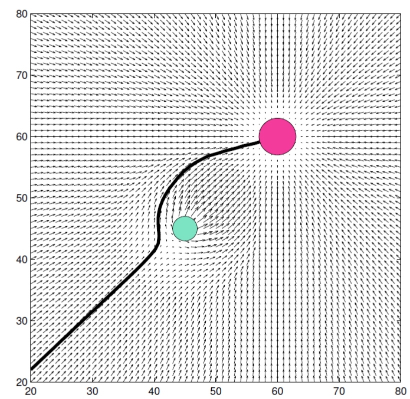
Navigation with Potential Fields

- 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



Example: reach goal and avoid obstacles

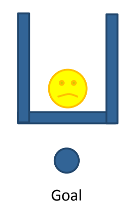


Corridor Following Robot

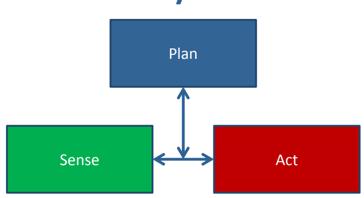
- Level 1 (collision avoidance)
  add repulsive fields for the detected obstacles
- Level 2 (wander)
  add a uniform field into a (random) direction
- Level 3 (corridor following)
  replaces the wander field by three fields (two perpendicular, one parallel to the walls)

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

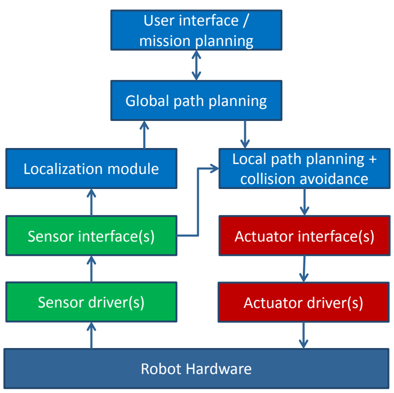
Best Practices for Robot Architectures

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

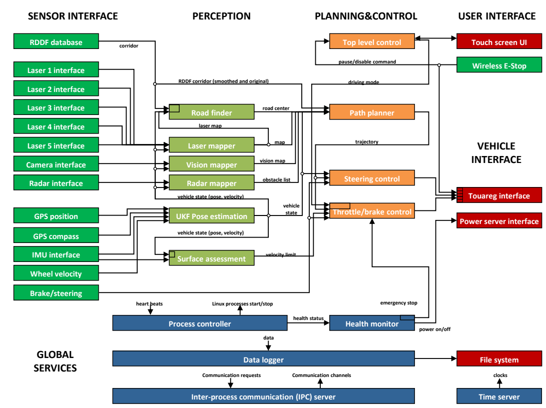
Robotic Middleware

- 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

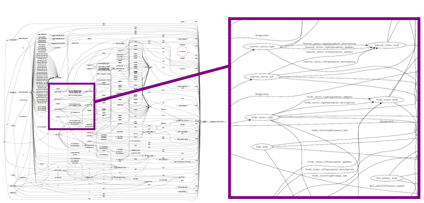


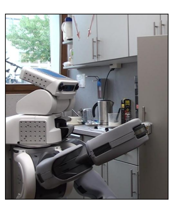
Stanley's Software Architecture



PR2 Software Architecture

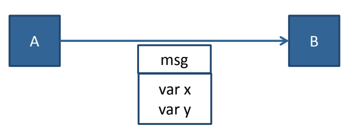
- Two 7-DOF arms, grippers, torso, 2-DOF head
- 7 cameras, 2 laser scanners
- Two 8-core CPUs, 3 network switches
- 73 nodes, 328 message topics, 174 services



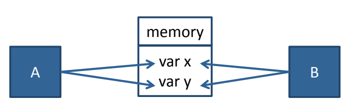


Communication Paradigms

- Message-based communication



- Direct (shared) memory access



Forms of Communication

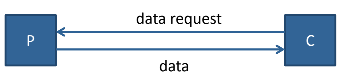
- Push
- Pull
- Publisher/subscriber
- Publish to blackboard
- Remote procedure calls / service calls
- Preemptive tasks / actions

Push

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

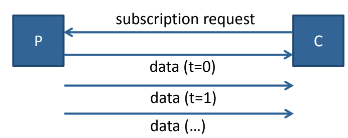
Pull

- 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



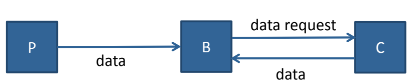
Publisher/Subscriber

- 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, …)



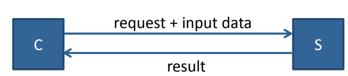
Publish to Blackboard

- 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/
- 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

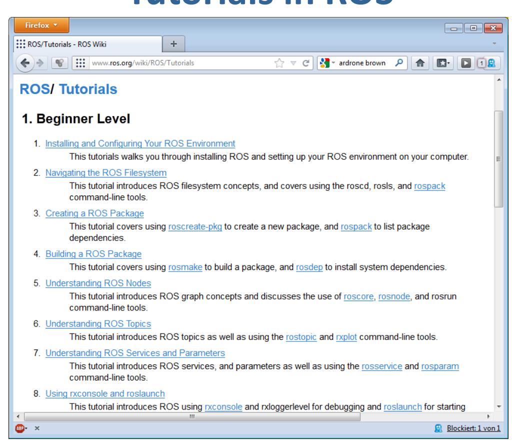
Software Management

- 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



Exercise Sheet 1

- On the course website
- Solutions are due in 2 weeks (May 1st)

- Theory part:
  Define the motion model of a quadrocopter
  (will be covered next week)
- Practical part:
  Playback a bag file with data from
  quadrocopter & plot trajectory

Summary

- History of mobile robotics
- Brief intro on quadrocopters
- Paradigms in robotics
- Architectures and middleware

Questions?

- See you next week!

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

3D Geometry and Sensors

Dr. Jürgen Sturm

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

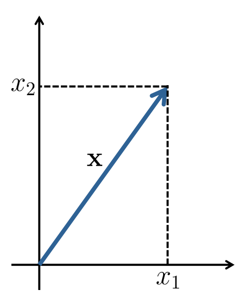
- Linear algebra
- 2D and 3D geometry
- Sensors

Vectors

- Vector and its coordinates

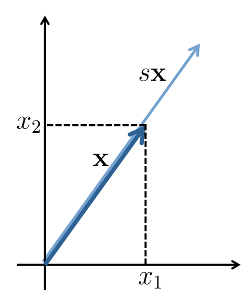
$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

- Vectors represent points in an n-dimensional space



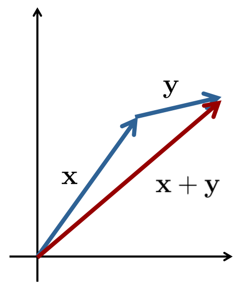
Vector Operations

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



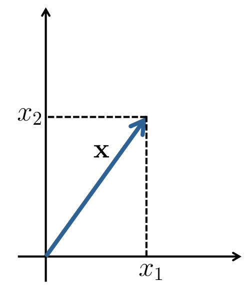
Vector Operations

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



Vector Operations

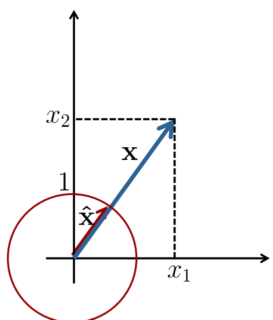
- Scalar multiplication
- Addition/subtraction
- **Length**
- Normalized vector
- Dot product
- Cross product



$$\|x\|_2 = \|x\| = \sqrt{x_1^2 + x_2^2 + \dots}$$

Vector Operations

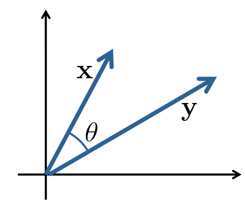
- Scalar multiplication
- Addition/subtraction
- Length
- **Normalized vector**
- Dot product
- Cross product



$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$$

Vector Operations

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



$$\mathbf{x}\cdot\mathbf{y} = \|\mathbf{x}\|\|\mathbf{y}\|\cos\theta$$

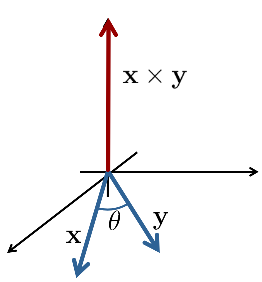
$\mathbf{x}, \mathbf{y}$ are orthogonal if $\mathbf{x}\cdot\mathbf{y} = 0$

$\mathbf{y}$ is linearly dependent from $\{\mathbf{x}_1, \mathbf{x}_2, \dots\}$ if

$$\mathbf{y} = \sum_i k_i \mathbf{x}_i$$

Vector Operations

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



$$\mathbf{x}\times\mathbf{y} = \|\mathbf{x}\|\|\mathbf{y}\|\sin(\theta)\mathbf{n}$$

Cross Product

- Definition

$$\mathbf{x}\times\mathbf{y} = \begin{pmatrix} x_2y_3 - x_3y_2 \\ x_3y_1 - x_1y_3 \\ x_1y_2 - x_2y_1 \end{pmatrix}$$

- Matrix notation for the cross product

$$[\mathbf{x}]_\times = \begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix}$$

- Verify that $\mathbf{x}\times\mathbf{y} = [\mathbf{x}]_\times \mathbf{y}$

Matrices

- Rectangular array of numbers

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & & & \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{pmatrix} \in \mathbb{R}^{\overset{\text{rows}}{n} \times \overset{\text{columns}}{m}}$$

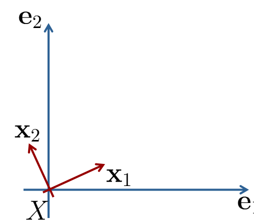
- First index refers to row
- Second index refers to column

Matrices

- Column vectors of a matrix

$$X = \begin{pmatrix} \boxed{\begin{matrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{matrix}} & \boxed{\begin{matrix} x_{12} \\ x_{22} \\ \\ x_{n2} \end{matrix}} & \begin{matrix} \dots \\ \dots \\ \\ \dots \end{matrix} & \boxed{\begin{matrix} x_{1m} \\ x_{2m} \\ \\ x_{nm} \end{matrix}} \end{pmatrix}$$

$$= \begin{pmatrix} \mathbf{x}_{*1} & \mathbf{x}_{*2} & \dots & \mathbf{x}_{*m} \end{pmatrix}$$



- Geometric interpretation: for example, column vectors can form basis of a coordinate system

Matrices

- Row vectors of a matrix

$$X = \begin{pmatrix} \boxed{x_{11} \quad x_{12} \quad \dots \quad x_{1m}} \\ \boxed{x_{21} \quad x_{22} \quad \dots \quad x_{2m}} \\ \vdots \\ \boxed{x_{n1} \quad x_{n2} \quad \dots \quad x_{nm}} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_{1*}^\top \\ \mathbf{x}_{2*}^\top \\ \vdots \\ \mathbf{x}_{n*}^\top \end{pmatrix}$$

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

Matrices

- Square matrix
- Diagonal matrix
- Upper and lower triangular matrix
- Symmetric matrix $X = X^\top$
- Skew-symmetric matrix $X = -X^\top \left(= \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}\right)$
- (Semi-)positive definite matrix $\mathbf{a}^\top X \mathbf{a} \geq 0$
- Invertible matrix
- Orthonormal matrix
- Matrix rank

Matrix Operations

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

Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- **Matrix-vector multiplication** $X\mathbf{b}$
- Matrix-matrix multiplication
- Inversion

Matrix-Vector Multiplication

- Definition

$$X \cdot \mathbf{b} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & & & \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} = \sum_{k=1}^{n} \mathbf{x}_{*k} \cdot b_k$$

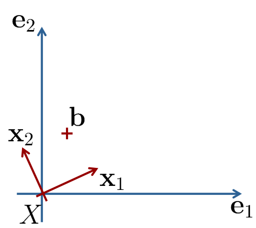
column vectors

- Geometric interpretation:
  a linear combination of the columns of X scaled by the coefficients of **b**

Matrix-Vector Multiplication

$$X \cdot \mathbf{b} = \sum_{k=1}^{n} \mathbf{x}_{*k} \cdot b_k$$

column vectors



- Geometric interpretation:
  A linear combination of the columns of A scaled by the coefficients of **b**
  → coordinate transformation

Matrix Operations

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

Matrix-Matrix Multiplication

- Operator $\mathbb{R}^{n \times m} \times \mathbb{R}^{m \times p} \to \mathbb{R}^{n \times p}$
- Definition

$$\begin{aligned} C &= AB \\ &= A \left(\mathbf{b}_{*1} \quad \mathbf{b}_{*2} \quad \cdots \mathbf{b}_{*p} \right) \end{aligned}$$

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

Matrix-Matrix Multiplication

- Not commutative (in general)

$$AB \neq BA$$

- Associative

$$A(BC) = (AB)C$$

- Transpose

$$(AB)^\top = B^\top A^\top$$

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 $\bar{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \in \mathbb{R}^3$
- Homogeneous coordinates $\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^2$

Geometric Primitives in 2D

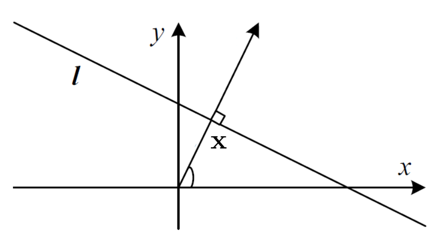
- Homogeneous vectors that differ only be scale represent the same 2D point
- Convert back to inhomogeneous coordinates by dividing through last element

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \\ 1 \end{pmatrix} = \tilde{w} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \tilde{w}\bar{\mathbf{x}}$$

- Points with $\tilde{w} = 0$ are called points at infinity or ideal points

Geometric Primitives in 2D

- 2D line $\tilde{\mathbf{l}} = (a, b, c)^\top$
- 2D line equation $\bar{\mathbf{x}} \cdot \tilde{\mathbf{l}} = ax + by + c = 0$

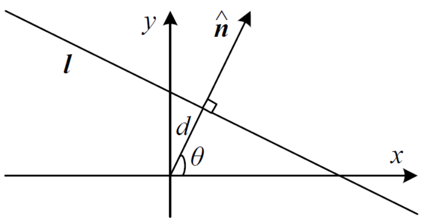


Geometric Primitives in 2D

- Normalized line equation vector

$$\tilde{\mathbf{l}} = (\hat{n}_x, \hat{n}_y, d)^\top = (\hat{\mathbf{n}}, d)^\top \quad \text{with} \quad \|\hat{\mathbf{n}}\| = 1$$

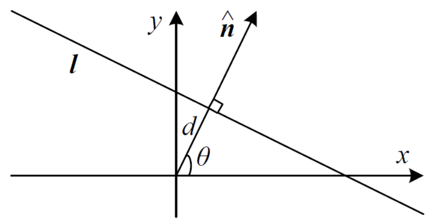
  where d is the distance of the line to the origin



Geometric Primitives in 2D

- Polar coordinates of a line: $(\theta, d)^\top$
  (e.g., used in Hough transform for finding lines)

$$\hat{\mathbf{n}} = (\cos\theta, \sin\theta)^\top$$



Geometric Primitives in 2D

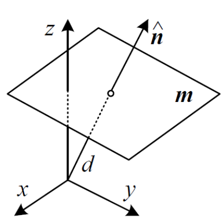
- Line joining two points $\quad \tilde{\mathbf{l}} = \tilde{\mathbf{x}}_1 \times \tilde{\mathbf{x}}_2$
- Intersection point of two lines $\quad \tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$

Geometric Primitives in 3D

- 3D point
  (same as before) $\quad \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3$
- Augmented vector $\quad \bar{\mathbf{x}} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \in \mathbb{R}^4$
- Homogeneous coordinates $\quad \tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^3$

Geometric Primitives in 3D

- 3D plane $\quad \tilde{\mathbf{m}} = (a, b, c, d)^\top$
- 3D plane equation $\quad \bar{\mathbf{x}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$
- Normalized plane
  with unit normal vector
  $\mathbf{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d)^\top = (\hat{\mathbf{n}}, d)$
  ($\|\hat{\mathbf{n}}\| = 1$)
  and distance d

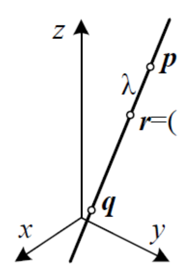


Geometric Primitives in 3D

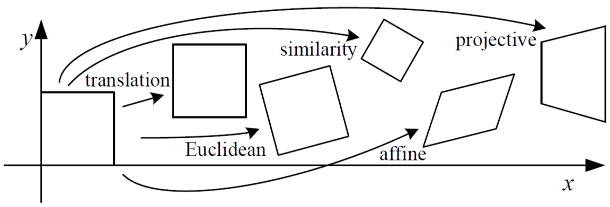
- 3D line $\mathbf{r} = (1 - \lambda)\mathbf{p} + \lambda\mathbf{q}$
  through points $\mathbf{p}, \mathbf{q}$
- Infinite line:

  $\lambda \in \mathbb{R}$
- Line segment joining $\mathbf{p}, \mathbf{q}$:

  $0 \le \lambda \le 1$



2D Planar Transformations



2D Transformations

- Translation $\quad \mathbf{x}' = \mathbf{x} + t$

$$\mathbf{x}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \end{pmatrix}}_{2\times 3} \bar{\mathbf{x}}$$

$$\bar{\mathbf{x}}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}}_{3\times 3} \bar{\mathbf{x}}$$

where $\mathbf{I}$ is the identity matrix (2x2) and $\mathbf{0}$ is the zero vector

2D Transformations

- Rotation + translation (2D rigid body motion, or 2D Euclidean transformation)

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + t \qquad \text{or} \qquad \bar{\mathbf{x}}' = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \bar{\mathbf{x}}$$

where $\mathbf{R} = \begin{pmatrix} \cos\theta & -sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$

is an orthonormal rotation matrix, i.e., $\mathbf{R}\mathbf{R}^\top = \mathbf{I}$

- Distances (and angles) are preserved

2D Transformations

- Scaled rotation/similarity transform

$$\mathbf{x}' = s\mathbf{R}\mathbf{x} + t \qquad \text{or} \qquad \bar{\mathbf{x}}' = \begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \bar{\mathbf{x}}$$

- Preserves angles between lines

2D Transformations

- Affine transform

$$\bar{\mathbf{x}}' = A\bar{\mathbf{x}} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \bar{\mathbf{x}}$$

- Parallel lines remain parallel

2D Transformations

- Projective/perspective transform

$$\tilde{\mathbf{x}}' = \tilde{H} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \tilde{\mathbf{x}}$$

- Note that $\tilde{H}$ is homogeneous (only defined up to scale)
- Resulting coordinates are homogeneous
- Parallel lines remain parallel

2D Transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

3D Transformations

- Translation

$$\bar{\mathbf{x}}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}}_{4\times 4} \bar{\mathbf{x}}$$

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

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

- Scaled rotation, affine transform, projective transform…

3D Transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	6	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	7	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{3\times 4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{4\times 4}$	15	straight lines	

3D Rotations

- Rotation matrix
  (also called the special orientation group SO(3))

- Euler angles
- Axis/angle
- Unit quaternion

Rotation Matrix

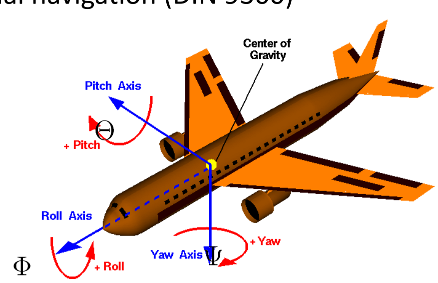
- Orthonormal 3x3 matrix

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

- Column vectors correspond to coordinate axes
- Special orientation group $R \in \mathrm{SO}(3)$
- 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)



Euler Angles

- Yaw Ψ, Pitch Θ, Roll Φ to rotation matrix

$$
\begin{aligned}
R &= R_Z(\Psi) R_Y(\Theta) R_X(\Phi) \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{pmatrix} \begin{pmatrix} \cos\Theta & 0 & -\sin\Theta \\ 0 & 1 & 0 \\ \sin\Theta & 0 & \cos\Theta \end{pmatrix} \begin{pmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos\Theta\cos\Psi & \cos\Theta\sin\Psi & -\sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi & \sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & \sin\Phi\cos\Theta \\ \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{pmatrix}
\end{aligned}
$$

- Rotation matrix to Yaw-Pitch-Roll

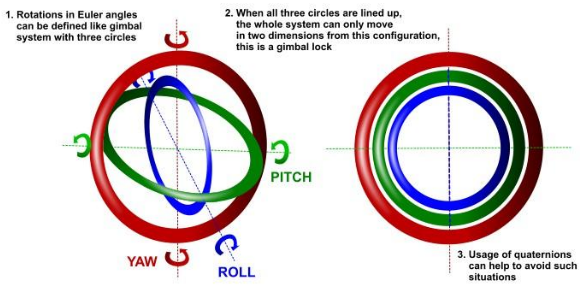
$$
\begin{aligned}
\phi &= \operatorname{Atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right) \\
\psi &= -\operatorname{Atan2}\left(\frac{r_{21}}{\cos(\phi)}, \frac{r_{11}}{\cos(\phi)}\right) \\
\theta &= \operatorname{Atan2}\left(\frac{r_{32}}{\cos(\phi)}, \frac{r_{33}}{\cos(\phi)}\right)
\end{aligned}
$$

Euler Angles

- Advantage:
  - Minimal representation (3 parameters)
  - Easy interpretation
- Disadvantages:
  - Many "alternative" Euler representations exist (XYZ, ZXZ, ZYX, …)
  - Singularities (gimbal lock)

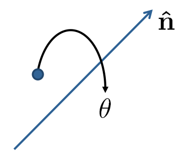
Gimbal Lock

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



Axis/Angle

- Represent rotation by
  - rotation axis $\hat{\mathbf{n}}$ and
  - rotation angle θ
- 4 parameters $(\hat{\mathbf{n}}, \theta)$
- 3 parameters $\boldsymbol{\omega} = \theta\hat{\mathbf{n}}$
  - 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)

$$R(t) \in \mathrm{SO}(3) \qquad t \in \mathbb{R}$$

- As this rotations are orthonormal matrices, we have $\quad R(t)R^\top(t) = I$
- Now take the derivative on both sides (w.r.t. *t*)

$$
\begin{aligned}
\dot{R}(t)R^\top(t) + R(t)\dot{R}^\top(t) &= 0 \\
\dot{R}(t)R^\top(t) &= -(\dot{R}(t)R^\top(t))^\top
\end{aligned}
$$

- Thus, $\dot{R}(t)R^\top(t)$ must be skew-symmetric, i.e.,

$$[\boldsymbol{\omega}(t)]_\times = \dot{R}(t)R^\top(t)$$

Derivation of Angular Velocities

→Linear ordinary differential equation (ODE)

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

- Solution of this ODE

$$R(t) = \exp([\boldsymbol{\omega}]_\times) R(0)$$

- Conversions

$$R = \exp([\boldsymbol{\omega}]_\times) \qquad [\boldsymbol{\omega}]_\times = \log R$$

Derivation of Angular Velocities

→Linear ordinary differential equation (ODE)

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

- The space of all skew-symmetric matrices is called the *tangent space*

$$\mathrm{so}(3) = \{[\boldsymbol{\omega}]_\times \in \mathbb{R}^{3\times 3} \mid \boldsymbol{\omega} \in \mathbb{R}^3\}$$

- Space of all rotations in 3D (Special orientation group)

$$\mathrm{SO}(3) = \{R \in \mathbb{R}^{3\times 3} \mid R^\top R = I, \det\, R = 1\}$$

Conversion

- Rodriguez' formula

$$R(\hat{\mathbf{n}}, \theta) = I + \sin\theta[\hat{\mathbf{n}}]_\times + (1 - \cos\theta)[\hat{\mathbf{n}}]_\times^2$$

- Inverse

$$\theta = \cos^{-1}\left(\frac{\mathrm{trace}(R) - 1}{2}\right), \hat{\mathbf{n}} = \frac{1}{2\sin\theta}\begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix}$$

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

Exponential Twist

- The exponential map can be generalized to Euclidean transformations (incl. translations)
- Tangent space $\mathrm{se}(3) = \mathrm{so}(3) \times \mathbb{R}^3$
- (Special) Euclidean group $\mathrm{SE}(3) = \mathrm{SO}(3) \times \mathbb{R}^3$ (group of all Euclidean transforms)
- Rigid body velocity

$$\xi = (\underbrace{\omega_x, \omega_y, \omega_z}_{\text{angular vel.}}, \underbrace{v_x, v_y, v_z}_{\text{linear vel.}}) \in \mathbb{R}^6$$

Exponential Twist

- Convert to homogeneous coordinates

$$\hat{\boldsymbol{\xi}} = \begin{pmatrix} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \in \mathrm{se}(3)$$

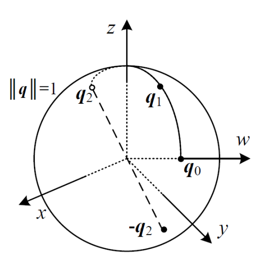
- Exponential map between se(3) and SE(3)

$$M = \exp\hat{\boldsymbol{\xi}} \qquad\qquad \hat{\xi} = \log M$$

- There are also direct formulas (similar to Rodriguez)

Unit Quaternions

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



Unit Quaternions

- Advantage: multiplication and inversion operations are really fast
- Quaternion-Quaternion Multiplication

$$\begin{aligned} \mathbf{q}_0\mathbf{q}_1 &= (\mathbf{v}_0, w_0)(\mathbf{v}_1, w_1) \\ &= (\mathbf{v}_0 \times \mathbf{v}_1 + w_0\mathbf{v}_1 + w_1\mathbf{v}_0, w_0w_1 - \mathbf{v}_0\mathbf{v}_1) \end{aligned}$$

- Inverse (flip sign of v or w)

$$\begin{aligned} \mathbf{q}_0/\mathbf{q}_1 &= (\mathbf{v}_0, w_0)/(\mathbf{v}_1, w_1) \\ &= (\mathbf{v}_0, w_0)(\mathbf{v}_1, -w_1) \\ &= (\mathbf{v}_0 \times \mathbf{v}_1 + w_0\mathbf{v}_1 - w_1\mathbf{v}_0, -w_0w_1 - \mathbf{v}_0\mathbf{v}_1) \end{aligned}$$

Unit Quaternions

- Quaternion-Vector multiplication (rotate point p with rotation q)

$$\mathbf{p}' = \mathbf{v}\bar{\mathbf{p}}/\mathbf{q}$$

with $\bar{\mathbf{p}} = (x, y, z, 0)^\top$

- Relation to Axis/Angle representation

$$\mathbf{q} = (\mathbf{v}, w) = (\sin\frac{\theta}{2}\hat{\mathbf{n}}, \cos\frac{\theta}{2})$$

Spherical Linear Interpolation (SLERP)

- Useful for interpolating between two rotations

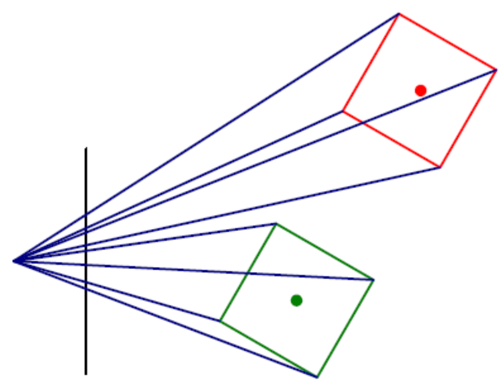
procedure *slerp*$(\boldsymbol{q}_0, \boldsymbol{q}_1, \alpha)$:

1. $\boldsymbol{q}_r = \boldsymbol{q}_1/\boldsymbol{q}_0 = (\boldsymbol{v}_r, w_r)$
2. if $w_r < 0$ then $\boldsymbol{q}_r \leftarrow -\boldsymbol{q}_r$
3. $\theta_r = 2\tan^{-1}(\|\boldsymbol{v}_r\|/w_r)$
4. $\hat{\boldsymbol{n}}_r = \mathcal{N}(\boldsymbol{v}_r) = \boldsymbol{v}_r/\|\boldsymbol{v}_r\|$
5. $\theta_\alpha = \alpha\,\theta_r$
6. $\boldsymbol{q}_\alpha = (\sin\frac{\theta_\alpha}{2}\hat{\boldsymbol{n}}_r, \cos\frac{\theta_\alpha}{2})$

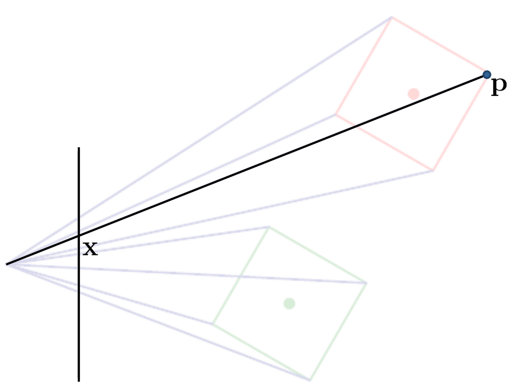
3D to 2D Projections

- Orthographic projections
- Perspective projections

3D to 2D Perspective Projection



3D to 2D Perspective Projection



3D to 2D Perspective Projection

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

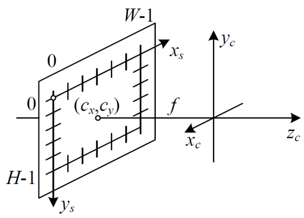
$$\tilde{\mathbf{x}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \bar{\mathbf{p}}$$

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

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} \quad \Rightarrow \quad \mathbf{x} = \begin{pmatrix} \tilde{x}/\tilde{z} \\ \tilde{y}/\tilde{z} \end{pmatrix}$$

Camera Intrinsics

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



Camera Intrinsics

- Need to apply some scaling/offset

$$\tilde{\mathbf{x}} = \underbrace{\begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{intrinsics } K} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{projection}} \tilde{\mathbf{p}}$$

- Focal length f_x, f_y
- Camera center c_x, c_y
- Skew s

Camera Extrinsics

- Assume $\tilde{\mathbf{p}}_w$ is given in world coordinates
- Transform from world to camera (also called the camera extrinsics)

$$\tilde{\mathbf{p}} = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \tilde{\mathbf{p}}_w$$

- Full camera matrix

$$\tilde{\mathbf{x}} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R & \mathbf{t} \end{pmatrix} \tilde{\mathbf{p}}_w$$

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

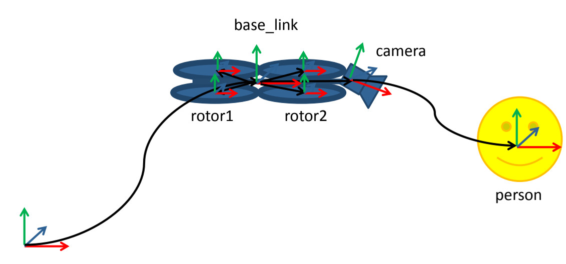
- You **really** have to know 2D/3D transformations by heart (read Szeliski, Chapter 2)

C++ Libraries for Lin. Alg./Geometry

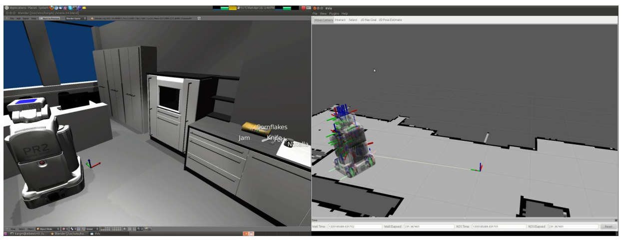
- Many C++ libraries exist for linear algebra and 3D geometry
- Typically conversion necessary
- Examples:
  - C arrays, std::vector (no linear alg. functions)
  - gsl (gnu scientific library, many functions, plain C)
  - boost::array (used by ROS messages)
  - Bullet library (3D geometry, used by ROS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



Example: Video from PR2



Sensors

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

Classification of 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

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

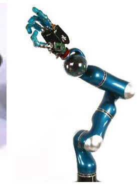
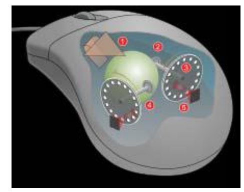
- Bandwidth or Frequency
- Delay
- Sensitivity
- Cross-sensitivity (cross-talk)
- Error (accuracy)
  - Deterministic errors (modeling/calibration possible)
  - Random errors
- Weight, power consumption, …

Sensors

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

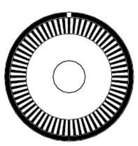
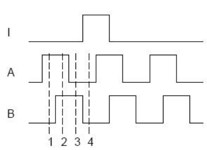
Motor/wheel encoders

- Device for measuring angular motion
- Often used in (wheeled) robots
- Output: position, speed (possibly integrate speed to get odometry)

Motor/wheel encoders

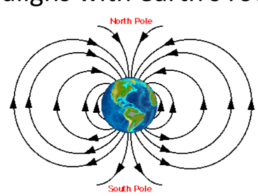
- Working principle:
  - Regular: counts the number of transitions but cannot tell direction
  - Quadrature: uses two sensors in quadrature phase-shift, ordering of rising edge tells direction
  - Sometimes: Reference pulse (or zero switch)

State	Ch A	Ch B
S_1	High	Low
S_2	High	High
S_3	Low	High
S_4	Low	Low

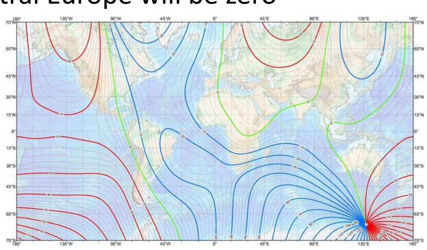
Magnetic Compass

- Measures earth's magnetic field
- Inclination angle approx. 60deg (Germany)
- Does not work indoor/affected by metal
- Alternative: gyro compass (spinning wheel, aligns with earth's rotational poles, for ships)



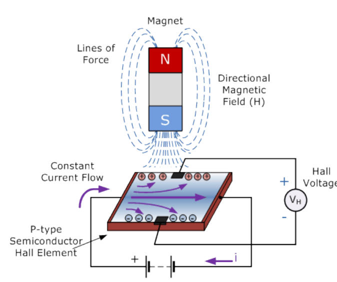
Magnetic Declination

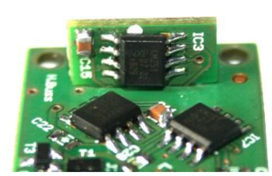
- Angle between magnetic north and true north
- Varies over time
- Good news ;-): by 2050, magnetic declination in central Europe will be zero



Magnetic Compass

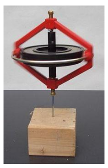
- Sensing principle: Hall sensor
- Construction: 3 orthogonal sensors

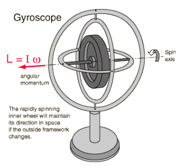




Mechanical Gyroscope

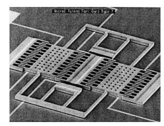
- Measures orientation (standard gyro) or angular velocity (rate gyro, needs integration for angle)
- Spinning wheel mounted in a gimbal device (can move freely in 3 dimensions)
- Wheel keeps orientation due to angular momentum (standard gyro)

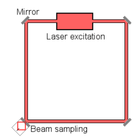


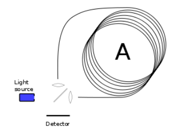


Modern Gyroscopes

- Vibrating structure gyroscope (MEMS)
  - Based on Coriolis effect
  - "Vibration keeps its direction under rotation"
  - Implementations: Tuning fork, vibrating wheels, …
- Ring laser / fibre optic gyro
  - Interference between counter-propagating beams in response to rotation







Accelerometer

- Measures all external forces acting upon them (including gravity)
- Acts like a spring-damper system
- To obtain inertial acceleration (due to motion alone), gravity must be subtracted

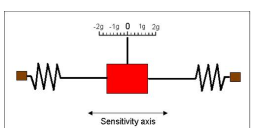
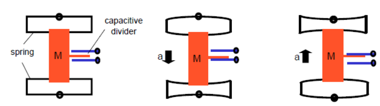
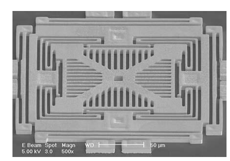


Fig. 1. Schematic structure of an accelerometer

MEMS Accelerometers

- Micro Electro-Mechanical Systems (MEMS)
- Spring-like structure with a proof mass
- Damping results from residual gas
- Implementations: capacitive, piezoelectric, …





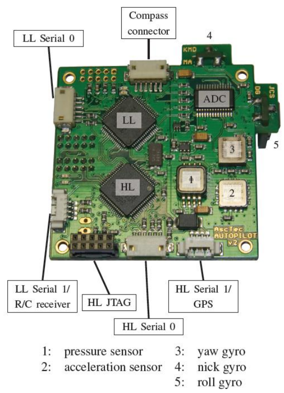
Inertial Measurement Unit

- 3-axes MEMS gyroscope
  - Provides angular velocity
  - Integrate for angular position
  - Problem: Drifts slowly over time (e.g., 1deg/hour), called the bias
- 3-axes MEMS accelerometer
  - Provides accelerations (including gravity)
- Can we use these sensors to estimate our position?

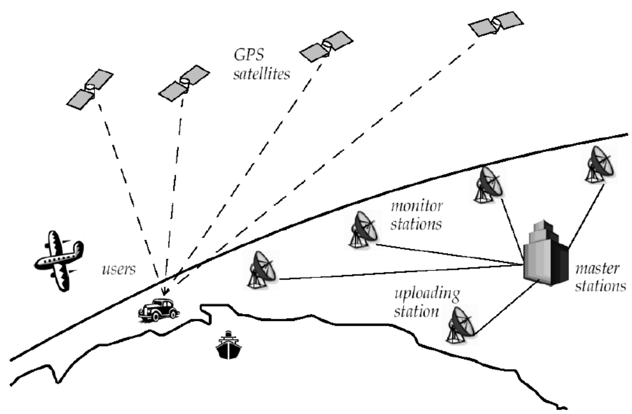
Inertial Measurement Unit

- IMU: Device that uses gyroscopes and accelerometers to estimate (relative) position, orientation, velocity and accelerations
- Integrate angular velocities to obtain absolute orientation
- Subtract gravity from acceleration
- Integrate acceleration to linear velocities
- Integrate linear velocities to position
- Note: All IMUs are subject to drift (position is integrated twice!), needs external reference

Example: AscTec Autopilot Board

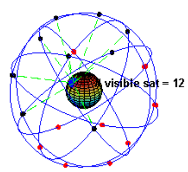


GPS



GPS

- 24+ satellites, 12 hour orbit, 20.190 km height
- 6 orbital planes, 4+ satellites per orbit, 60deg distance



- Satellite transmits orbital location + time
- 50bits/s, msg has 1500 bits → 12.5 minutes

GPS

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

Range Sensors

- **Sonar**

- **Laser range finder**

- Time of flight camera

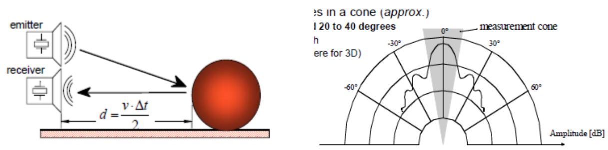
- Structured light (will be covered later)

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: 340m/s
- Light speed: 300.000km/s

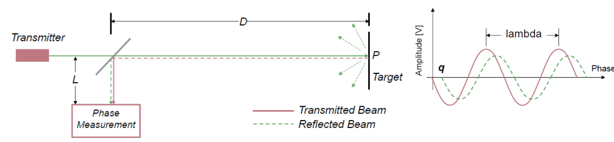
Ultrasonic Range Sensors

- Range between 12cm and 5m
- Opening angle around 20 to 40 degrees
- Soft surfaces absorb sound
- Reflections → ghosts
- Lightweight and cheap



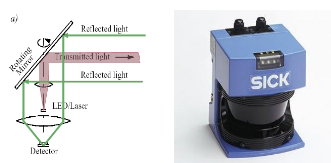
Laser Scanner

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



Laser Scanner

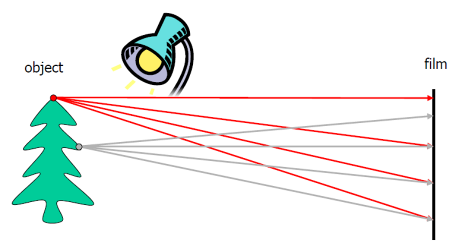
- 2D scanners



- 3D scanners

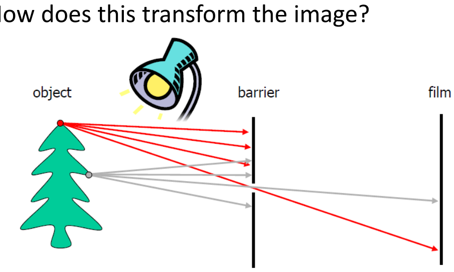
Camera

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



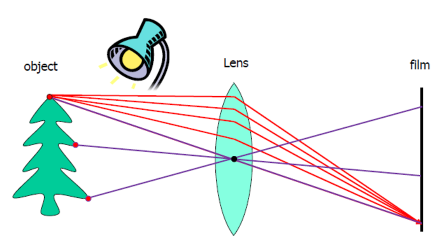
Camera

- Add a barrier to block off most of the rays
  - This reduces blurring
  - The opening known as the **aperture**
  - How does this transform the image?



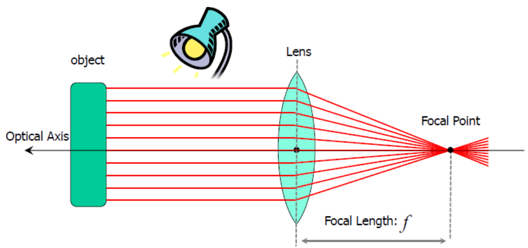
Camera Lens

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



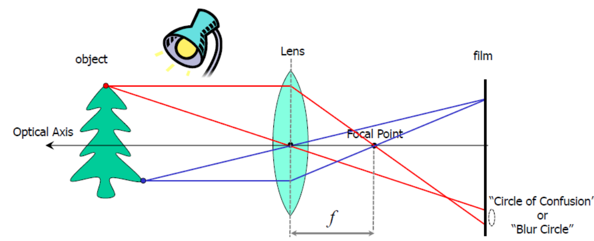
Camera Lens

- 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**



Camera Lens

- There is a specific distance at which objects are "in focus"
- Other points project to a "blur circle" in the image



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

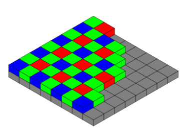
Lens Distortions

- 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

$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$
$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$

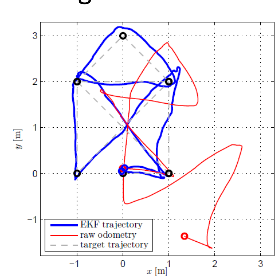
Digital Cameras

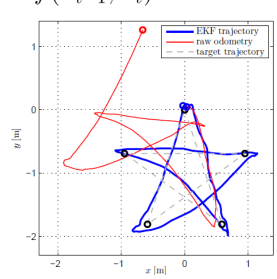
- Vignetting
- De-bayering
- Rolling shutter and motion blur
- Compression (JPG)
- Noise



Dead Reckoning and Odometry

- Estimating the position $\mathbf{x}_t$ based on the issued controls (or IMU) readings $\mathbf{u}_t$
- Integrated over time $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t)$



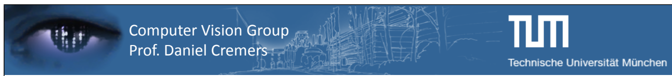


Exercise Sheet 1

- 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 (vx/vy)
  - 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



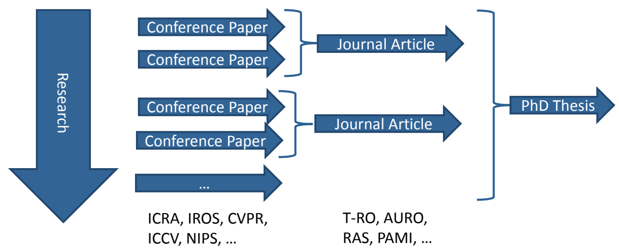
Visual Navigation for Flying Robots

Probabilistic Models and State Estimation

Dr. Jürgen Sturm

Organization

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

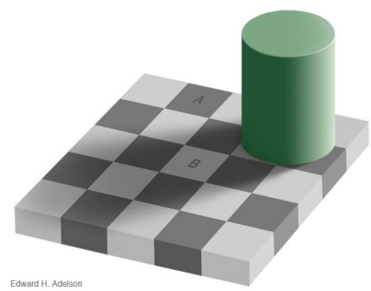


Guest Talks

- An Evaluation of the RGB-D SLAM System (F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, W. Burgard), In Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), 2012.
- Real-Time Visual Odometry from Dense RGB-D Images (F. Steinbruecker, J. Sturm, D. Cremers), In Workshop on Live Dense Reconstruction with Moving Cameras at the Intl. Conf. on Computer Vision (ICCV), 2011.
- 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.

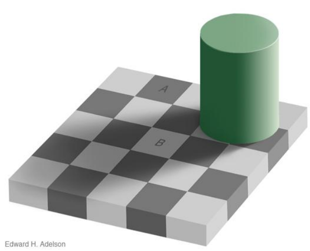
Perception

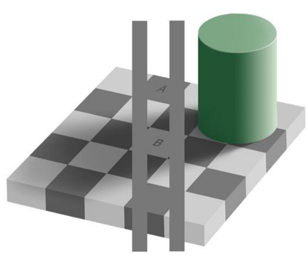
- Perception and models are strongly linked

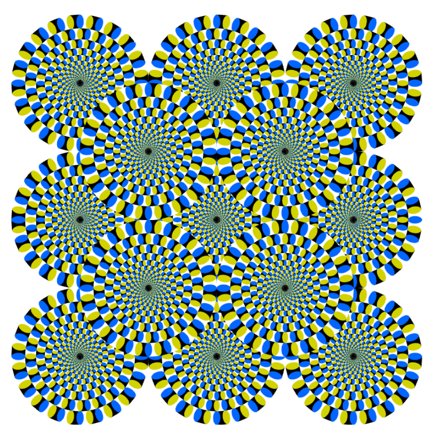


Perception

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



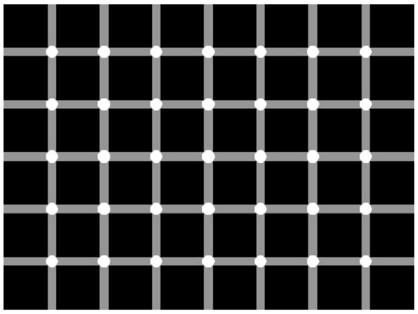




more on http://michaelbach.de/ot/index.html

Models in Human Perception

- Count the black dots



State Estimation

- Cannot observe world state directly
- Need to estimate the world state
- Robot maintains belief about world state
- Update belief according to observations and actions using models
- Sensor observations + sensor model
- Executed actions + action/motion model

State Estimation

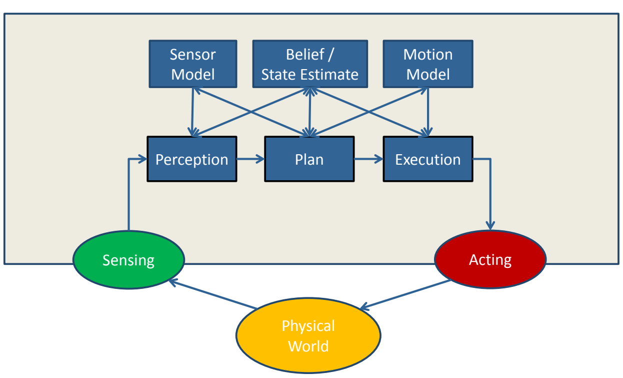
What parts of the world state are (most) relevant for a flying robot?

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



(Deterministic) Sensor Model

- Robot perceives the environment through its sensors

$$z = h(x)$$

sensor reading

world state

observation function

- 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

transition function   executed action

$$x' = g(x, u)$$

current state   previous state

Probabilistic Robotics

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

Probabilistic Robotics

- Probabilistic sensor and motion models

$$p(z \mid x) \qquad p(x' \mid x, u)$$

- Integrate information from multiple sensors (multi-modal)

$$p(x \mid z_{\text{vision}}, z_{\text{ultrasound}}, z_{\text{IMU}})$$

- Integrate information over time (filtering)

$$p(x \mid z_1, z_2, \ldots, z_t)$$

$$p(x \mid u_1, z_1, \ldots, u_t, z_t)$$

Agenda for Today

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

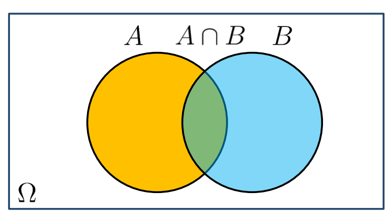
The Axioms of Probability Theory

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

1. $0 \leq P(A) \leq 1$
2. $P(\Omega) = 1 \qquad P(\emptyset) = 0$
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 $\{x_1, x_2, \dots x_n\}$
- $P(X = x_i)$ is the **probability** that the random variable X takes on value x_i
- $P(\cdot)$ is called the **probability mass function**

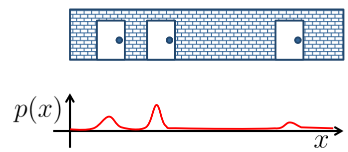
- Example: $P(\text{Room}) =< 0.7, 0.2, 0.08, 0.02 >$
  $\text{Room} \in \{\text{office}, \text{corridor}, \text{lab}, \text{kitchen}\}$

Continuous Random Variables

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

$$P(x \in [a, b]) = \int_a^b p(x)\mathrm{d}x$$

- Example



Proper Distributions Sum To One

- Discrete case $\sum_x P(x) = 1$

- Continuous case $\int p(x)\mathrm{d}x = 1$

Joint and Conditional Probabilities

- $P(X = x \text{ and } Y = y) = P(x, y)$

- 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**
  $$P(x \mid y)P(y) = P(x, y)$$
- 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 $P(x \mid z) = P(x \mid y, z)$
  $P(y \mid z) = P(x \mid x, z)$

- 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 $p(x) = \int p(x, y)\mathrm{d}y$

Example: Marginalization

	x_1	x_2	x_3	x_4	$p_y(Y) \downarrow$
y_1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{4}$
y_2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{4}$
y_3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{4}$
y_4	$\frac{1}{4}$	0	0	0	$\frac{1}{4}$
$p_x(X) \rightarrow$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$	1

Law of Total Probability

- Discrete case
$$P(x) = \sum_y P(x,y) = \sum_y P(x \mid y)P(y)$$
- Continuous case
$$p(x) = \int p(x,y)\mathrm{d}y = \int p(x \mid y)p(y)\mathrm{d}y$$

Expected Value of a Random Variable

- Discrete case $E[X] = \sum_i x_i P(x_i)$
- Continuous case $E[X] = \int x P(X = x)\mathrm{d}x$
- 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

$$\mathrm{Cov}[X] = E[X - E[X]]^2 = E[X^2] - E[X]^2$$

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.,

$$p(x \mid z) \qquad p(x' \mid x, u)$$

Causal vs. Diagnostic Reasoning

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

observation likelihood prior on world state

$$P(x \mid z) = \frac{P(z \mid x)P(x)}{P(z)}$$

prior on sensor observations

Bayes Formula

$$P(x,z) = P(x \mid z)P(z) = P(z \mid x)P(x)$$

$$\Rightarrow$$

$$\boxed{P(x \mid z) = \frac{P(z \mid x)P(x)}{P(z)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}}$$

Normalization

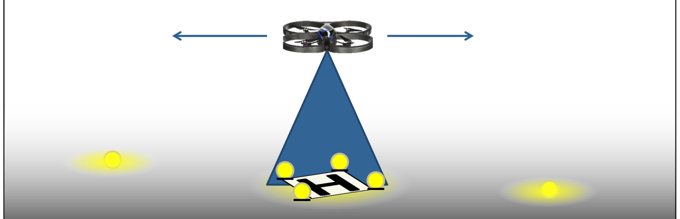
- Direct computation of $P(z)$ can be difficult
- Idea: Compute improper distribution, normalize afterwards
- Step 1: $L(x \mid z) = P(z \mid x)P(x)$
- Step 2: (Law of total probability) $P(z) = \sum_x P(z \mid x)P(x) = \sum_x L(x \mid z)$
- Step 3: $P(x \mid z) = L(x \mid z)/P(z)$

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

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



Example: Sensor Measurement

- Binary sensor $Z \in \{\text{bright}, \neg\text{bright}\}$
- Binary world state $X \in \{\text{home}, \neg\text{home}\}$
- Sensor model $P(Z = \text{bright} \mid X = \text{home}) = 0.6$
  $P(Z = \text{bright} \mid X = \neg\text{home}) = 0.3$
- 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?

Example: Sensor Measurement

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

$$P(X = \text{home} \mid Z = \text{noise})$$
$$= \frac{P(\text{bright} \mid \text{home})P(\text{home})}{P(\text{bright} \mid \text{home})P(\text{home}) + P(\text{bright} \mid \neg\text{home})P(\neg\text{home})}$$
$$= \frac{0.6 \cdot 0.5}{0.6 \cdot 0.5 + 0.3 \cdot 0.5} = \frac{0.3}{0.3 + 0.15} = 0.67$$

Example: Sensor Measurement

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

$$P(X = \text{home} \mid Z = \text{noise})$$
$$= \frac{P(\text{bright} \mid \text{home})P(\text{home})}{P(\text{bright} \mid \text{home})P(\text{home}) + P(\text{bright} \mid \neg\text{home})P(\neg\text{home})}$$
$$= \frac{0.6 \cdot 0.5}{0.6 \cdot 0.5 + 0.3 \cdot 0.5} = \frac{0.3}{0.3 + 0.15} = 0.67$$

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 $p(x \mid z_1, z_2, \ldots)$?

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

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

Recursive Bayesian Updates

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

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

Recursive Bayesian Updates

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

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

$$\begin{aligned} P(x \mid z_1, \ldots, z_n) &= \frac{P(z_n \mid x)P(x \mid z_1, \ldots, z_{n-1})}{P(z_n \mid z_1, \ldots, z_{n-1})} \\ &= \eta P(z_n \mid x)P(x \mid z_1, \ldots, z_{n-1}) \\ &= \eta_{1:n} \prod_{i=1,\ldots,n} P(z_i \mid x)P(x) \end{aligned}$$

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?

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

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?

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

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

Actions (Motions)

- Often the world is dynamic since
  - actions carried out by the robot…
  - actions carried out by other agents…
  - or just time passing by…

  …change the world
- How can we incorporate actions?

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

Action Models

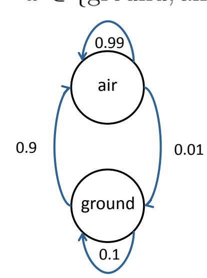
- To incorporate the outcome of an action $\mathbf{u}$ into the current state estimate ("belief"), we use the conditional pdf

$$p(x' \mid u, x)$$

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

Example: Take-Off

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



Integrating the Outcome of Actions

- Discrete case

$$P(x' \mid u) = \sum_x P(x' \mid u, x) P(x)$$

- Continuous case

$$p(x' \mid u) = \int p(x' \mid u, x) p(x) \mathrm{d}x$$

Example: Take-Off

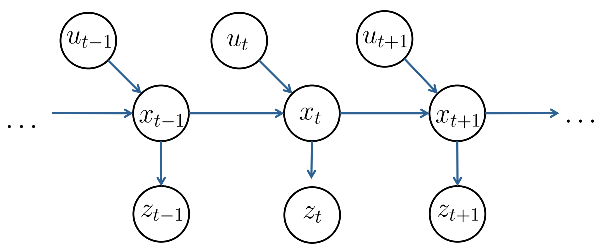
- Prior belief on robot state: $P(x = \text{ground}) = 1.0$ (robot is located on the ground)
- Robot executes "take-off" action
- What is the robot's belief after one time step?

$$\begin{aligned}P(x' = \text{ground}) = & \sum_x P(x' = \text{ground} \mid u, x) P(x) \\ = & P(x' = \text{ground} \mid u, x = \text{ground}) P(x = \text{ground}) \\ & + P(x' = \text{ground} \mid u, x = \text{air}) P(x = \text{air}) \\ = & 0.1 \cdot 1.0 + 0.01 \cdot 0.0 = 0.1\end{aligned}$$

- Question: What is the probability at t=2?

Markov Chain

- A Markov chain is a stochastic process where, given the present state, the past and the future states are independent



Markov Assumption

- Observations depend only on current state

$$P(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) = P(z_t \mid x_t)$$

- Current state depends only on previous state and current action

$$P(x_t \mid x_{0:t-1}, z_{1:t}, u_{1:t}) = P(x_t \mid x_{t-1}, u_t)$$

- Underlying assumptions
  - Static world
  - Independent noise
  - Perfect model, no approximation errors

Bayes Filter

- Given:
  - Stream of observations z and actions u:

$$\mathbf{d}_t = (u_1, z_1, \ldots, u_t, z_t)^\top$$

  - 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**

$$Bel(x_t) = P(x_t \mid u_1, z_1, \ldots, u_t, z_t)$$

Bayes Filter

For each time step, do

1. Apply motion model

$$\overline{\mathrm{Bel}}(x_t) = \sum_{x_{t-1}} P(x_t \mid x_{t-1}, u_t) \mathrm{Bel}(x_{t-1})$$

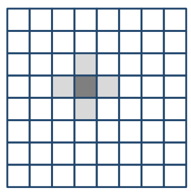
2. Apply sensor model

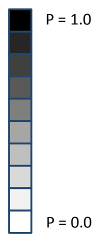
$$\mathrm{Bel}(x_t) = \eta P(z_t \mid x_t) \overline{\mathrm{Bel}}(x_t)$$

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**

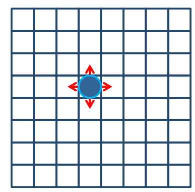




$P(x_{ij}) = a_{ij}$

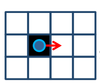
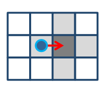
Example: Localization

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



Example: Localization

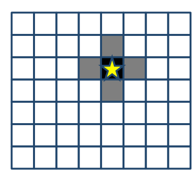
- Action $u \in \{\text{north}, \text{east}, \text{south}, \text{west}\}$
- Robot can move one cell in each time step
- Actions are not perfectly executed
- Example: move east

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

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

Example: Localization

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

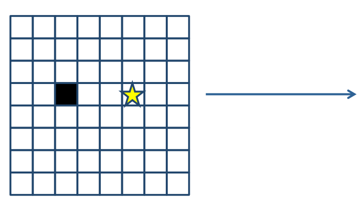


Example: Localization

- Let's start a simulation run… (shades are hand-drawn, not exact!)

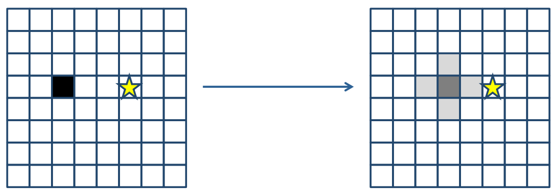
Example: Localization

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



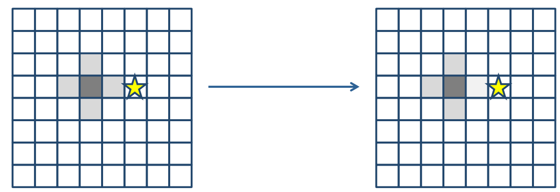
Example: Localization

- t=1, u=east, z=no-marker
- Bayes filter step 1: Apply motion model



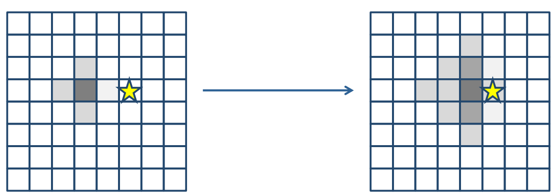
Example: Localization

- t=1, u=east, z=no-marker
- Bayes filter step 2: Apply observation model



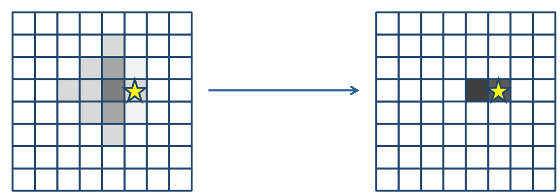
Example: Localization

- t=2, u=east, z=marker
- Bayes filter step 2: Apply motion model



Example: Localization

- t=2, u=east, z=marker
- 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

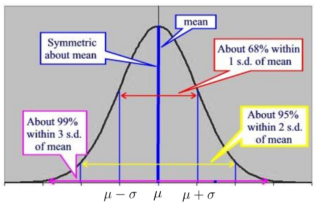
- Bayes filter with continuous states
- State represented with a normal distribution
- Developed in the late 1950's
- Kalman filter is very efficient (only requires a few matrix operations per time step)
- Applications range from economics, weather forecasting, satellite navigation to robotics and many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

Normal Distribution

- Univariate normal distribution

$$X \sim \mathcal{N}(\mu, \sigma)$$

$$p(X = x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$$



Normal Distribution

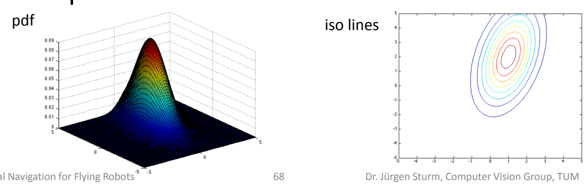
- Multivariate normal distribution

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

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mu, \Sigma)$$

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

- Example: 2-dimensional normal distribution



Properties of Normal Distributions

- Linear transformation → remains Gaussian

$$X \sim \mathcal{N}(\mu, \Sigma), Y \sim AX + B$$

$$\Rightarrow Y \sim \mathcal{N}(A\mu + B, A\Sigma A^\top)$$

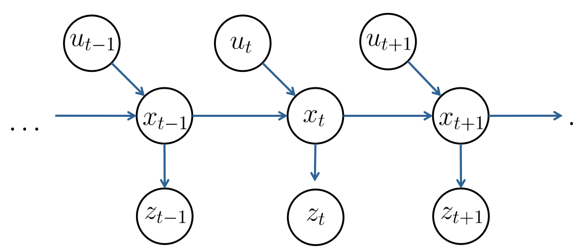
- Intersection of two Gaussians → remains Gaussian

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

$$\Rightarrow p(X_1, X_2) = \mathcal{N}\left(\frac{\Sigma_2}{\Sigma_1 + \Sigma_2}\mu_1 + \frac{\Sigma_1}{\Sigma_1 + \Sigma_2}\mu_2, \frac{1}{\Sigma_1^{-1} + \Sigma_2^{-1}}\right)$$

Linear Process Model

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



Linear Process Model

- Consider a time-discrete stochastic process
- Represent the estimated state (belief) by a Gaussian $x_t \sim \mathcal{N}(\mu_t, \Sigma_t)$

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, 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 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, depends linearly on the controls, and has zero-mean, normally distributed process noise

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

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

Linear Observations

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

$$z_t = Cx_t$$

Linear Observations

- Further, assume we make observations that depend linearly on the state and that are perturbed by zero-mean, normally distributed observation noise

$$z_t = Cx_t + \delta_t$$

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

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

$$x_t = \underbrace{A}_{n \times n} x_{t-1} + \underbrace{B}_{n \times l} u_t + \epsilon$$

- Measurement equation

$$z_t = \underbrace{C}_{n \times k} x_t + \delta_t$$

Kalman Filter

- Initial belief is Gaussian

$$\mathrm{Bel}(x_0) = \mathcal{N}(x_0; \mu_0, \Sigma_0)$$

- 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

$$\overline{\mathrm{Bel}}(x_t) = \int \underbrace{p(x_t \mid x_{t-1}, u_t)}_{\mathcal{N}(x_t; Ax_t + Bu_t, Q)} \underbrace{\mathrm{Bel}(x_{t-1})}_{\mathcal{N}(x_{t-1}; \mu_{t-1}, \Sigma_{t-1})} \, \mathrm{d}x_{t-1}$$

From Bayes Filter to Kalman Filter

For each time step, do

1. Apply motion model

$$\overline{\mathrm{Bel}}(x_t) = \int \underbrace{p(x_t \mid x_{t-1}, u_t)}_{\mathcal{N}(x_t; Ax_{t-1} + Bu_t, Q)} \underbrace{\mathrm{Bel}(x_{t-1})}_{\mathcal{N}(x_{t-1}; \mu_{t-1}, \Sigma_{t-1})} \, \mathrm{d}x_{t-1}$$

$$= \mathcal{N}(x_t; A\mu_{t-1} + Bu_t, A\Sigma A^\top + Q)$$

$$= \mathcal{N}(x_t; \bar{\mu}_t, \bar{\Sigma}_t)$$

From Bayes Filter to Kalman Filter

For each time step, do

2. Apply sensor model

$$\mathrm{Bel}(x_t) = \eta \underbrace{p(z_t \mid x_t)}_{\mathcal{N}(z_t; Cx_t, R)} \underbrace{\overline{\mathrm{Bel}}(x_t)}_{\mathcal{N}(x_t; \bar{\mu}_t, \bar{\Sigma}_t)}$$

$$= \mathcal{N}(x_t; \bar{\mu}_t + K_t(z_t - C\bar{\mu}), (I - K_t C)\bar{\Sigma})$$

$$= \mathcal{N}(x_t; \mu_t, \Sigma_t)$$

with $K_t = \bar{\Sigma}_t C^\top (C\bar{\Sigma}_t C^\top + R)^{-1}$

Kalman Filter

For each time step, do

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

1. Apply motion model

$$\bar{\mu}_t = A\mu_{t-1} + Bu_t$$
$$\bar{\Sigma}_t = A\Sigma A^\top + Q$$

2. Apply sensor model

$$\mu_t = \bar{\mu}_t + K_t(z_t - C\bar{\mu}_t)$$
$$\Sigma_t = (I - K_t C)\bar{\Sigma}_t$$

with $K_t = \bar{\Sigma}_t C^\top (C\bar{\Sigma}_t C^\top + R)^{-1}$

Kalman Filter

- Highly efficient: Polynomial in the measurement dimensionality *k* and state dimensionality *n*:

$$O(k^{2.376} + n^2)$$

- **Optimal for linear Gaussian systems**!
- Most robotics systems are **nonlinear**!

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

$$g(x_{t-1}, u_t) \approx g(\mu_{t-1}, u_t) + \frac{\partial g(\mu_{t-1}, u_t)}{\partial x_{t-1}}(x_{t-1} - \mu_{t-1})$$
$$= g(\mu_{t-1}, u_t) + G_t(x_{t-1} - \mu_{t-1})$$

- Observation function

$$h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t}(x_t - \mu_t)$$
$$= h(\bar{\mu}_t) + H_t(x_t - \mu_t)$$

Extended Kalman Filter

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

For each time step, do

1. Apply motion model

$$\bar{\mu}_t = g(\mu_{t-1}, u_t)$$
$$\bar{\Sigma}_t = G_t \Sigma G_t^\top + Q \quad \text{with } G_t = \frac{\partial g(\mu_{t-1}, u_t)}{\partial x_{t-1}}$$

2. Apply sensor model

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$
$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$$

with $K_t = \bar{\Sigma}_t H_t^\top (H_t \bar{\Sigma}_t H_t^\top + R)^{-1}$ and $H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t}$

Example

- 2D case
- State $\mathbf{x} = \begin{pmatrix} x & y & \psi \end{pmatrix}^\top$
- Odometry $\mathbf{u} = \begin{pmatrix} \dot{x} & \dot{y} & \dot{\psi} \end{pmatrix}^\top$
- Observations of visual marker $\mathbf{z} = \begin{pmatrix} x & y & \psi \end{pmatrix}^\top$ (relative to robot pose)

Example

- Motion Function and its derivative

$$g(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} x + (\cos(\psi)\dot{x} - \sin(\psi)\dot{y})\Delta t \\ y + (\sin(\psi)\dot{x} + \cos(\psi)\dot{y})\Delta t \\ \psi + \dot{\psi}\Delta t \end{pmatrix}$$

$$G = \frac{\partial g(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} = \begin{pmatrix} 1 & 0 & (-\sin(\psi)\dot{x} - \cos(\psi)\dot{y})\Delta t \\ 0 & 1 & (\cos(\psi)\dot{x} + \sin(\dot{\psi})\dot{y})\Delta t \\ 0 & 0 & 1 \end{pmatrix}$$

Example

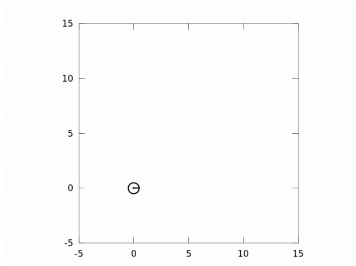
- Observation Function (→ Sheet 2)

$$h(\mathbf{x}) = \dots$$

$$H = \frac{\partial h(\mathbf{x})}{\partial x} = \dots$$

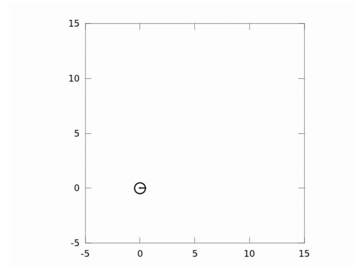
Example

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



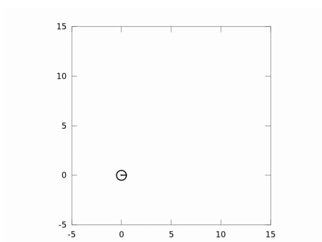
Example

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



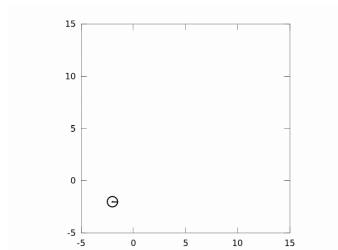
Example

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



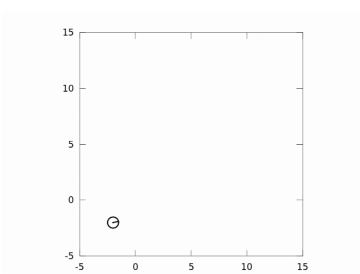
Example

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



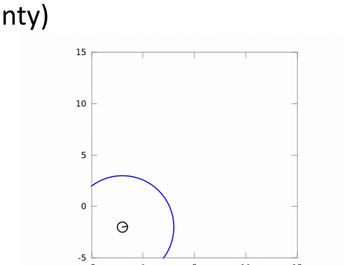
Example

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

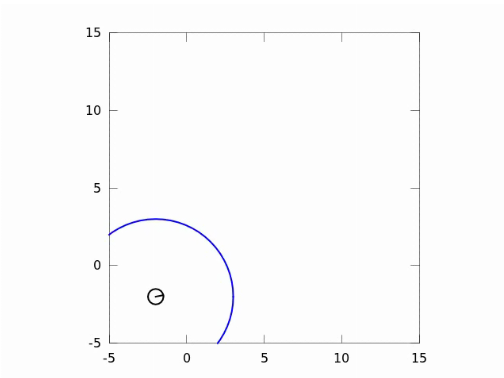


Example

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



Example



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

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

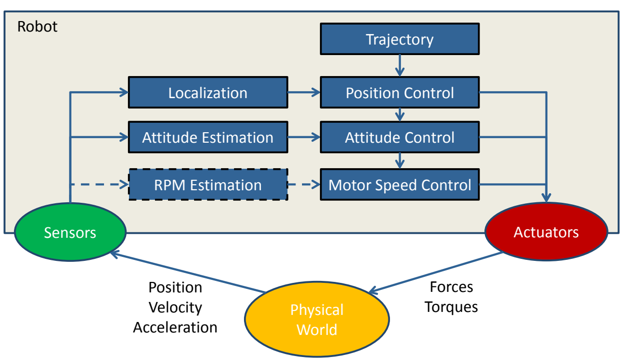
Robot Control

Dr. Jürgen Sturm

Organization - Exam

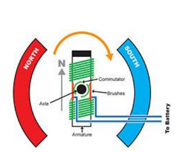
- Oral exams **in teams** (2-3 students)
- At least 15 minutes per student
  → individual grades
- Questions will address
  - Material from the lecture
  - Material from the exercise sheets
  - Your mini-project

Control Architecture

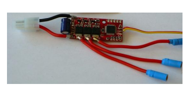


DC Motors

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

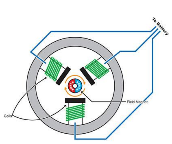


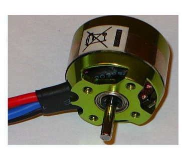
Brushless Motors



- Used in most quadrocopters
- Permanent magnets on the axis
- Electromagnets on the outside
- Requires motor controller to switch currents

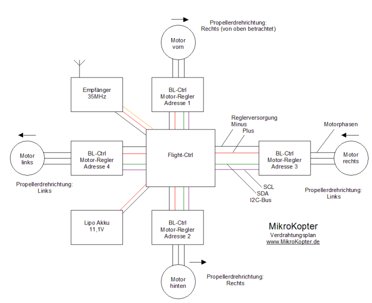
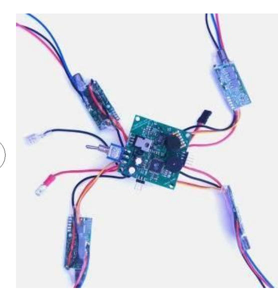
→ Does not require brushes (less maintenance)





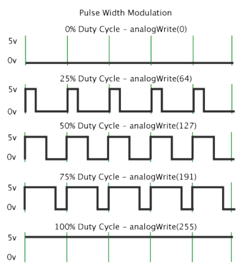
Attitude + Motor Controller Boards

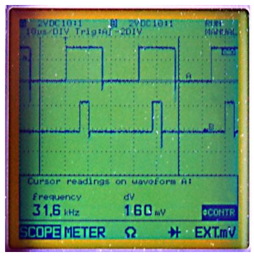
- Example: Mikrokopter Platform

Pulse Width Modulation (PWM)

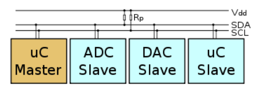
- Protocol used to control motor speed
- Remote controls typically output PWM

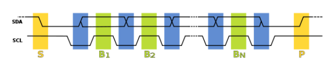




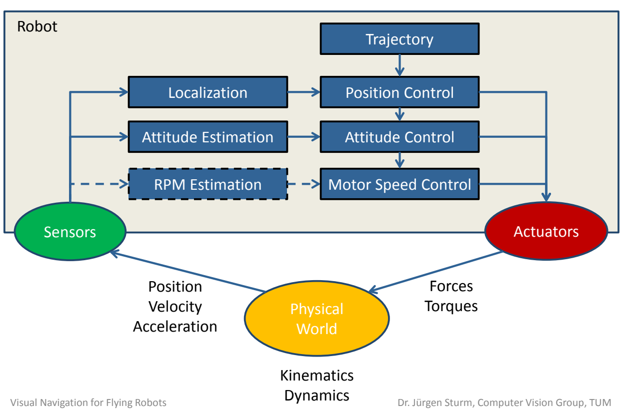
I2C Protocol

- 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





Control Architecture



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 $\mathbf{x} = \begin{pmatrix} x & \dot{x} & \ddot{x} \end{pmatrix}^\top \in \mathbb{R}^3$
- 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)

$$m\ddot{\mathbf{x}} = \sum_i F_i$$

- Torque (Drehmoment)

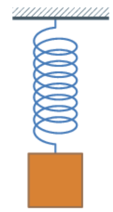
$$J\boldsymbol{\alpha} = \sum_i \boldsymbol{\tau}_i$$

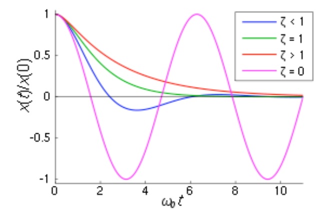
Forces

- 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 $F_{\text{spring}} = K(x - x_{\text{eq}})$
- Magnetic force
- …

Example: Spring-Damper System

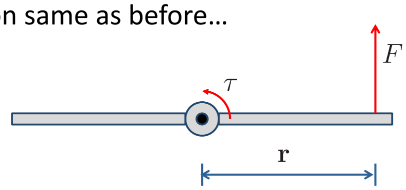
- Combination of spring and damper
- Forces $F = F_{\text{damping}} + F_{\text{spring}}$
- Resulting dynamics $m\ddot{x} = D\dot{x} + K(x - x_{\text{eq}})$





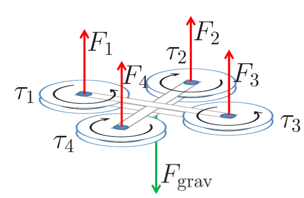
Torques

- Definition $\boldsymbol{\tau} = F \times \mathbf{r}$
- Torques sum up $\boldsymbol{\tau}_{\text{net}} = \sum \boldsymbol{\tau}_i$
- Torque results in angular acceleration $\boldsymbol{\tau} = J\boldsymbol{\alpha}$ (with $\boldsymbol{\alpha} = \frac{\mathrm{d}\omega}{\mathrm{d}t}$, J moment of inertia)
- Friction same as before…



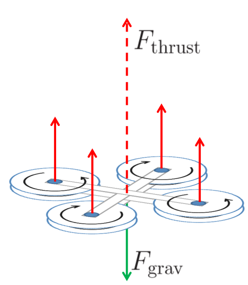
Dynamics of a Quadrocopter

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



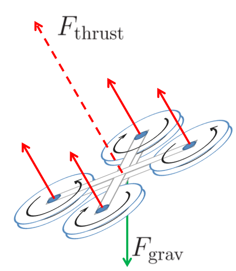
Vertical Acceleration

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



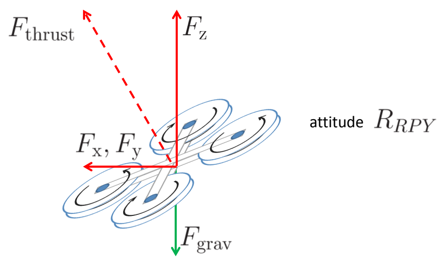
Vertical and Horizontal Acceleration

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



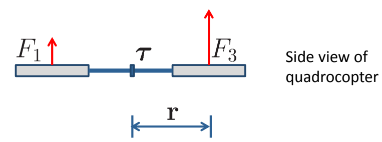
Vertical and Horizontal Acceleration

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



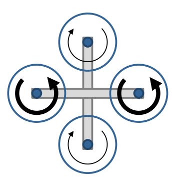
Pitch (and Roll)

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

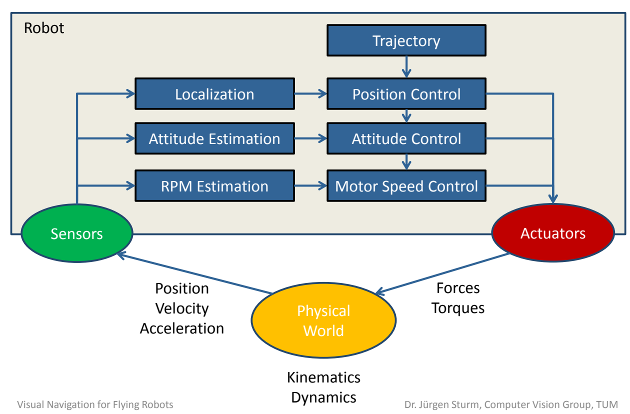


Yaw

- Each propeller induces torque due to rotation and the interaction with the air
- Induced torque $\boldsymbol{\tau} = \boldsymbol{\tau}_1 - \boldsymbol{\tau}_2 + \boldsymbol{\tau}_3 - \boldsymbol{\tau}_4$
- Induced angular acceleration



Cascaded Control



Assumptions of Cascaded Control

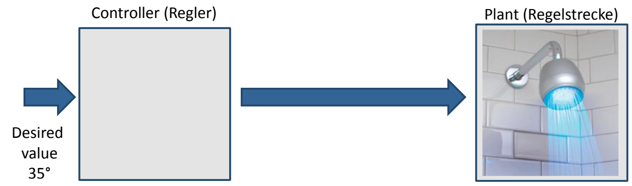
- Dynamics of inner loops is so fast that it is not visible from outer loops
- Dynamics of outer loops is so slow that it appears as static to the inner loops

Cascaded Control Example

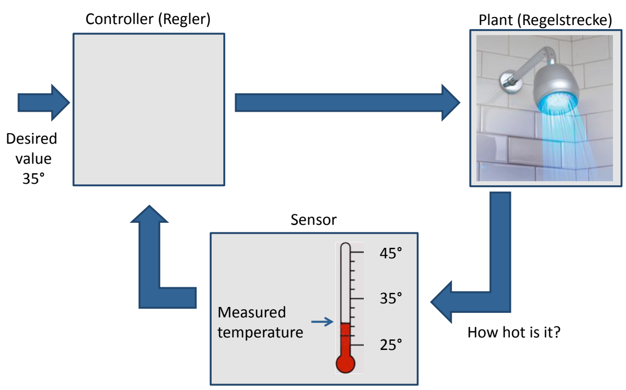
- Motor control happens on motor boards (controls every motor tick)
- Attitude control implemented on micro-controller with hard real-time (at 1000 Hz)
- Position control (at 10 – 250 Hz)
- Trajectory (waypoint) control (at 0.1 – 1 Hz)

Feedback Control - Generic Idea

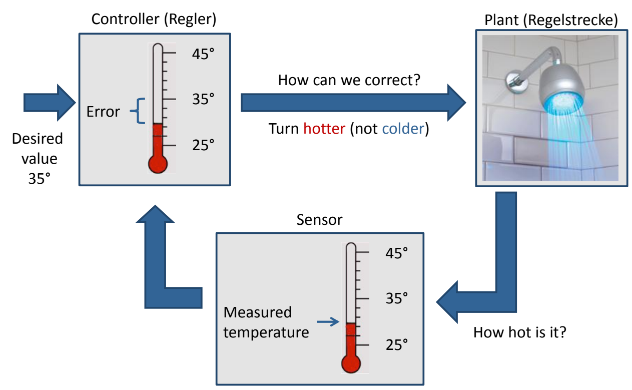
Feedback Control - Generic Idea



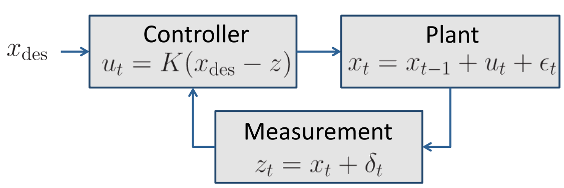
Feedback Control - Generic Idea

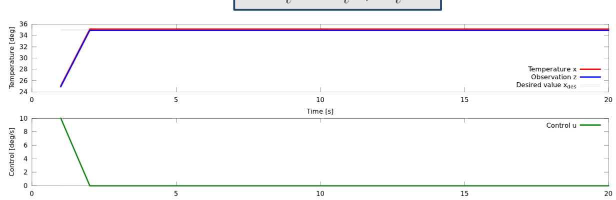


Feedback Control - Generic Idea



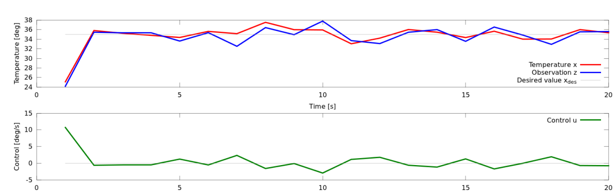
Feedback Control - Example





Measurement Noise

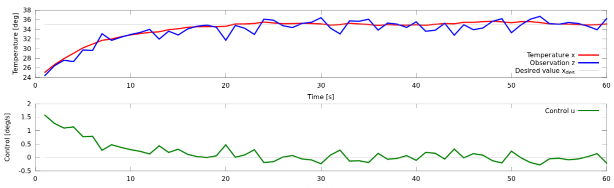
- What effect has noise in the measurements?



- Poor performance for K=1
- How can we fix this?

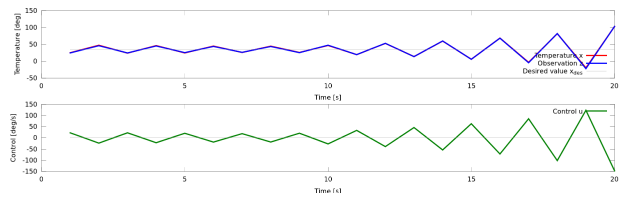
Proper Control with Measurement Noise

- Lower the gain… (K=0.15)



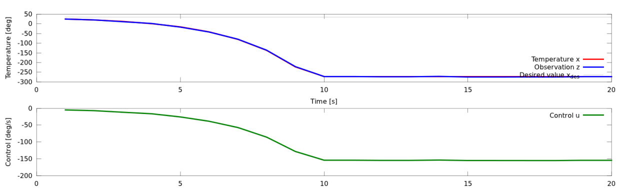
What do High Gains do?

- High gains are always problematic (K=2.15)



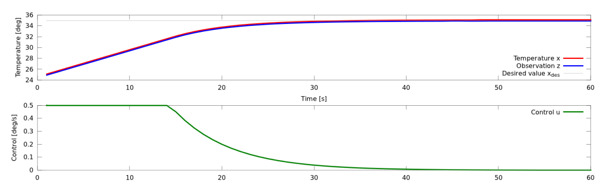
What happens if sign is messed up?

- Check K=-0.5

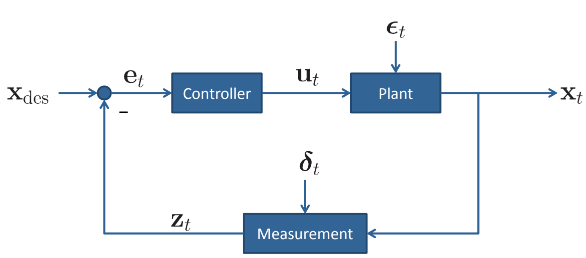


Saturation

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

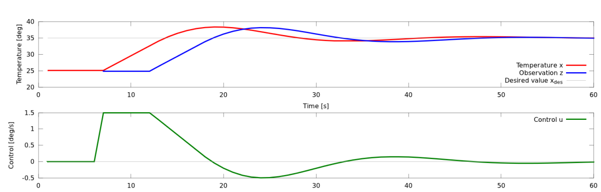


Block Diagram



Delays

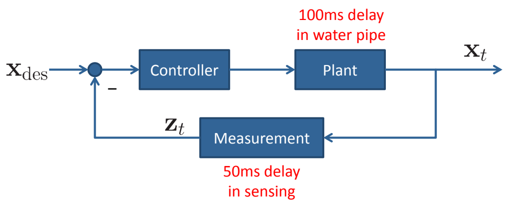
- In practice most systems have delays
- Can lead to overshoots/oscillations/de-stabilization



- One solution: lower gains (why is this bad?)

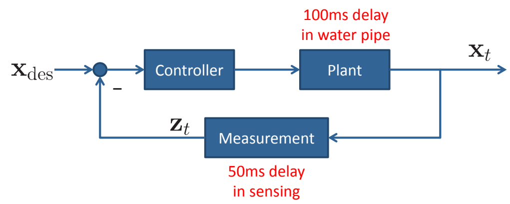
Delays

- What is the total dead time of this system?



Delays

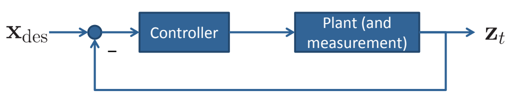
- What is the total dead time of this system?



- Can we distinguish delays in the measurement from delays in actuation?

Delays

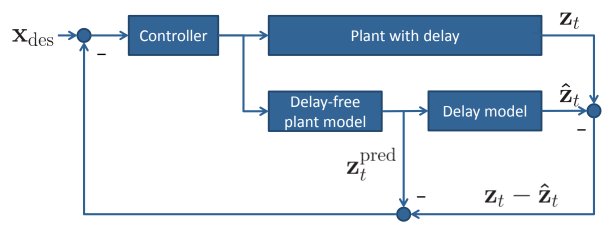
- What is the total dead time of this system?



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

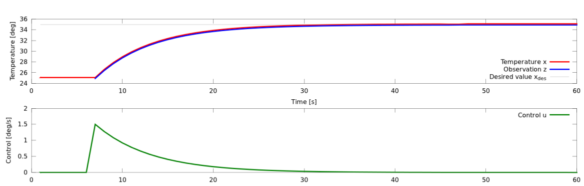
Smith Predictor

- Allows for higher gains
- Requires (accurate) model of plant



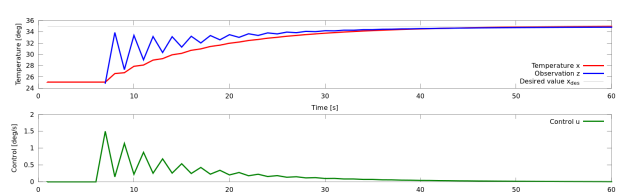
Smith Predictor

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



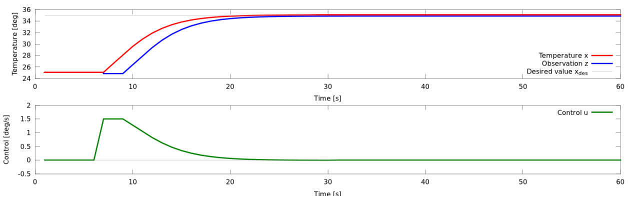
Smith Predictor

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

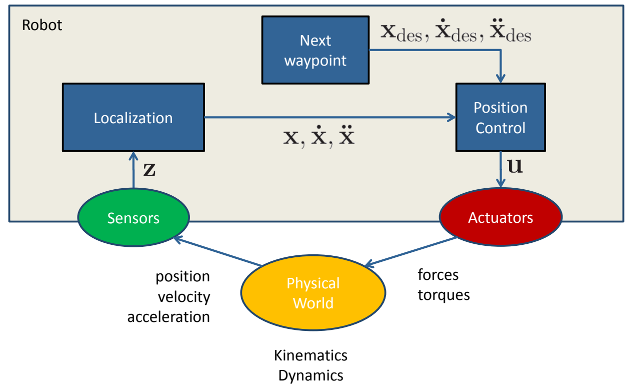


Smith Predictor

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



Position Control

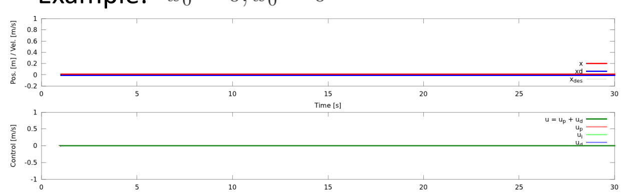


Rigid Body Kinematics

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

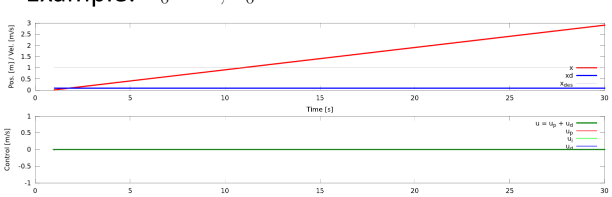
Rigid Body Kinematics

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



Rigid Body Kinematics

- Consider a rigid body
- Free floating in 1D space, no gravity
- How does this system evolve over time?
- Example: $x_0 = 0, \dot{x}_0 = 0.1$



Rigid Body Kinematics

- Consider a rigid body
- Free floating in 1D space, no gravity
- In each time instant, we can apply a force F
- Results in acceleration $\ddot{x} = F/m$
- Desired position $x_{\text{des}} = 1$

P Control

- What happens for this control law?

$$u_t = K(x_{\text{des}} - x_{t-1})$$

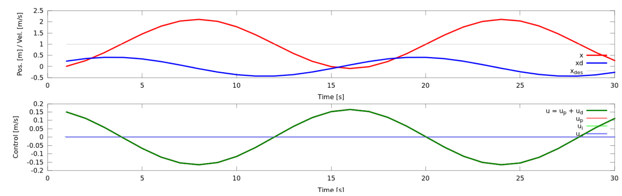
- This is called proportional control

P Control

- What happens for this control law?

$$u_t = K(x_{\text{des}} - x_{t-1})$$

- This is called proportional control

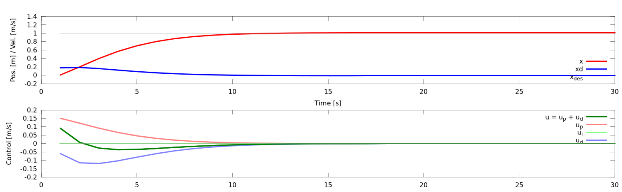


PD Control

- 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

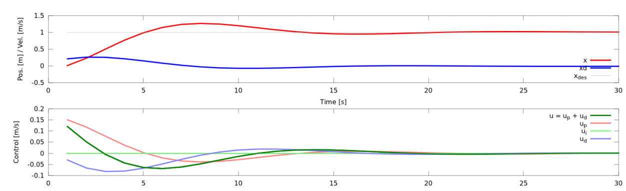


PD Control

- What happens 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?

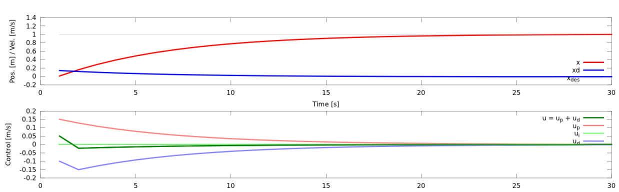


PD Control

- What happens for this control law?

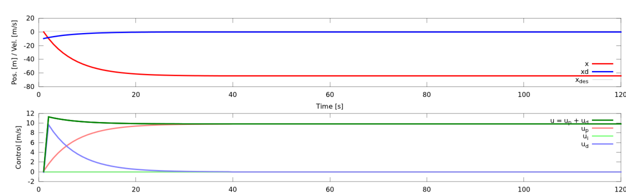
$$u_t = K_P(x_{\text{des}} - x_{t-1}) + K_D(\dot{x}_{\text{des}} - \dot{x}_{t-1})$$

- What if we set **lower** gains?



PD Control

- What happens when we add gravity?

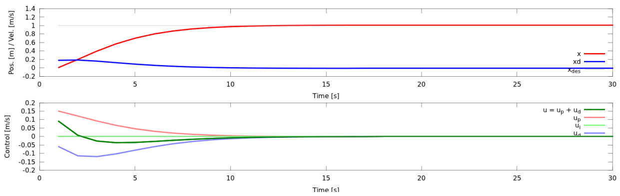


Gravity compensation

- Add as an additional term in the control law

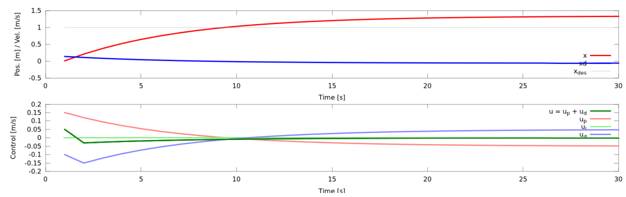
$$u_t = K_P(x_{\text{des}} - x_{t-1}) + K_D(\dot{x}_{\text{des}} - \dot{x}_{t-1}) + F_{\text{grav}}$$

- 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?

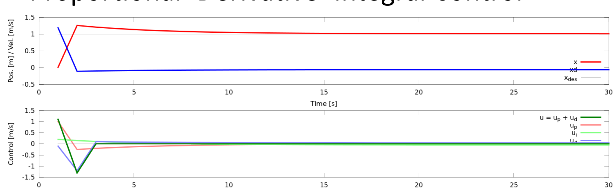


PID Control

- Idea: Estimate the system error (bias) by integrating the error

$$u_t = K_P(x_{\text{des}} - x_t) + K_D(\dot{x}_{\text{des}} - \dot{x}_t) + K_I \int_{-\infty}^{t} x_{des} - x_t \mathrm{d}t$$

- Proportional+Derivative+Integral Control



PID Control

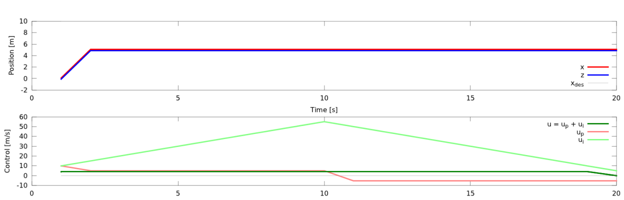
- Idea: Estimate the system error (bias) by integrating the error

$$u_t = K_P(x_{\text{des}} - x_t) + K_D(\dot{x}_{\text{des}} - \dot{x}_t) + K_I \int_{-\infty}^{t} x_{des} - x_t \mathrm{d}t$$

- Proportional+Derivative+Integral Control
- For steady state systems, this can be reasonable
- Otherwise, it may create havoc or even disaster (wind-up effect)

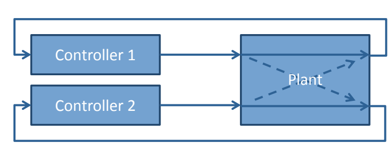
Example: Wind-up effect

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



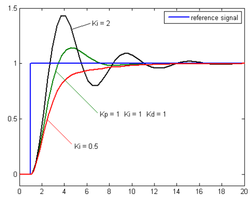
De-coupled Control

- 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?

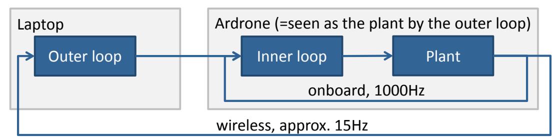
- Gains too large: overshooting, oscillations
- Gains too small: long time to converge
- Heuristic methods exist
- In practice, often tuned manually



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

$$\mathbf{u}_{\text{inner}} = \begin{pmatrix} \tau_1 & \tau_2 & \tau_3 & \tau_4 \end{pmatrix}^\top$$

- Plant output: roll, pitch, yaw rate, z velocity

$$\mathbf{x}_{\text{inner}} = \begin{pmatrix} \omega_x & \omega_y & \dot{\omega}_z & \dot{z} \end{pmatrix}^\top$$

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

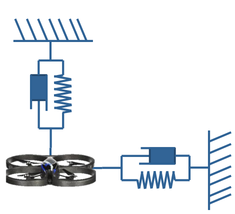
$$\mathbf{u}_{\text{outer}} = \begin{pmatrix} \omega_x & \omega_y & \dot{\omega}_z & \dot{z} \end{pmatrix}^\top$$

- Plant output:

$$\mathbf{x}_{\text{outer}} = \begin{pmatrix} x & y & z & \psi \end{pmatrix}^\top$$

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 → simple proportional control, often enough
- PI control → can compensate for bias (e.g., wind)
- PD control → can be used to reduce overshoot (e.g., when acceleration is controlled)
- PID control → all of the above

Optimal Control

What other control techniques do exist?

- Linear-quadratic regulator (LQR)
- Reinforcement learning
- Inverse reinforcement learning
- ... and many more

Optimal Control

- Find the controller that provides the best performance
- Need to define a measure of performance
- What would be a good performance measure?
  - Minimize the error?
  - Minimize the controls?
  - Combination of both?

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]

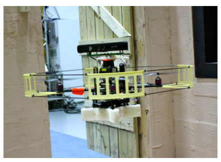
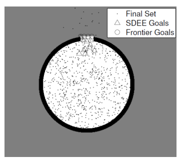
Interesting Papers at ICRA 2012

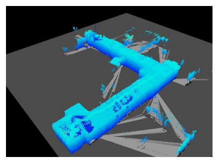
- Flying robots are a hot topic in the robotics community
- 4 (out of 27) sessions on flying robots, 4 sessions on localization and mapping
- Robots: quadrocopters, nano quadrocopters, fixed-wing airplanes
- Sensors: monocular cameras, Kinect, motion capture, laser-scanners

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

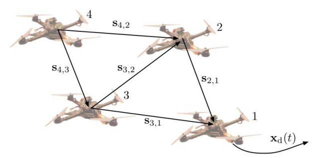





Decentralized Formation Control with Variable Shapes for Aerial Robots

Matthew Turpin, Nathan Michael, and Vijay Kumar

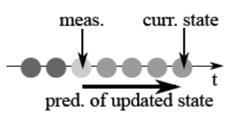
- Move in formation (e.g., to traverse a window)
- Avoid collisions
- Dynamic role switching

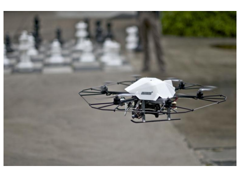
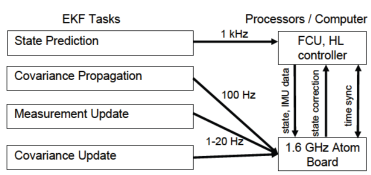


Versatile Distributed Pose Estimation and Sensor Self-Calibration for an Autonomous MAV

Stephan Weiss, Markus W. Achtelik, Margarita Chli, Roland Siegwart

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

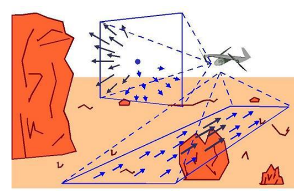


On-board Velocity Estimation and Closed-loop Control of a Quadrotor UAV based on Optical Flow

Volker Grabe, Heinrich H. Bülthoff, and Paolo Robuffo Giordano

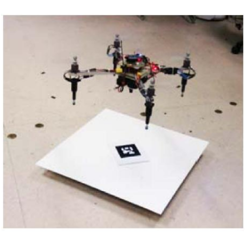
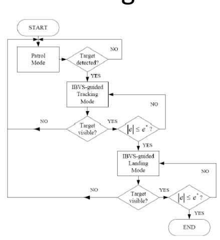
- Ego-motion from optical flow using homography constraint
- Use for velocity control



Autonomous Landing of a VTOL UAV on a Moving Platform Using Image-based Visual Servoing

Daewon Lee, Tyler Ryan and H. Jin. Kim

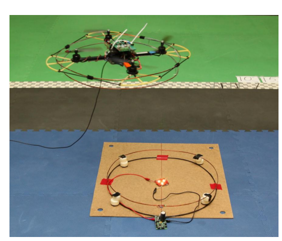
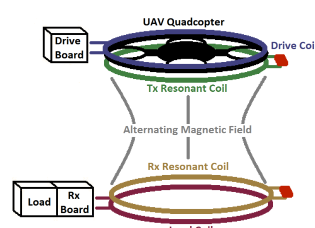
- Tracking and landing on a moving platform
- Switch between tracking and landing behavior

Resonant Wireless Power Transfer to Ground Sensors from a UAV

Brent Griffin and Carrick Detweiler

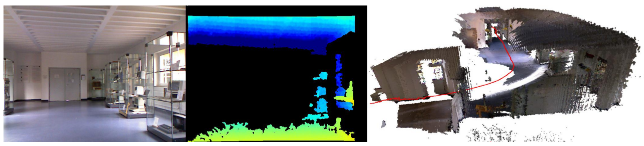
- Quadrocopter transfers power to light a LED

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

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

Visual Motion Estimation

Dr. Jürgen Sturm

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

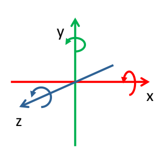
Motivation

Visual Motion Estimation

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

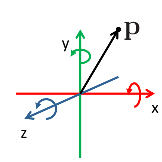
Angular and linear velocities

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



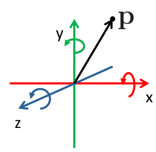
Angular and linear velocities

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



Angular and linear velocities

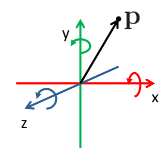
- Linear velocity $\boldsymbol{v} = (v_x, v_y, v_z)^\top \in \mathbb{R}^3$
- Angular velocity $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^\top \in \mathbb{R}^3$
- Now consider a 3D point $\mathbf{p} \in \mathbb{R}^3$ of a rigid body moving with twist $\boldsymbol{\xi} = (\boldsymbol{v}^\top,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



Angular and linear velocities

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

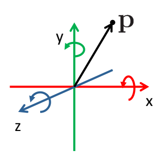
$$\begin{aligned}\mathbf{p}(t) &= R(t)\mathbf{p}(0) + \mathbf{t}(t)\\ &= \exp([\boldsymbol{\omega}]_\times t)\mathbf{p}(0) + \boldsymbol{v}t\end{aligned}$$



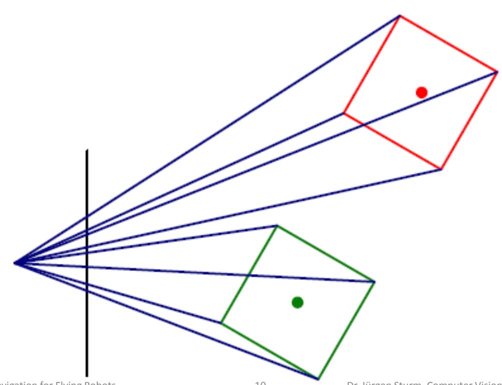
Angular and linear velocities

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

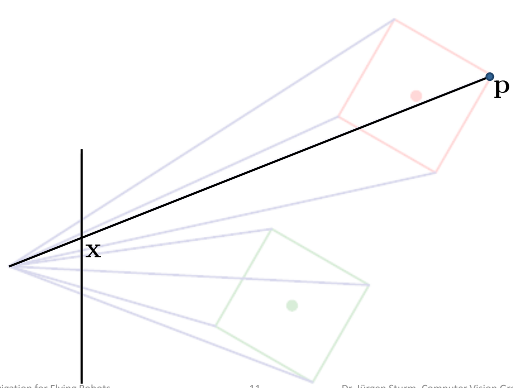
$$\begin{aligned}\mathbf{p}(t) &= R(t)\mathbf{p}(0) + \mathbf{t}(t)\\ &= \exp([\boldsymbol{\omega}]_\times t)\mathbf{p}(0) + \boldsymbol{v}t\\ \Rightarrow \dot{\mathbf{p}}(t) &= \exp([\boldsymbol{\omega}]_\times t)[\boldsymbol{\omega}]_\times\mathbf{p}(0) + \boldsymbol{v}\\ \dot{\mathbf{p}}(0) &= [\boldsymbol{\omega}]_\times\mathbf{p}(0) + \boldsymbol{v}\end{aligned}$$



Recap: Perspective Projection



Recap: Perspective Projection



3D to 2D Perspective Projection

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

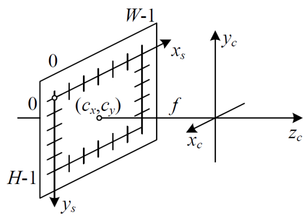
$$\tilde{\mathbf{x}} = \lambda\bar{\mathbf{x}} = \mathbf{p}$$

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

$$\tilde{\mathbf{x}} = \begin{pmatrix}\tilde{x}\\ \tilde{y}\\ \tilde{z}\end{pmatrix} \quad \Rightarrow \quad \mathbf{x} = \begin{pmatrix}\tilde{x}/\tilde{z}\\ \tilde{y}/\tilde{z}\end{pmatrix}$$

Camera Intrinsics

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



Camera Intrinsics

- Need to apply some scaling/offset

$$\tilde{\mathbf{x}} = \underbrace{\begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{intrinsics } K} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{projection}} \tilde{\mathbf{p}}$$

- Focal length f_x, f_y
- Camera center c_x, c_y
- Skew s

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

- We can think of an image as a function $f : \Omega \mapsto \mathbb{R}$
- $f(\mathbf{x})$ gives the intensity at position $\mathbf{x}$
- Color images are vector-valued functions

$$f(\mathbf{x}) = \begin{pmatrix} r(\mathbf{x}) \\ g(\mathbf{x}) \\ b(\mathbf{x}) \end{pmatrix}$$

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

$$F_{ij}, f(i, j), f(x, y), f(\mathbf{x}), \ldots$$

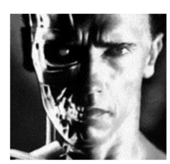
often (row,column)

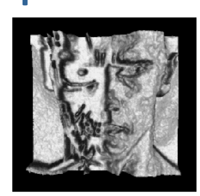
often (column,row)

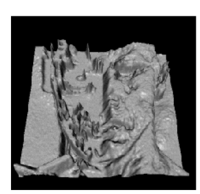
$j \rightarrow$, $i \downarrow$

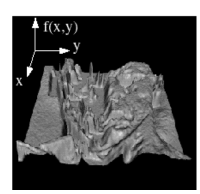
111	115	113	111	112	111	112	111
135	138	137	139	145	146	149	147
163	168	188	196	206	202	206	207
180	184	206	219	202	200	195	193
189	193	214	216	104	79	83	77
191	201	217	220	103	59	60	68
195	205	216	222	113	68	69	83
199	203	223	228	108	68	71	77

Example









Digital Images

- 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)

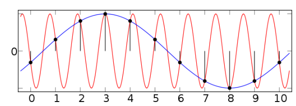
$$V = B^{\frac{1}{\gamma}} \text{ with } \gamma = 2.2$$

- CRTs / monitors do the inverse $B = V^{\gamma}$
- Almost all images are gamma compressed

→ Double brightness results only in a 37% higher intensity value (!)

Aliasing

- High frequencies in the scene and a small fill factor on the chip can lead to (visually) unpleasing effects
- Examples



Rolling Shutter

- Most CMOS sensors have a rolling shutter
- Rows are read out sequentially
- Sensitive to camera and object motion
- Can we correct for this?

Image Filtering

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

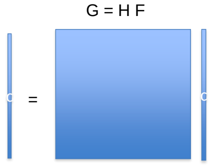
- Example tasks:
de-noising, (de-)blurring, computing derivatives, edge detection, …

Linear Filtering

- Each output is a linear combination of all the input values

$$g(i,j) = \sum_{k,l} h(i,j,k,l) f(k,l)$$

- In matrix form



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

111	115	113	111	112	111	112	111
135	138	137	139	145	146	149	147
163	168	188	196	206	202	206	207
180	184	206	219	202	200	195	193
189	193	214	216	104	79	83	77
191	201	217	220	103	59	60	68
195	205	216	222	113	68	69	83
199	203	223	228	108	68	71	77

$*$

-1	2	-1
-1	2	-1
-1	2	-1

$=$?

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

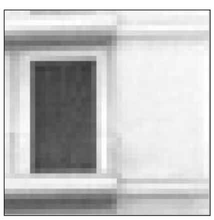
$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 111&115&113&111&112&111&112&111\\\hline 135&138&137&139&145&146&149&147\\\hline 163&168&188&196&206&202&206&207\\\hline 180&184&206&219&202&200&195&193\\\hline 189&193&214&216&104&79&83&77\\\hline 191&201&217&220&103&59&60&68\\\hline 195&205&216&222&113&68&69&83\\\hline 199&203&223&228&108&68&71&77\\\hline\end{array} * \begin{array}{|c|c|c|}\hline -1&2&-1\\\hline -1&2&-1\\\hline -1&2&-1\\\hline\end{array} = \begin{array}{|c|c|c|c|c|c|c|c|}\hline ?&?&?&?&?&?&?&?\\\hline ?&-5&9&-9&21&-12&10&?\\\hline ?&-29&18&24&4&-7&5&?\\\hline ?&-50&40&142&-88&-34&10&?\\\hline ?&-41&41&264&-175&-71&0&?\\\hline ?&-24&37&349&-224&-120&-10&?\\\hline ?&-23&33&360&-217&-134&-23&?\\\hline ?&?&?&?&?&?&?&?\\\hline\end{array}$$

Important Filters

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

Impulse

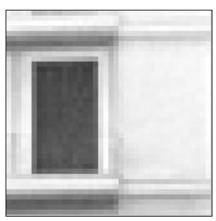
$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$



$f(i,j)$

$$* \begin{array}{|c|c|c|c|c|}\hline 0&0&0&0&0\\\hline 0&0&0&0&0\\\hline 0&0&1&0&0\\\hline 0&0&0&0&0\\\hline 0&0&0&0&0\\\hline\end{array} =$$

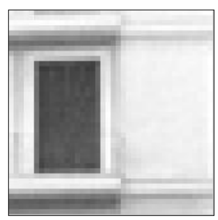
$h(i,j)$



$g(i,j)$

Image shift (translation)

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

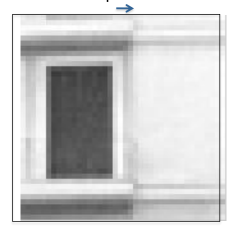


$f(i,j)$

$$* \begin{array}{|c|c|c|c|c|}\hline 0&0&0&0&0\\\hline 0&0&0&0&0\\\hline 1&0&0&0&0\\\hline 0&0&0&0&0\\\hline 0&0&0&0&0\\\hline\end{array} =$$

$h(i,j)$

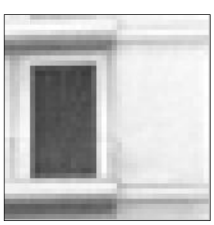
2 pixels →



$g(i,j)$

Image rotation

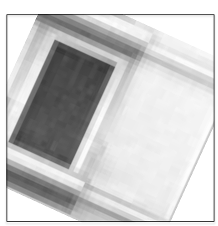
$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$



$f(i,j)$

$* \; ? \; =$

$h(i,j)$

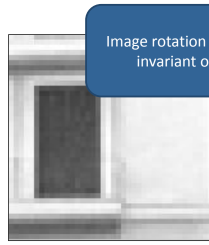


$g(i,j)$

Image rotation

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

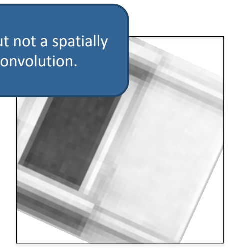
Image rotation is a linear operator (why?), but not a spatially invariant operation (why?). There is no convolution.



$f(i,j)$

$* \; ? \; =$

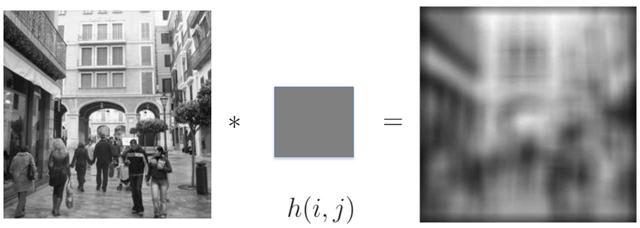
$h(i,j)$



$g(i,j)$

Rectangular Filter

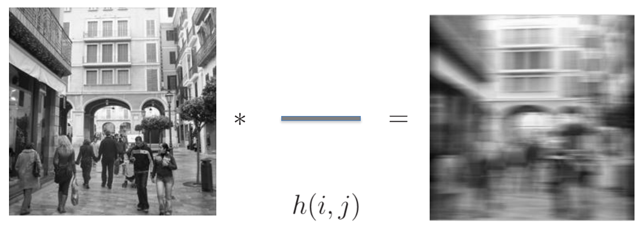
$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$



$f(i,j)$ $*$ $h(i,j)$ $=$ $g(i,j)$

Rectangular Filter

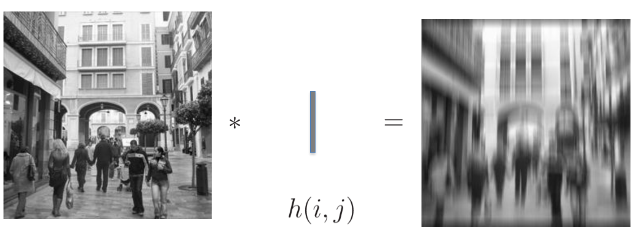
$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$



$f(i,j)$ $*$ $h(i,j)$ $=$ $g(i,j)$

Rectangular Filter

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



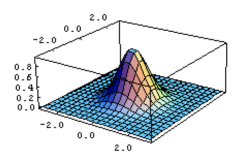
$f(i,j)$ $*$ $h(i,j)$ $=$ $g(i,j)$

Gaussian Blur

- Gaussian distribution

$$g_\sigma(i,i) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)$$

- Example of resulting kernel



Gaussian Blur

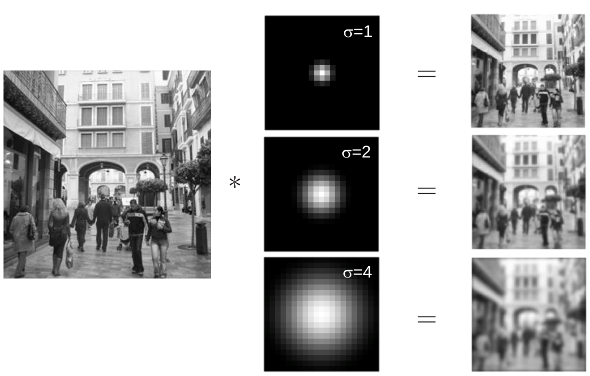
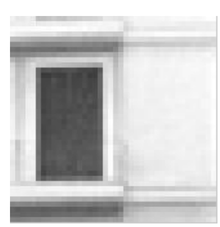


Image Gradient

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



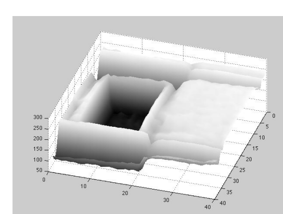
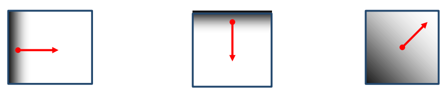


Image Gradient

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



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

Image Gradient

- Gradient direction (related to edge orientation)

$$\theta = \operatorname{atan2}\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right)$$

- Gradient magnitude (edge strength)

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

Image Gradient

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)

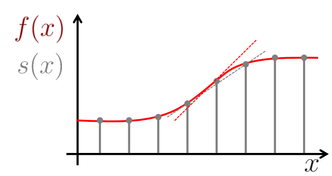
Finite difference

- First-order central difference

$$\frac{\partial f}{\partial x}(x,y) \approx \frac{f(x+1,y) - f(x-1,y)}{2}$$

- Corresponding convolution kernel:

-.5	0	.5



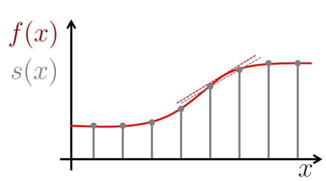
Finite difference

- First-order central difference (half pixel)

$$\frac{\partial f}{\partial x}(x,y) \approx f(x+0.5,y) - f(x-0.5,y)$$

- Corresponding convolution kernel:

-1	1



Second-order Derivative

- Differentiate again to get second-order central difference

$$\frac{\partial f(x)}{\partial x^2} \approx f(x+1) - 2f(x) + f(x-1)$$

Corresponding convolution kernel:

1	-2	1

Example

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

$*$

-1	1

$h(i,j)$

$=$

$f(i,j)$ $g(i,j)$

Example

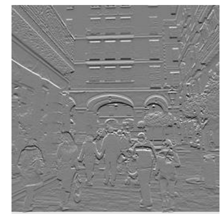
$$g(i,j) = f * h = \sum_{k,l} h(i-k, j-l) f(k,l)$$

$*$

-1
1

$h(i,j)$

$=$



$f(i,j)$ $g(i,j)$

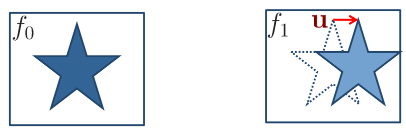
(Dense) Motion Estimation

- 2D motion

- 3D motion

Problem Statement

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

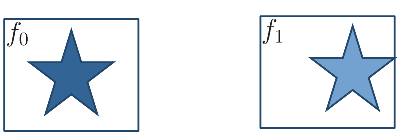


- For the moment, let's assume that the camera only moves in the xy-plane, i.e., $\mathbf{u} = (u\ v)^\top$
- Extension to 3D follows

General Idea

1. Define an error metric $E(\mathbf{u})$ that defines how well the two images match given a motion vector
2. Find the motion vector with the lowest error

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} E(\mathbf{u})$$



Error Metrics for Image Comparison

- Sum of Squared Differences (SSD)

$$E_{\text{SSD}}(\mathbf{u}) = \sum_i (f_1(\mathbf{x}_i + \mathbf{u}) - f_0(\mathbf{x}_i))^2 = \sum_i e_i^2$$

with displacement $\mathbf{u} = (u\ v)^\top$
and residual errors $e_i = f_1(\mathbf{x}_i + \mathbf{u}) - f_0(\mathbf{x}_i)$

Robust Error Metrics

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

$$E_{\mathrm{SRD}}(\mathbf{u}) = \sum_i \rho\left(f_1(\mathbf{x}_i + \mathbf{u}) - f_0(\mathbf{x}_i)\right) = \sum_i \rho(e_i)$$

Robust Error Metrics

- Sum of Absolute Differences

$$\rho_{\mathrm{SAD}}(e) = |e|$$

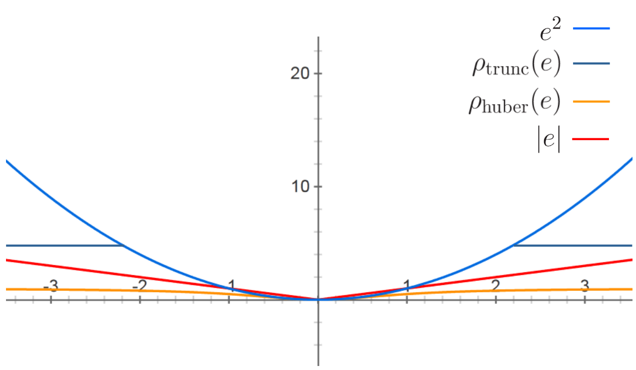
- Sum of truncated errors

$$\rho_{\mathrm{trunc}}(e) = \begin{cases} e^2 & \text{if } |e| < b \\ b^2 & \text{otherwise} \end{cases}$$

- Geman-McClure function (Huber norm)

$$\rho_{\mathrm{huber}}(e) = \frac{e^2}{1 + e^2/b^2}$$

Robust Error Metrics



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

$$E_{\mathrm{RMS}}(\mathbf{u}) = \sqrt{E_{\mathrm{SSD}}/A}$$

Exposure Differences

- Images might be taken with different exposure (auto shutter, white balance, …)
- Bias and gain model

$$f_1(\mathbf{x} + \mathbf{u}) = (1 + \alpha) f_0(\mathbf{x}) + \beta$$

- With SSD we get

$$E_{\mathrm{BG}}(\mathbf{u}) = \sum_i \left(f_1(\mathbf{x}_i + \mathbf{u}) - (1 + \alpha) f_0(\mathbf{x}_i) + \beta\right)^2$$
$$= \sum_i \alpha f_0(\mathbf{x}) + \beta - e_i^2$$

Cross-Correlation

- Maximize the product (instead of minimizing the differences)

$$E_{\mathrm{CC}}(\mathbf{u}) = -\sum_i f_0(\mathbf{x}_i) f_1(\mathbf{x}_i + \mathbf{u})$$

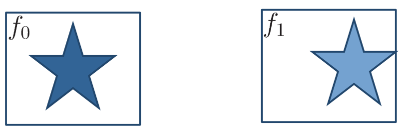
- Normalized cross-correlation (between -1..1)

$$E_{\mathrm{NCC}}(\mathbf{u}) =$$
$$-\sum_i \frac{(f_0(\mathbf{x}_i) - \mathrm{mean}\, f_0)(f_1(\mathbf{x}_i + \mathbf{u}) - \mathrm{mean}\, f_1)}{\sqrt{\mathrm{var}\, f_0 \mathrm{var}\, f_1}}$$

General Idea

1. Define an error metric $E(\mathbf{u})$ that defines how well the two images match given a motion vector
2. Find the motion vector with the lowest error

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} E(\mathbf{u})$$



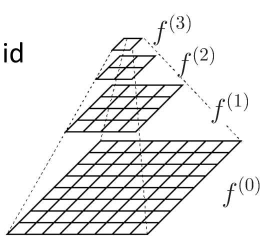
Finding the minimum

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

Hierarchical motion estimation

- Construct image pyramid

$$f_k^{(l+1)}(\mathbf{x}_i) \leftarrow f_k^{(l)}(2\mathbf{x}_i)$$



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

$$\hat{\mathbf{u}}^{(l-1)} \leftarrow 2\mathbf{u}^{(l)}$$

Gradient Descent

- Perform gradient descent on the SSD energy function (Lucas and Kanade, 1981)
- Taylor expansion of energy function

$$E_{\text{LK-SSD}}(\mathbf{u}+\Delta\mathbf{u}) = \sum_i (f_1(\mathbf{x}_i+\mathbf{u}+\Delta\mathbf{u}) - f_0(\mathbf{x}_i))^2$$

$$\approx \sum_i (f_1(\mathbf{x}_i+\mathbf{u}) + J_1(\mathbf{x}+\mathbf{u})\Delta\mathbf{u} - f_0(\mathbf{x}_i))^2$$

$$= \sum_i (J_1(\mathbf{x}+\mathbf{u})\Delta\mathbf{u} + e_i)^2$$

with $J_1(\mathbf{x}_i+\mathbf{u}) = \nabla f_1(\mathbf{x}_i+\mathbf{u}) = (\frac{\partial f_1}{\partial x}, \frac{\partial f_1}{\partial y})(\mathbf{x}_i+\mathbf{u})$

Least Squares Problem

- Goal: Minimize

$$E(\mathbf{u}+\Delta\mathbf{u}) = \sum_i (J_1(\mathbf{x}_i+\mathbf{u})\Delta\mathbf{u} + e_i)^2$$

- Solution: Compute derivative (and set to zero)

$$\frac{\partial E(\mathbf{u}+\Delta\mathbf{u})}{\partial \Delta\mathbf{u}} = 2A\Delta\mathbf{u} + 2\mathbf{b}$$

with $A = \sum_i J_1^\top(\mathbf{x}_i+\mathbf{u}) J_1(\mathbf{x}+\mathbf{u})$

and $\mathbf{b} = \sum_i e_i J_1^\top(\mathbf{x}_i+\mathbf{u})$

Least Squares Problem

1. Compute A,b from image gradients using

$$A = \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} \sum f_x f_t \\ \sum f_y f_t \end{pmatrix}$$

with $f_x = \frac{\partial f_1(\mathbf{x})}{\partial x}, f_y = \frac{\partial f_1(\mathbf{x})}{\partial y}$

and $f_t = \frac{\partial f_t(\mathbf{x})}{\partial t} [\approx f_1(\mathbf{x}) - f_0(\mathbf{x})]$

2. Solve $A\Delta\mathbf{u} = -\mathbf{b}$

$$\Rightarrow \Delta\mathbf{u} = -A^{-1}\mathbf{b}$$

All of these computation are super fast!

Covariance of the Estimated Motion

- Assuming (small) Gaussian noise in the images

$$f_{\text{obs}}(\mathbf{x}_i) = f_{\text{true}}(\mathbf{x}_i) + \epsilon_i$$

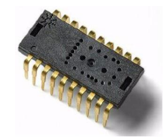
with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

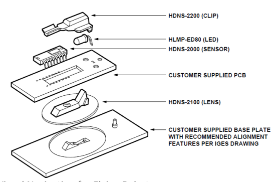
- … results in uncertainty in the motion estimate with covariance (e.g., useful for Kalman filter)

$$\Sigma_u = \sigma^2 A^{-1}$$

Optical Computer Mouse (since 1999)

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





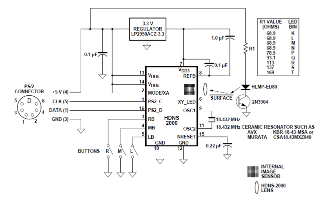
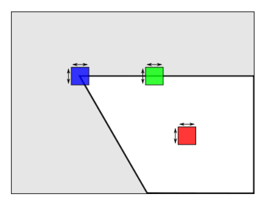


Image Patches

- Sometimes we are interested of 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?

Image Patches

- Sometimes we are interested of the motion of a small image patches
- **Problem:** some patches are easier to track than others



Example

- Let's look at the shape of the energy functional

Corner Detection

$$A = \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix}$$

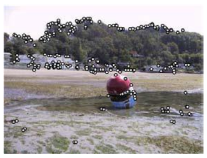
- **Idea:** Inspect eigenvalues λ_1, λ_2 of Hessian A
  - λ_1, λ_2 small → no point of interest
  - λ_1 large, λ_2 small → edge
  - λ_1, λ_2 large → corner
- Harris detector (does not need eigenvalues)

$$\lambda_1 \lambda_2 > \kappa (\lambda_1 + \lambda_2)^2 \Leftrightarrow \det(A) > \kappa \operatorname{trace}^2(A)$$

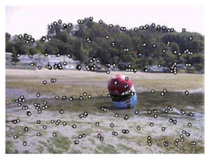
- Shi-Tomasi (or Kanade-Lucas) $\min(\lambda_1, \lambda_2) > \kappa$

Corner Detection

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



strongest responses



non-maximal suppression

Other Detectors

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

Kanade-Lucas-Tomasi (KLT) Tracker

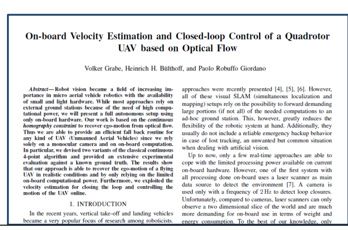
- Algorithm
  1. Find (Shi-Tomasi) corners in first frame and initialize tracks
  2. Track from frame to frame
  3. Delete track if error exceeds threshold
  4. Initialize additional tracks when necessary
  5. Repeat step 2-4
- KLT tracker is highly efficient (real-time on CPU) but provides only sparse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

Example

3D Motion Estimation

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

- Research paper: Grabe et al., ICRA 2012
  http://www9.in.tum.de/~sturmju/dirs/icra2012/data/papers/2025.pdf



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

***Abstract*— Robot vision became a field of increasing importance in micro aerial vehicle robotics with the availability of small and light hardware. While most approaches rely on external ground stations because of the need of high computational power, we will present a full autonomous setup using only on-board hardware. Our work is based on the *continuous homography constraint* to recover ego-motion from optical flow. Thus we are able to provide an efficient fall back routine for any kind of UAV (Unmanned Aerial Vehicles) since we rely solely on a monocular camera and on on-board computation. In particular, we devised two variants of the classical continuous 4-point algorithm and provided an extensive experimental evaluation against a known ground truth. The results show that our approach is able to recover the ego-motion of a flying UAV in realistic conditions and by only relying on the limited on-board computational power. Furthermore, we exploited the velocity estimation for closing the loop and controlling the motion of the UAV online.**

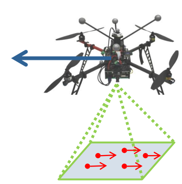
I. INTRODUCTION

In the recent years, vertical take-off and landing vehicles became a very popular focus of research among roboticists. approaches were recently presented [4], [5], [6]. However, all of these visual SLAM (simultaneous localization and mapping) setups rely on the possibility to forward demanding large portions (if not all) of the needed computations to an ad-hoc ground station. This, however, greatly reduces the flexibility of the robotic system at hand. Additionally, they usually do not include a reliable emergency backup behavior in case of lost tracking, an unwanted but common situation when dealing with artificial vision.

Up to now, only a few real-time approaches are able to cope with the limited processing power available on current on-board hardware. However, one of the first system with all processing done on-board uses a laser scanner as main data source to detect the environment [7]. A camera is used only with a frequency of 2 Hz to detect loop closures. Unfortunately, compared to cameras, laser scanners can only observe a two dimensional slice of the world and are much more demanding for on-board use in terms of weight and energy consumption. To the best of our knowledge, only

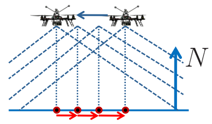
Approach [Grabe et al., ICRA'12]

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

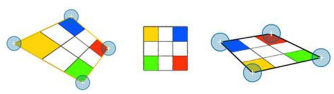


Assumptions

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

$$\mathbf{p} \in \mathbb{R}^3$$

- Moving camera with twist

$$\boldsymbol{\xi} = (\mathbf{v}^\top, \boldsymbol{\omega}^\top)^\top \in \mathbb{R}^6$$

- **Apparent velocity** of the point in camera frame

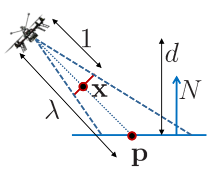
$$\dot{\mathbf{p}} = [\boldsymbol{\omega}]_\times \mathbf{p} + \boldsymbol{v}$$

Continuous Homography Matrix

- Assumption: All feature points are located on a plane

$$N^\top \mathbf{p} = d$$

  with plane normal $N \in \mathbb{R}^3$
  and distance $d \in \mathbb{R}$



Continuous Homography Matrix

- Rewrite this to $\frac{1}{d} N^\top \mathbf{p} = 1$ and plug it into the equation for the apparent velocity, we obtain

$$\dot{\mathbf{p}} = [\boldsymbol{\omega}]_\times \mathbf{p} + \boldsymbol{v} \frac{1}{d} N^\top \mathbf{p} = \underbrace{\left([\boldsymbol{\omega}]_\times + \boldsymbol{v} \frac{1}{d} N^\top \right)}_{H \in \mathbb{R}^{3\times 3}} \mathbf{p} = H\mathbf{p}$$

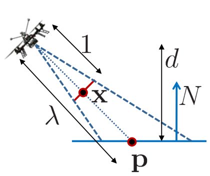
- H is called the **continuous homography matrix**
- Note: H contains both the linear/angular velocity $(\boldsymbol{v}, \boldsymbol{\omega})$ and the scene structure (N, d)

Continuous Homography Constraint

- The camera observes point $\mathbf{p} \in \mathbb{R}^3$ at pixel $\mathbf{x} \in \mathbb{R}^2$ (assuming $K = I$ for simplicity)

$$\tilde{\mathbf{x}} = \lambda \bar{\mathbf{x}} = \mathbf{p}$$

- The KLT tracker estimates the motion $\mathbf{u}$ of the feature track in the image
- Constraint: $\boxed{\mathbf{u} = \dot{\mathbf{x}}}$



Continuous Homography Constraint

- We now have
  1. $\dot{\mathbf{p}} = H\mathbf{p}$
  2. $\dot{\mathbf{p}} = \dot{\lambda}\bar{\mathbf{x}} + \lambda\bar{\mathbf{u}}$ (time derivative of $\mathbf{p} = \lambda\bar{\mathbf{x}}$ and the optical flow constraint $\mathbf{u} = \dot{\mathbf{x}}$)
- Let's combine these two formulas…

Continuous Homography Constraint

- Combining these formulas gives us

$$\dot{\lambda}\bar{\mathbf{x}} + \lambda\bar{\mathbf{u}} = H\mathbf{p}$$
$$\lambda\bar{\mathbf{u}} = H\mathbf{p} - \dot{\lambda}\bar{\mathbf{x}}$$
$$\bar{\mathbf{u}} = H\bar{\mathbf{x}} - \tfrac{\dot{\lambda}}{\lambda}\bar{\mathbf{x}}$$

- Multiply both sides with $[\bar{\mathbf{x}}]_\times$ gives us

$$[\bar{\mathbf{x}}]_\times\bar{\mathbf{u}} = [\bar{\mathbf{x}}]_\times H\bar{\mathbf{x}} - \underbrace{[\bar{\mathbf{x}}]_\times\tfrac{\dot{\lambda}}{\lambda}\bar{\mathbf{x}}}_{=0}$$

$$\boxed{\Rightarrow [\bar{\mathbf{x}}]_\times\bar{\mathbf{u}} = [\bar{\mathbf{x}}]_\times H\bar{\mathbf{x}}}$$

Approach

- **Result:** For all observed motions in the image, the continuous homography constraint holds

$$[\bar{\mathbf{x}}]_\times\bar{\mathbf{u}} = [\bar{\mathbf{x}}]_\times H\bar{\mathbf{x}}$$

- How can we use this to estimate the camera motion?!

Approach

- **Result:** For all observed motions in the image, the continuous homography constraint holds

$$[\bar{\mathbf{x}}]_\times\bar{\mathbf{u}} = [\bar{\mathbf{x}}]_\times H\bar{\mathbf{x}}$$

- How can we use this to estimate the camera motion?
  1. Estimate H from at least 4 feature tracks
  2. Recover $(\boldsymbol{v}, \boldsymbol{\omega})$ and (N, d) from H

Remember: $H = [\boldsymbol{\omega}]_\times + \boldsymbol{v}\frac{1}{d}N^\top$

Step 1: Estimate H

- Continuous homography constraint

$$[\bar{\mathbf{x}}]_\times H\bar{\mathbf{x}} = [\bar{\mathbf{x}}]_\times\bar{\mathbf{u}}$$

- Stack matrix H as a vector $\mathbf{h} \in \mathbb{R}^9$ and rewrite

$$M^\top\mathbf{h} = [\bar{\mathbf{x}}]_\times\bar{\mathbf{u}}$$

→Linear system of equations

- For several feature tracks

$$\begin{pmatrix} M_1^\top \\ M_2^\top \\ \vdots \end{pmatrix}\mathbf{h} = \begin{pmatrix} [\bar{\mathbf{x}}]_\times\bar{\mathbf{u}}_1^\top \\ [\bar{\mathbf{x}}]_\times\bar{\mathbf{u}}_2^\top \\ \vdots \end{pmatrix}$$

Step 1: Estimate H

- Linear set of equations

$$\underbrace{\begin{pmatrix} M_1^\top \\ M_2^\top \\ \vdots \end{pmatrix}}_{A}\mathbf{h} = \underbrace{\begin{pmatrix} [\bar{\mathbf{x}}]_\times\bar{\mathbf{u}}_1^\top \\ [\bar{\mathbf{x}}]_\times\bar{\mathbf{u}}_2^\top \\ \vdots \end{pmatrix}}_{\mathbf{b}}$$

- Solve for $\mathbf{h}$ using least squares

$$A\mathbf{h} = \mathbf{b}$$
$$\Rightarrow \mathbf{h} = (A^\top A)^{-1}A^\top\mathbf{b}$$

Step 2: Recover camera motion

Grabe et al. investigated three alternatives:

1. Recover $(\boldsymbol{\omega}, \frac{\boldsymbol{v}}{d}, N)$ from $H = [\boldsymbol{\omega}]_\times + \boldsymbol{v}\frac{1}{d}N^\top$ using the 8-point algorithm (not yet explained)
2. Use angular velocity $\boldsymbol{\omega}$ from IMU to de-rotate observed feature tracks beforehand, then:
$$H = \boldsymbol{v}\tfrac{1}{d}N^\top$$
3. Additionally use gravity vector from IMU as plane normal $N = N_{\text{IMU}}$, then
$$\tfrac{\boldsymbol{v}}{d} = H(N^\top N)^{-1}$$

Evaluation

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

Algorithm	Norm error	Std. deviation
Pure vision	0.134 $\frac{m}{s}$	0.094 $\frac{m}{s}$
Ang. vel. known	0.117 $\frac{m}{s}$	0.093 $\frac{m}{s}$
Normal known	0.113 $\frac{m}{s}$	0.088 $\frac{m}{s}$

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

Algorithm	Norm error	Std. deviation
Pure vision	0.084 $\frac{m}{s}$	0.139 $\frac{m}{s}$
Ang. vel. known	0.039 $\frac{m}{s}$	0.042 $\frac{m}{s}$
Normal known	0.028 $\frac{m}{s}$	0.031 $\frac{m}{s}$

Visual Velocity Control

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

[Grabe et al., ICRA '12]

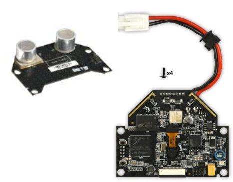
Landing on a Moving Platform

- Similar approach, but with offboard computation

[Herissé et al., T-RO '12]

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]

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

Simultaneous Localization and Mapping (SLAM)

Dr. Jürgen Sturm

Organization: Exam Dates

- Registration deadline: June 30
- Course ends: July 19
- Examination dates: August 9+14 (Thu+Tue)
  - Oral team exam
  - Sign up for a time slot starting from now
  - List placed on blackboard in front of our secretary

VISNAV Oral Team Exam

Date and Time	Student Name	Student Name	Student Name
Tue, Aug. 9, 10am			
Tue, Aug. 9, 11am			
Tue, Aug. 9, 2pm			
Tue, Aug. 9, 3pm			
Tue, Aug. 9, 4pm			
Thu, Aug. 14, 10am			
Thu, Aug. 14, 11am			
Thu, Aug. 14, 2pm			
Thu, Aug. 14, 3pm			
Thu, Aug. 14, 4pm			

The SLAM Problem

SLAM is the process by which a robot **builds a map** of the environment and, at the same time, uses the map to **compute its location**

- Localization: inferring location given a map
- Mapping: inferring a map given a location

The SLAM Problem

Given:

- The robot's controls $\mathbf{u}_{1:t} = < \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_t >$
- (Relative) observations $\mathbf{z}_{1:t} = < \mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t >$

Wanted:

- Map of features $\mathbf{m} = < \mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_k >$
- Trajectory of the robot $\mathbf{x}_{1:t} = < \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t >$

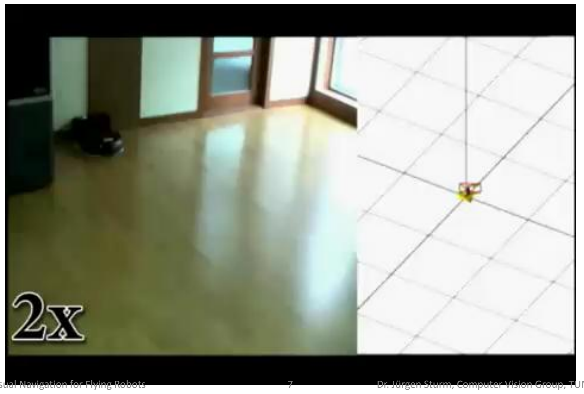
SLAM Applications

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

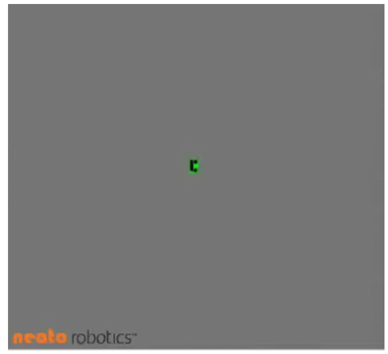
Examples

- 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)



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



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

SLAM vs. SfM

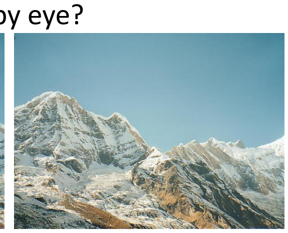
- In Robotics: Simultaneous Localization and Mapping (SLAM)
  - Laser scanner, ultrasound, monocular/stereo camera
  - Typically in combination with an odometry sensor
  - Typically pre-calibrated sensors
- In Computer Vision: Structure from Motion (SfM), sometimes: Structure and Motion
  - Monocular/stereo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

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

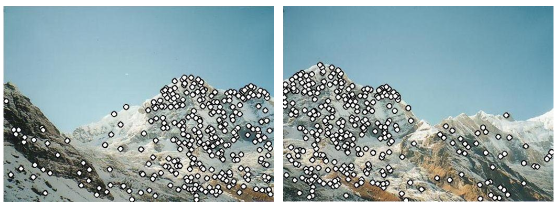
How Do We Build a Panorama Map?

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

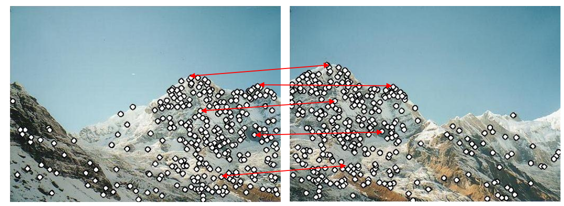
Matching with Features

- Detect features in both images



Matching with Features

- Detect features in both images
- Find corresponding pairs

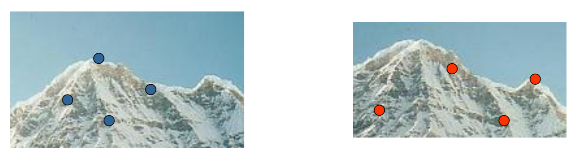


Matching with Features

- Detect features in both images
- Find corresponding pairs
- Use these pairs to align images

Matching with Features

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



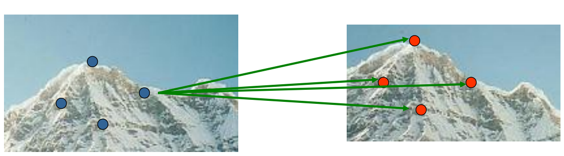
no chance to match!

→ We need a reliable detector

Matching with Features

- Problem 2:
  For each point correctly recognize the corresponding one

?



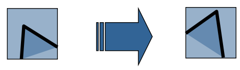
→ We need a reliable and distinctive descriptor

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

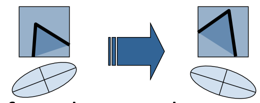
Harris Detector

- Rotation invariance?



Harris Detector

- Rotation invariance?

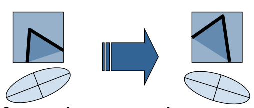


- Remember from last week

$$A = \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix} \qquad R = \lambda_1 \lambda_2 - \kappa \left(\lambda_1 + \lambda_2\right)^2$$

Harris Detector

- Rotation invariance



- Remember from last week

$$A = \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix} \qquad R = \lambda_1 \lambda_2 - \kappa \left(\lambda_1 + \lambda_2\right)^2$$

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

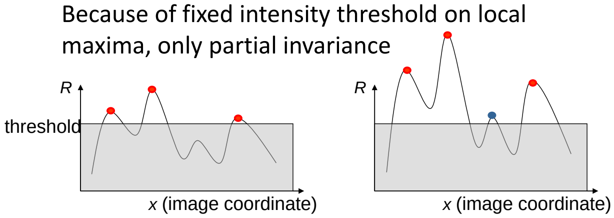
→ Corner response R is invariant to rotation

Harris Detector

- Invariance to intensity change?

Harris Detector

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

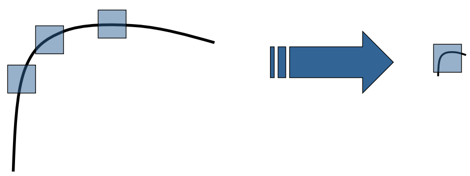


Harris Detector

- Invariant to scaling?

Harris Detector

- Not invariant to image scale

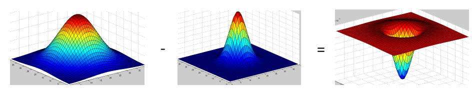


All points classified as edge

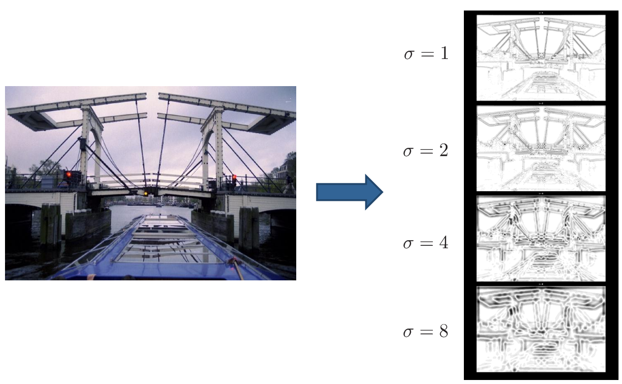
Point classified as corner

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

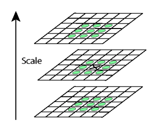


Example: Difference of Gaussians

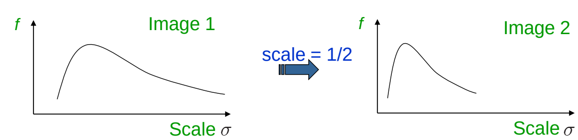


SIFT Detector

- Search for local maximum in space and scale



- Corner detections are invariant to scale change

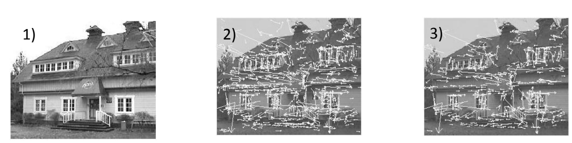


SIFT Detector

1. Detect maxima in scale-space
2. Non-maximum suppression
3. Eliminate edge points (check ratio of eigenvalues)
4. For each maximum, fit quadratic function and compute center at sub-pixel accuracy

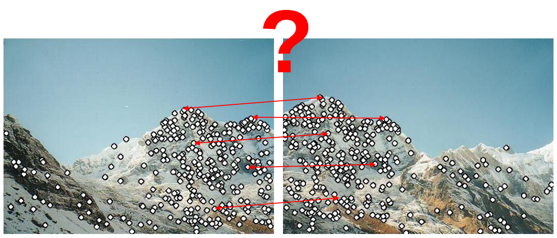
Example

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



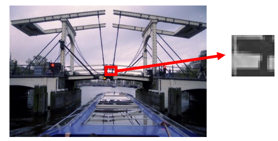
Feature Matching

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

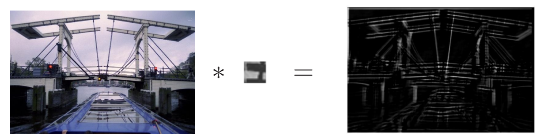


Template Convolution

- Extract a small as a template



- Convolve image with this template



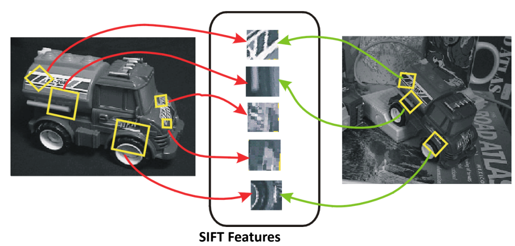
Template Convolution

Invariances

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

Scale Invariant Feature Transform (SIFT)

- Lowe, 2004: Transform patches into a canonical form that is invariant to translation, rotation, scale, and other imaging parameters



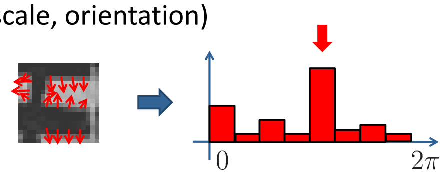
Scale Invariant Feature Transform (SIFT)

Approach

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

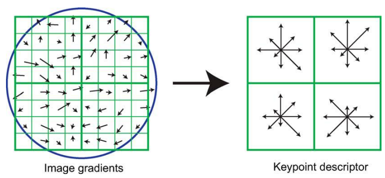
Select Dominant Orientation

- 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_1, how to find best match in I_2?

- Define distance function that compares two features
- Test all the features in I_2, find the one with the minimal distance

Feature Distance

How to define the difference between features?

- Simple approach is Euclidean distance (or SSD)

$$\mathrm{d}(\mathbf{d}_1, \mathbf{d}_2) = \|\mathbf{d}_1 - \mathbf{d}_2\|$$

Feature Distance

How to define the difference between features?

- Simple approach is Euclidean distance (or SSD)

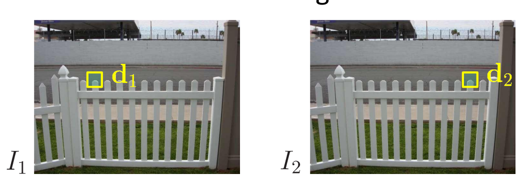
$$\mathrm{d}(\mathbf{d}_1, \mathbf{d}_2) = \|\mathbf{d}_1 - \mathbf{d}_2\|$$

- Problem: can give good scores to ambiguous (bad) matches

Feature Distance

How to define the difference between features?

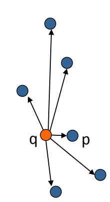
- Better approach $\mathrm{d}(\mathbf{d}_1, \mathbf{d}_2) = \|\mathbf{d}_1 - \mathbf{d}_2\| / \|\mathbf{d}_1 - \mathbf{d}_2'\|$
  with $\mathbf{d}_2$ best matching feature from I_2
  $\mathbf{d}_2'$ second best matching feature from I_2
- Gives small values for ambiguous matches



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

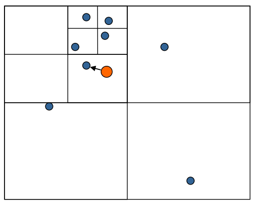
- Exhaustive search $\mathrm{O}(n^2)$



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

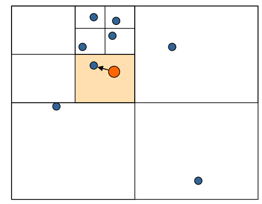
- Exhaustive search $O(n^2)$
- Indexing (k-d tree)



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

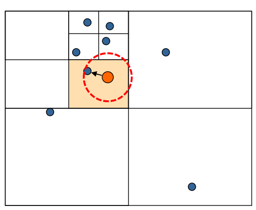
- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

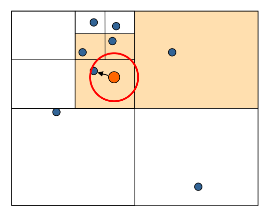
- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

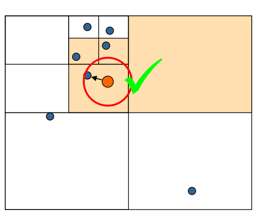
- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

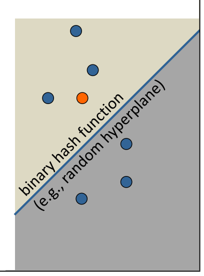
- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found
  - Best-bin-first: use priority queue for unchecked leafs



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

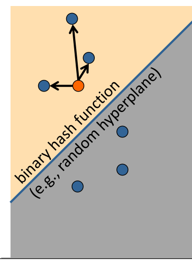
- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
- Approximate search
  - Locality sensitive hashing
  - Approximate nearest neighbor



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
- 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 $\mathrm{O}(n^2)$
- Indexing (k-d tree)
- Approximate search
- Vocabulary trees

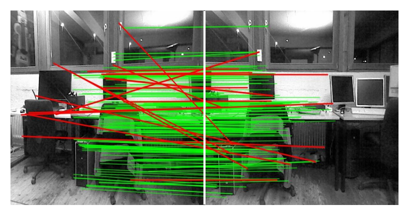
Other Descriptors

- 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)



I_1 I_2

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 $\mathbf{x}_i \leftrightarrow \mathbf{p}_i$
- **Wanted:** $M = K(R \;\; \mathbf{t})$
  such that $\tilde{\mathbf{x}}_i = M\mathbf{p}_i$
- The algorithm has two parts:
  1. Compute $M \in \mathbb{R}^{3\times 4}$
  2. Decompose M into $K, R, \mathbf{t}$ via QR decomposition

Step 1: Estimate M

- $\tilde{\mathbf{x}}_i = M\mathbf{p}_i$
- Each correspondence generates two equations

$$x = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}W}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}W} \qquad y = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}W}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}W}$$

- Multiplying out gives equations **linear** in the elements of M

$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34}W)x = m_{11}X + m_{12}Y + m_{13}Z + m_{14}W$$
$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34}W)y_j = m_{21}X + m_{22}Y + m_{23}Z + m_{24}W$$

- Re-arrange in matrix form…

Step 1: Estimate M

- Re-arranged in matrix form

$$\begin{pmatrix} X & Y & Z & 1 & 0 & 0 & 0 & 0 & -xX & -xY & -xZ & -x \\ 0 & 0 & 0 & 0 & X & Y & Z & 1 & -yX & -yY & -yZ & -y \end{pmatrix} \mathbf{m} = \mathbf{0}$$

  with $\mathbf{m} = (m_{11} \;\; m_{12} \;\; \ldots \;\; m_{34}) \in \mathbb{R}^{12}$
- Concatenate equations for n≥6 correspondences

$$A\mathbf{m} = \mathbf{0}$$

- Wanted vector $\mathbf{m}$ 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 $M = K(R \;\; \mathbf{t})$
- The first 3x3 submatrix is the product of an upper triangular and orthogonal (rot.) matrix

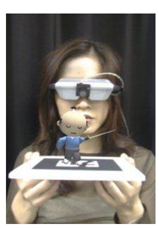
$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Procedure:

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

Example: ARToolkit Markers (1999)

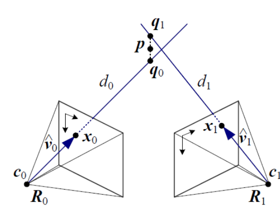
1. Threshold image
2. Detect edges and fit lines
3. Intersect lines to obtain corners
4. Estimate projection matrix M
5. Extract camera pose R,t (assume K is known)



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

Triangulation

- **Given:** cameras $\{M_j = K_j(R_j \;\; \mathbf{t}_j)\}$
  point correspondence $\mathbf{x}_0, \mathbf{x}_1$
- **Wanted:** Corresponding 3D point $\mathbf{p}$



Triangulation

- Where do we expect to see $\mathbf{p} = (X\ Y\ Z\ W)^\top$?

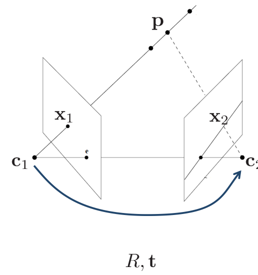
$$\hat{x} = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}W}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}W} \qquad \hat{y} = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}W}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}W}$$

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

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \sum_j d(\mathbf{x}_j, \hat{\mathbf{x}}_j)^2$$

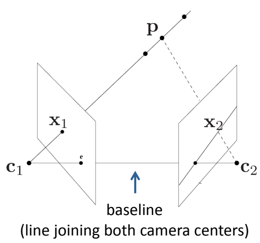
Epipolar Geometry

- Consider two cameras that observe a 3D world point



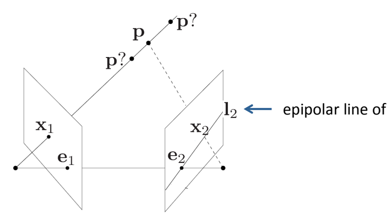
Epipolar Geometry

- The line connecting both camera centers is called the **baseline**



Epipolar Geometry

- Given the image of a point in one view, what can we say about its position in another?

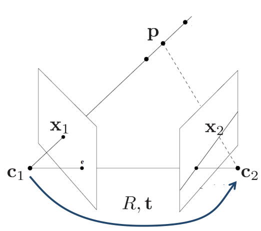


- A point in one image "generates" a line in another image (called the **epipolar line**)

Epipolar Geometry

- Left line in left camera frame $\mathbf{p}_1 = d_1\hat{\mathbf{x}}_1$
- Right line in right camera frame $\mathbf{p}_2 = d_2\hat{\mathbf{x}}_2$

where $\hat{\mathbf{x}}_j = K^{-1}\bar{\mathbf{x}}_j$ are the (local) ray directions



Epipolar Geometry

- Left line in **right** camera frame $\mathbf{p}_1' = Rd_1\hat{\mathbf{x}}_1 + t$
- Right line in right camera frame $\mathbf{p}_2 = d_2\hat{\mathbf{x}}_2$

where $\hat{\mathbf{x}}_j = K^{-1}\bar{\mathbf{x}}_j$ are the (local) ray directions

- Intersection of both lines

$$d_2\hat{\mathbf{x}}_2 = Rd_1\hat{\mathbf{x}}_1 + \mathbf{t} \qquad |[\mathbf{t}]_\times \cdot$$

$$d_2[\mathbf{t}]_\times\hat{\mathbf{x}}_2 = d_1[\mathbf{t}]_\times R\hat{\mathbf{x}}_1 + \underbrace{\cancel{[\mathbf{t}]_\times\mathbf{t}}}_{=0} \qquad |\hat{\mathbf{x}}_2^\top \cdot$$

$$\underbrace{\cancel{d_2\hat{\mathbf{x}}_2^\top[\mathbf{t}]_\times\hat{\mathbf{x}}_2}}_{0=} = d_1\hat{\mathbf{x}}_2^\top[\mathbf{t}]_\times R\hat{\mathbf{x}}_1$$

$$0 = \hat{\mathbf{x}}_2^\top[\mathbf{t}]_\times R\hat{\mathbf{x}}_1$$

$$\boxed{0 = \hat{\mathbf{x}}_2^\top E\hat{\mathbf{x}}_1}$$

this is called the epipolar constraint

Epipolar Geometry

Note: The epipolar constraint holds for **every** pair of corresponding points $\mathbf{x}_1, \mathbf{x}_2$

$$\hat{\mathbf{x}}_2^\top E \hat{\mathbf{x}}_1 = 0$$

where E is called the essential matrix

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

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 $\hat{\mathbf{x}}_2^\top E \hat{\mathbf{x}}_1 = 0$
- Written out (with $\mathbf{x}_j = (x_j, y_j, 1)^\top$)

$$\begin{array}{ccccccc} x_1x_2e_{11} & + & y_1x_2e_{12} & + & x_2e_{13} & + & \\ x_1y_2e_{21} & + & y_1y_2e_{22} & + & y_2e_{23} & + & \\ x_1e_{31} & + & y_1e_{32} & + & 1e_{33} & = & 0 \end{array}$$

- Stack the elements into two vectors

$$\left.\begin{array}{l} \mathbf{z} = (x_1x_2 \;\; y_1x_2 \;\; \dots \;\; 1)^\top \\ \mathbf{e} = (e_{11} \;\; e_{12} \;\; \dots \;\; e_{33})^\top \end{array}\right\} \mathbf{z}^\top \mathbf{e} = 0$$

Step 1: Estimate E

- Each correspondence gives us one constraint

$$\left.\begin{array}{c} \mathbf{z}_1^\top \mathbf{e} = 0 \\ \mathbf{z}_2^\top \mathbf{e} = 0 \\ \vdots \\ \mathbf{z}_n^\top \mathbf{e} = 0 \end{array}\right\} Z\mathbf{e} = \mathbf{0}$$

- Linear system with *n* equations
- e is in the null-space of Z
- Solve using SVD (assuming $\|\mathbf{e}\| = 1$)

Normalized 8-Point Algorithm

[Hartley 1997]

- Noise in the point observations is unequally distributed in the constraints, e.g.,

double noise

$$\begin{array}{ccccccc} x_1x_2e_{11} & + & y_1x_2e_{12} & + & x_2e_{13} & + & \\ x_1y_2e_{21} & + & y_1y_2e_{22} & + & y_2e_{23} & + & \\ x_1e_{31} & + & y_1e_{32} & + & 1e_{33} & = & 0 \end{array}$$

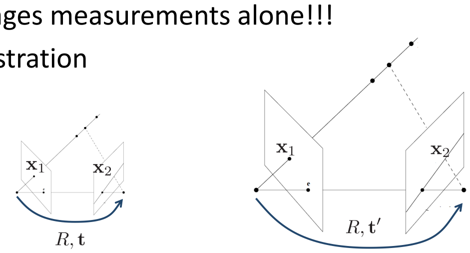
normal noise

noise free

- 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 $\hat{\mathbf{t}}$ up to scale

Step 2a: Recover t

- Remember: $E = [\mathbf{t}]_\times R$
- Therefore, $\mathbf{t}^\top$ is in the null space of E

$$\mathbf{t}^\top E = \underbrace{\mathbf{t}^\top [\mathbf{t}]_\times}_{=0} R = 0$$

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

$$E = [\hat{\mathbf{t}}]_\times R = U \Sigma V^\top$$

$$= \begin{pmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \boxed{\hat{\mathbf{t}}} \end{pmatrix} \begin{pmatrix} 1 & & \\ & 1 & \\ & & \boxed{0} \end{pmatrix} \begin{pmatrix} \mathbf{v}_0^\top & \mathbf{v}_1^\top & \mathbf{v}_2^\top \end{pmatrix}$$

Step 2b: Recover R

Remember, the cross-product $[\hat{\mathbf{t}}]_\times$

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

$$[\hat{\mathbf{t}}]_\times = S Z R_{90^\circ} S^\top$$

$$= \begin{pmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \hat{\mathbf{t}} \end{pmatrix} \begin{pmatrix} 1 & & \\ & 1 & \\ & & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 & \\ 1 & 0 & \\ & & 1 \end{pmatrix} \begin{pmatrix} \mathbf{s}_0 \\ \mathbf{s}_1 \\ \hat{\mathbf{t}} \end{pmatrix}$$

Step 2b: Recover R

- Plug this into the essential matrix equation

$$E = [\mathbf{t}]_\times R = S Z R_{90^\circ} S^\top R = U \Sigma V^\top$$

=

=

- By identifying $S = U$ and $Z = \Sigma$, we obtain

$$R_{90^\circ} U^\top R = V^\top$$

$$\boxed{R = U R_{90^\circ}^\top V^\top}$$

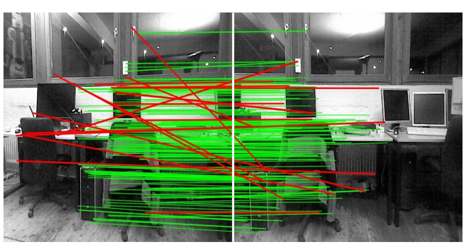
Summary: 8-Point Algorithm

Given: Image pair

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

- Compute correspondences
- Compute essential matrix
- Extract camera motion

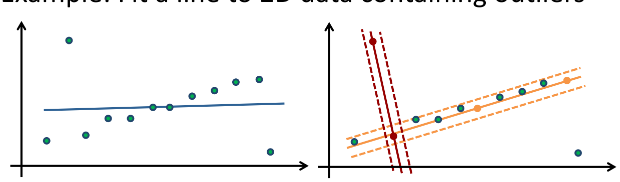
How To Deal With Outliers?



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 $\arg\min_l \sum_i d_i^2$
2. **Classify** the data into inliers (valid points) and outliers (using some threshold)

RANdom SAmple Consensus (RANSAC)

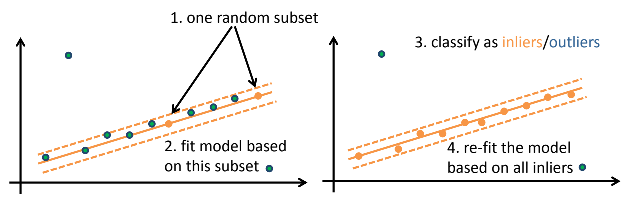
[Fischler and Bolles, 1981]

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

RANdom SAmple Consensus (RANSAC)



- RANSAC is used very widely
- Many improvements/variants, e.g., MLESAC:

$$\arg\min_{l} \sum_i \rho(d_i) \text{ with } \rho(d) = \begin{cases} d^2 & \text{if } d \leq e(inlier) \\ e^2 & \text{if } d > e(outlier) \end{cases}$$

How Many Samples?

- For probability p of having no outliers, we need to sample $N = \dfrac{\log(1-p)}{\log(1-(1-\epsilon)^s)}$ subsets

for subset size s and outlier ratio ϵ

- E.g., for p=0.95:

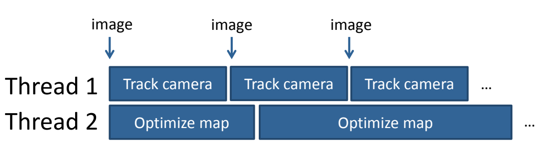
Sample size	Proportion of outliers ϵ						
s	5%	10%	20%	25%	30%	40%	50%
2	2	2	3	4	5	7	11
3	2	3	5	6	8	13	23
4	2	3	6	8	11	22	47
5	3	4	8	12	17	38	95
6	3	4	10	16	24	63	191
7	3	5	13	21	35	106	382
8	3	6	17	29	51	177	766

Two Examples

- PTAM
  G. Klein and D. Murray, Parallel Tracking and Mapping for Small AR Workspaces, International Symposium on Mixed and Augmented Reality (ISMAR), 2007
  http://www.robots.ox.ac.uk/~gk/publications/KleinMurray2007ISMAR.pdf
- Photo Tourism
  N. Snavely, S. M. Seitz, R. Szeliski, Photo tourism: Exploring photo collections in 3D, ACM Transactions on Graphics (SIGGRAPH), 2006
  http://phototour.cs.washington.edu/Photo_Tourism.pdf

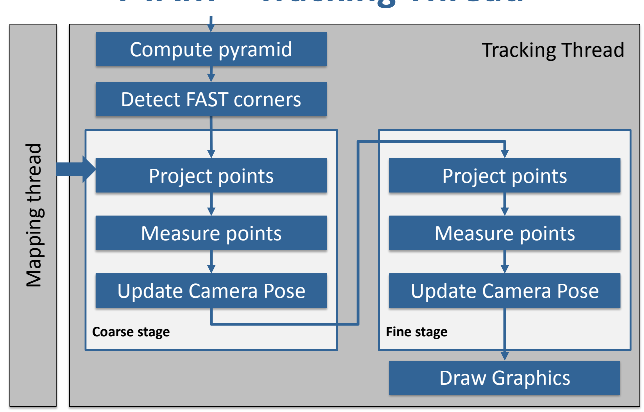
PTAM (2007)

- Architecture optimized for dual cores



- Tracking thread runs in real-time (30Hz)
- Mapping thread is not real-time

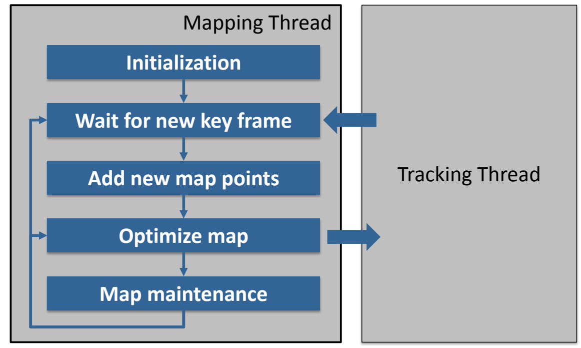
PTAM – Tracking Thread



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

PTAM – 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	270 ms	440 ms
Global Bundle Adjustment	380 ms	1.7 s	6.9 s

PTAM Video

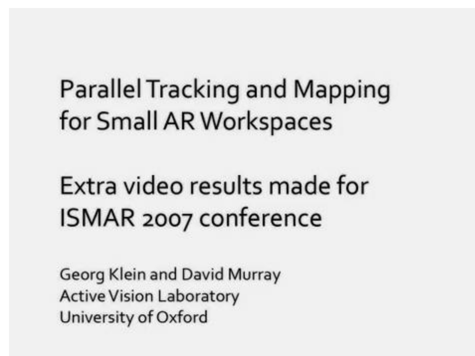


Photo Tourism (2006)

- Overview

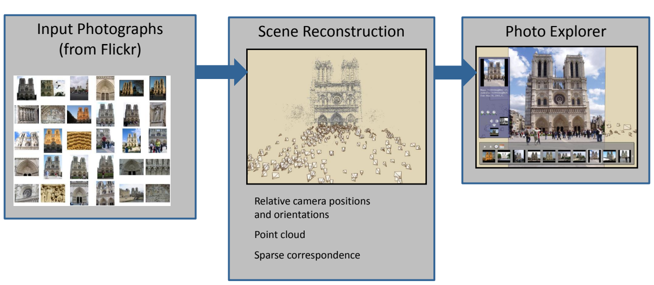
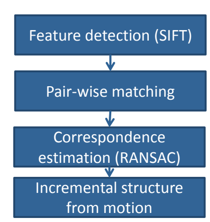


Photo Tourism – Scene Reconstruction

- Processing pipeline



- Automatically estimate
  - Position, orientation and focal length of all cameras
  - 3D positions of point features

Photo Tourism – Input Images

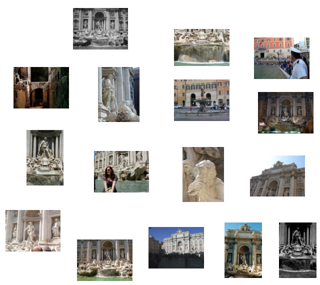


Photo Tourism – Feature Detection

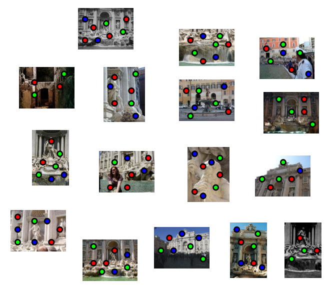
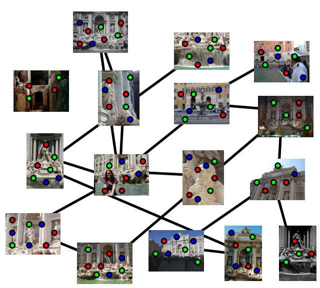


Photo Tourism – Feature Matching



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

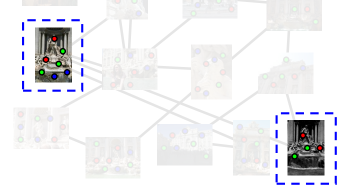
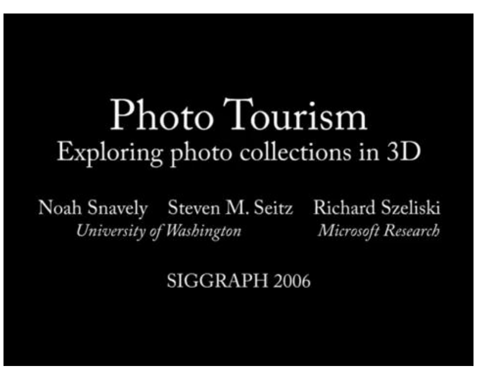


Photo Tourism – Video



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

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

Bundle Adjustment and Stereo Correspondence

Dr. Jürgen Sturm

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

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

$$M = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \in \mathrm{SE}(3) \subset \mathbb{R}^{4\times 4}$$

Pro: easy to concatenate and invert
Con: not minimal

- Representation as a twist coordinates

$$\boldsymbol{\xi} = (v_x \quad v_y \quad v_z \quad \omega_x \quad \omega_y \quad \omega_z)^\top \in \mathbf{R}^6$$

Pro: minimal
Con: need to convert to matrix for concatenation and inversion

Remember: 3D Transformations

- From twist coordinates to twist

$$\hat{\boldsymbol{\xi}} = \begin{pmatrix} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \in \mathrm{se}(3)$$

- Exponential map between se(3) and SE(3)

$$M = \exp \hat{\boldsymbol{\xi}} \qquad \hat{\boldsymbol{\xi}} = \log M$$

alternative notation: $M = \exp[\boldsymbol{\xi}]^\wedge \qquad \boldsymbol{\xi} = [\log M]^\vee$

Remember: Rodrigues' formula

- **Given:** Twist coordinates

$$\boldsymbol{\xi} = (\boldsymbol{\omega}^\top, \boldsymbol{v}^\top)^\top = (\omega_x, \omega_y, \omega_z, v_x, v_y, v_z)^\top$$
$$= (t\bar{\boldsymbol{\omega}}^\top, \boldsymbol{v}^\top)^\top \text{ with } \|\bar{\boldsymbol{\omega}}\| = 1, t = \|\boldsymbol{\omega}\|$$

- **Return:** Homogeneous transformation

$$R = I + [\bar{\boldsymbol{\omega}}]_\times \sin(t) + [\bar{\boldsymbol{\omega}}]_\times^2 (1 - \cos t)$$
$$\mathbf{t} = (I - R)[\bar{\boldsymbol{\omega}}]_\times \boldsymbol{v} + \bar{\boldsymbol{\omega}}\bar{\boldsymbol{\omega}}^\top \boldsymbol{v} t$$
$$M = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}$$

Notation

- Camera poses in a minimal representation (e.g., twists)

$$\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n$$

- … as transformation matrices

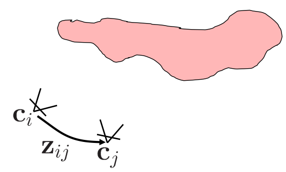
$$M_1, M_2, \ldots, M_n$$

- … as rotation matrices and translation vectors

$$(R_1, \mathbf{t}_1), (R_2, \mathbf{t}_2), \ldots, (R_n, \mathbf{t}_n)$$

Incremental Motion Estimation

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

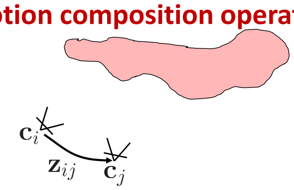


Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame
- **Motion concatenation (for twists)**

$$\hat{\mathbf{c}}_j = \log\left(\exp \hat{\mathbf{c}}_i \exp \hat{\mathbf{z}}_{ij}\right)$$

- **Motion composition operator (in general)**

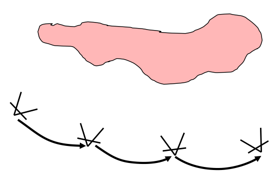


$$\mathbf{c}_j = \mathbf{c}_i \oplus \mathbf{z}_{ij}$$

$$\mathbf{z}_{ij} = \mathbf{c}_j \ominus \mathbf{c}_i$$

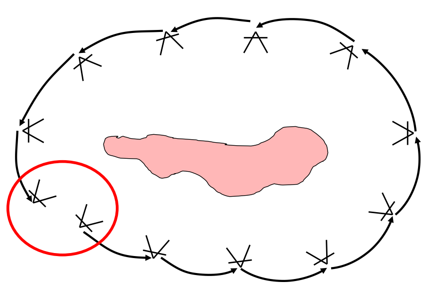
Incremental Motion Estimation

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



Incremental Motion Estimation

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

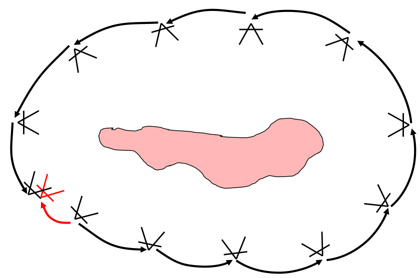


Loop Closures

- **Idea:** Estimate camera motion from frame to frame
- **Problem:**
  - Estimates are inherently noisy
  - Error accumulates over time → drift

Incremental Motion Estimation

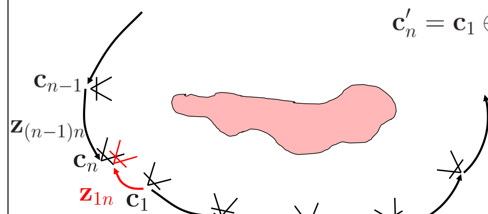
- **Idea:** Estimate camera motion from frame to frame



Incremental Motion Estimation

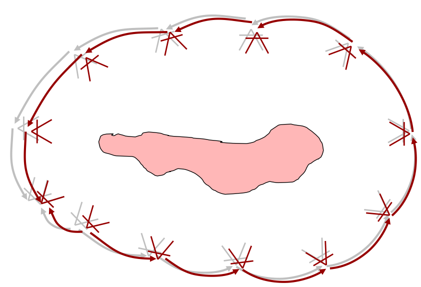
- **Idea:** Estimate camera motion from frame to frame
- Two ways to compute $\mathbf{c}_n$: $\mathbf{c}_n = \mathbf{c}_{n-1} \oplus \mathbf{z}_{(n-1)n}$

$$\mathbf{c}'_n = \mathbf{c}_1 \oplus \mathbf{z}_{1n}$$



Loop Closures

- **Solution:** Use loop-closures to minimize the drift / minimize the error over all constraints

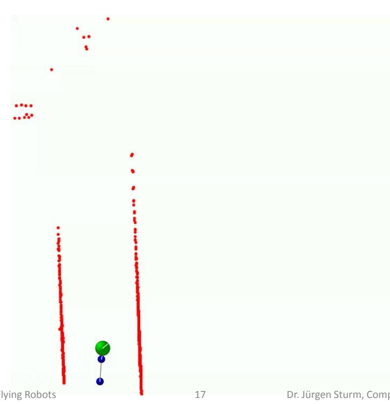


Graph SLAM

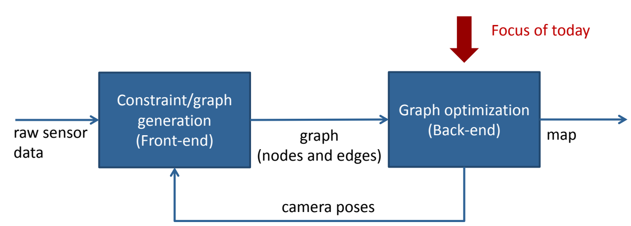
[Thrun and Montemerlo, 2006; Olson et al., 2006]

- 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

Example: Graph SLAM on Intel Dataset



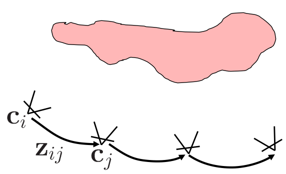
Graph SLAM Architecture



- 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 $\mathbf{z}_{ij} \in \mathbb{R}^6$
- **Wanted:** Set of camera poses $\mathbf{c}_1, \ldots, \mathbf{c}_n \in \mathbb{R}^6$
  ➔ State vector $\mathbf{x} = (\mathbf{c}_1^\top, \ldots, \mathbf{c}_n^\top)^\top \in \mathbb{R}^{6n}$



Map Error

- Real observation $\mathbf{z}_{ij}$
- Expected observation $\bar{\mathbf{z}}_{ij} = \mathbf{c}_j \ominus \mathbf{c}_i$
- Difference between observation and expectation

$$\mathbf{e}_{ij} = \mathbf{z}_{ij} \ominus \bar{\mathbf{z}}_{ij}$$

- Given the correct map, this difference is the result of sensor noise…

Error Function

- **Assumption:** Sensor noise is normally distributed

$$\mathbf{e}_{ij} \sim \mathcal{N}(\mathbf{0}, \Sigma_{ij})$$

- Error term for one observation (proportional to negative loglikelihood)

$$f_{ij}(\mathbf{x}) = \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- Note: error is a scalar $f_{ij}(\mathbf{x}) \in \mathbb{R}$

Error Function

- Map error (over all observations)

$$f(\mathbf{x}) = \sum_{ij} f_{ij}(\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- **Minimize this error** by optimizing the camera poses

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- How can we solve this optimization problem?

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

$$f(\mathbf{x}) = \sum_{ij} f_{ij}(\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- Evaluate the error function around the initial guess

$$f(\mathbf{x}+\Delta\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x}+\Delta\mathbf{x})^\top \Sigma_{ij}^{-1} \underbrace{\mathbf{e}_{ij}(\mathbf{x}+\Delta\mathbf{x})}_{\text{Let's derive this term first...}}$$

Linearize the Error Function

- Approximate the error function around an initial guess $\mathbf{x}$ using Taylor expansion

$$\mathbf{e}_{ij}(\mathbf{x}+\Delta\mathbf{x}) \simeq \mathbf{e}_{ij}(\mathbf{x}) + J_{ij}\Delta\mathbf{x}$$

with

$$J_{ij}(\mathbf{x}) = \begin{pmatrix} \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_1} & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_2} & \cdots & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_n} \end{pmatrix}$$

Derivatives of the Error Terms

- Does one error function $\mathbf{e}_{ij}(\mathbf{x})$ depend on all state variables in $\mathbf{x}$?

Derivatives of the Error Terms

- Does one error function $\mathbf{e}_{ij}(\mathbf{x})$ depend on all state variables in $\mathbf{x}$?
  - No, $\mathbf{e}_{ij}(\mathbf{x})$ depends only on $\mathbf{c}_i$ and $\mathbf{c}_j$

Derivatives of the Error Terms

- Does one error function $\mathbf{e}_{ij}(\mathbf{x})$ depend on all state variables in $\mathbf{x}$?
  - No, $\mathbf{e}_{ij}(\mathbf{x})$ depends only on $\mathbf{c}_i$ and $\mathbf{c}_j$
- Is there any consequence on the **structure** of the Jacobian?

Derivatives of the Error Terms

- Does one error function $\mathbf{e}_{ij}(\mathbf{x})$ depend on all state variables in $\mathbf{x}$?
  - No, $\mathbf{e}_{ij}(\mathbf{x})$ depends only on $\mathbf{c}_i$ and $\mathbf{c}_j$
- Is there any consequence on the **structure** of the Jacobian?
  - Yes, it will be non-zero only in the columns corresponding to $\mathbf{c}_i$ and $\mathbf{c}_j$
  - Jacobian is **sparse**

$$J_{ij}(\mathbf{x}) = \begin{pmatrix} \mathbf{0} & \cdots & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_i} & \cdots & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_j} & \cdots & \mathbf{0} \end{pmatrix}$$

Linearizing the Error Function

Linearize $f(\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^T \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$

$$\simeq \mathbf{c} + 2\mathbf{b}^\top \Delta\mathbf{x} + \Delta\mathbf{x}^\top H \Delta\mathbf{x}$$

with $\mathbf{b}^\top = \sum_{ij} \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$

$$H = \sum_{ij} J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

- What is the structure of $\mathbf{b}^\top$ and H? (Remember: all J_{ij}'s are sparse)

Illustration of the Structure

$\mathbf{b}_{ij}^\top = \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$

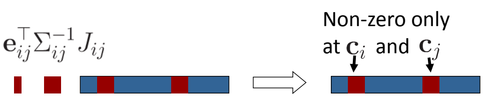


Illustration of the Structure

$\mathbf{b}_{ij}^\top = \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$

$H_{ij} = J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$

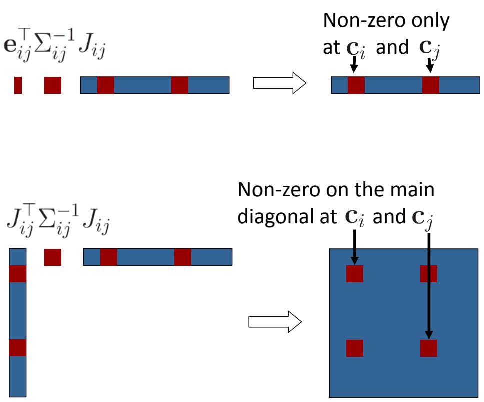


Illustration of the Structure

$\mathbf{b}_{ij}^\top = \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$

$H_{ij} = J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$

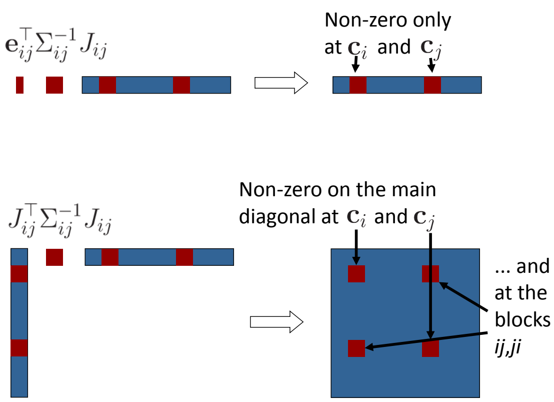
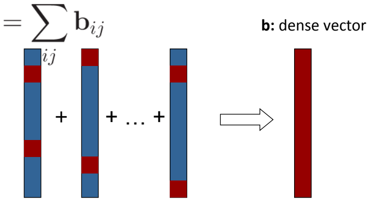
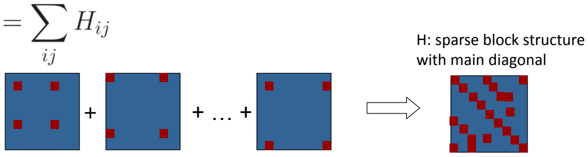


Illustration of the Structure

$\mathbf{b} = \sum_{ij} \mathbf{b}_{ij}$



$H = \sum_{ij} H_{ij}$



(Linear) Least Squares Minimization

1. Linearize error function
$$f(\mathbf{x} + \Delta\mathbf{x}) \simeq \mathbf{c} + 2\mathbf{b}^\top \Delta\mathbf{x} + \Delta\mathbf{x}^\top H \Delta\mathbf{x}$$
2. Compute the derivative
$$\frac{\mathrm{d}f(\mathbf{x} + \Delta\mathbf{x})}{\mathrm{d}\Delta\mathbf{x}} = 2\mathbf{b} + 2H\Delta\mathbf{x}$$
3. Set derivative to zero
$$H\Delta\mathbf{x} = -\mathbf{b}$$
4. Solve this linear system of equations, e.g.,
$$\Delta\mathbf{x} = -H^{-1}\mathbf{b}$$

Gauss-Newton Method

Problem: $f(\mathbf{x})$ is non-linear!

Algorithm: Repeat until convergence

1. Compute the terms of the linear system

$$\mathbf{b}^\top = \sum_{ij} \mathbf{e}_{ij}^T \Sigma_{ij}^{-1} J_{ij} \qquad H = \sum_{ij} J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

2. Solve the linear system to get new increment

$$H \Delta \mathbf{x} = -\mathbf{b}$$

3. Update previous estimate

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta \mathbf{x}$$

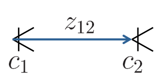
Sparsity of the Hessian

- 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 $\mathbf{x} = (c_1, c_2)^\top \in \mathbb{R}^2$
- One (distance) observation $z_{12} \in \mathbb{R}$

- Initial guess $c_1 = c_2 = 0$
- Observation $z_{12} = 1$
- Sensor noise $\Sigma_{12} = 0.5$



Example in 1D

- Error $\quad e_{12} = z_{12} - \bar{z}_{12}$

$$= z_{12} - (c_2 - c_1) = 1 - (0 - 0) = 1$$

- Jacobian $J_{12} = \begin{pmatrix} \frac{\partial e_{12}}{\partial c_1} & \frac{\partial e_{12}}{\partial c_2} \end{pmatrix} = \begin{pmatrix} 1 & -1 \end{pmatrix}$
- Build linear system of equations

$$b^\top = e_{12}^\top \Sigma^{-1} e_{12} = \begin{pmatrix} 2 & -2 \end{pmatrix}$$

$$H = J_{12}^\top \Sigma^{-1} J_{12} = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix}$$

- Solve the system

$$\Delta x = -H^{-1} b \qquad \textbf{but } \det H = 0 \textbf{ ???}$$

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 $H, \mathbf{b}$ 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)**

$$H = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} + \boxed{\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}}$$

additional constraint that sets $\Delta c_1 = 0$

$$\Delta x = -H^{-1} b$$

$$\Delta x = \begin{pmatrix} 0 & 1 \end{pmatrix}^\top$$

Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor

$$(H + \lambda I)\Delta\mathbf{x} = -\mathbf{b}$$
$$(J^\top J + \lambda I)\Delta\mathbf{x} = -J^\top\mathbf{e}$$

- What is the effect of this damping factor?
  - Small λ?
  - Large λ?

Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor

$$(H + \lambda I)\Delta\mathbf{x} = -\mathbf{b}$$
$$(J^\top J + \lambda I)\Delta\mathbf{x} = -J^\top\mathbf{e}$$

- What is the effect of this damping factor?
  - Small λ → same as least squares
  - Large λ → steepest descent (with small step size)
- **Algorithm**
  - If error decreases, accept $\Delta\mathbf{x}$ and reduce λ
  - If error increases, reject $\Delta\mathbf{x}$ and increase λ

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

Bundle Adjustment

- **Graph SLAM:** Optimize (only) the camera poses

$$\mathbf{x} = (\mathbf{c}_1^\top, \ldots, \mathbf{c}_n^\top)^\top \in \mathbb{R}^{6n}$$

- **Bundle Adjustment:** Optimize both 6DOF camera poses and 3D (feature) points

$$\mathbf{x} = (\underbrace{\mathbf{c}_1^\top, \ldots, \mathbf{c}_n^\top}_{\mathbf{c}\in\mathbb{R}^{6n}}, \underbrace{\mathbf{p}_1^\top, \ldots, \mathbf{p}_m^\top}_{\mathbf{p}\in\mathbb{R}^{3m}})^\top \in \mathbb{R}^{6n+3m}$$

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

Error Function

- Camera pose $\mathbf{c}_i \in \mathbb{R}^6$
- Feature point $\mathbf{p}_j \in \mathbb{R}^3$
- Observed feature location $\mathbf{z}_{ij} \in \mathbb{R}^2$
- Expected feature location

$$g(\mathbf{c}_i, \mathbf{p}_j) = R_i^\top(\mathbf{t}_i - \mathbf{p}_j)$$
$$h(\mathbf{c}_i, \mathbf{p}_j) = g_{x,y}(\mathbf{c}_i, \mathbf{p}_j)/g_z(\mathbf{c}_i, \mathbf{p}_j)$$

Error Function

- Difference between observation and expectation

$$\mathbf{e}_{ij} = \mathbf{z}_{ij} - h(\mathbf{c}_i, \mathbf{p}_j)$$

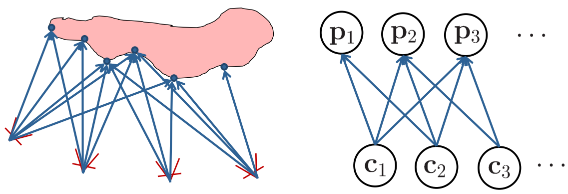
- Error function

$$f(\mathbf{c}, \mathbf{p}) = \sum_{ij} \mathbf{e}_{ij}^\top \Sigma^{-1} \mathbf{e}_{ij}$$

- Covariance Σ 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

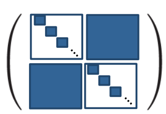
- Characteristic structure

$$\begin{pmatrix} J_\mathbf{c}^\top J_\mathbf{c} & J_\mathbf{c}^\top J_\mathbf{p} \\ J_\mathbf{p}^\top J_\mathbf{c} & J_\mathbf{p}^\top J_\mathbf{p} \end{pmatrix} \begin{pmatrix} \Delta\mathbf{c} \\ \Delta\mathbf{p} \end{pmatrix} = \begin{pmatrix} -J_\mathbf{c}^\top \mathbf{e}_\mathbf{c} \\ -J_\mathbf{p}^\top \mathbf{e}_\mathbf{p} \end{pmatrix}$$

$$\begin{pmatrix} H_\mathbf{cc} & H_\mathbf{cp} \\ H_\mathbf{pc} & H_\mathbf{pp} \end{pmatrix} \begin{pmatrix} \Delta\mathbf{c} \\ \Delta\mathbf{p} \end{pmatrix} = \begin{pmatrix} -J_\mathbf{c}^\top \mathbf{e}_\mathbf{c} \\ -J_\mathbf{p}^\top \mathbf{e}_\mathbf{p} \end{pmatrix}$$

Primary Structure

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



$$\begin{pmatrix} \Delta\mathbf{c} \\ \Delta\mathbf{p} \end{pmatrix} = \begin{pmatrix} -J_\mathbf{c}^\top \mathbf{e}_\mathbf{c} \\ -J_\mathbf{p}^\top \mathbf{e}_\mathbf{p} \end{pmatrix}$$

- This can be efficiently solved using the Schur Complement

Schur Complement

- Given: Linear system

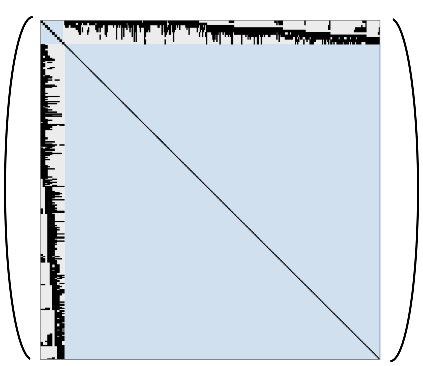
$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix}$$

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

$$(A - BD^{-1}C)\mathbf{x} = \mathbf{a} - BD^{-1}\mathbf{b}$$
$$\mathbf{y} = D^{-1}(\mathbf{b} - C\mathbf{x})$$

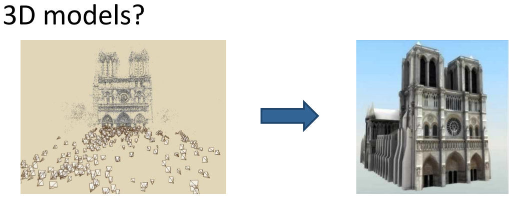
- **Reduced complexity**, i.e., invert one $p \times p$ and $p \times p$ matrix instead of one $(p+q) \times (p+q)$ matrix

Example Hessian (Lourakis and Argyros, 2009)

$H =$ 

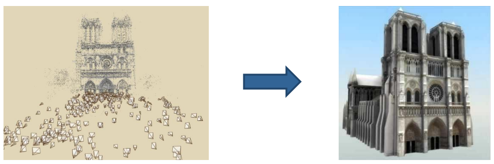
From Sparse Maps to Dense Maps

- 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?

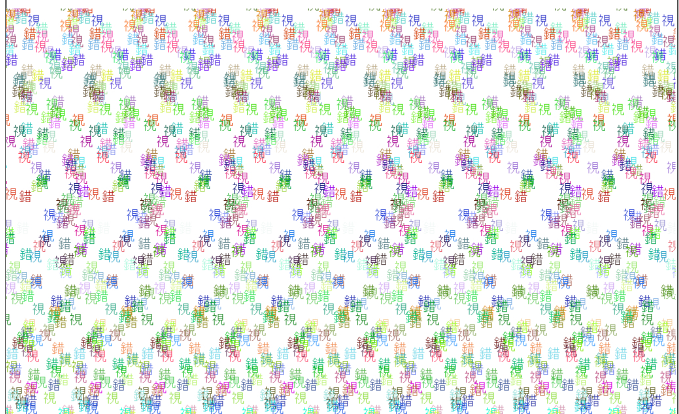


From Sparse Maps to Dense Maps

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

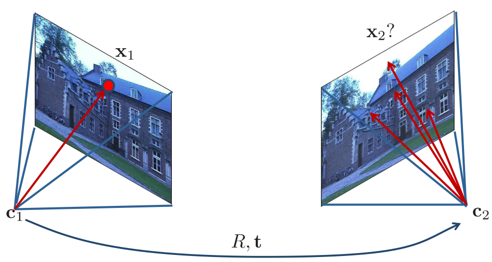


Human Stereo Vision



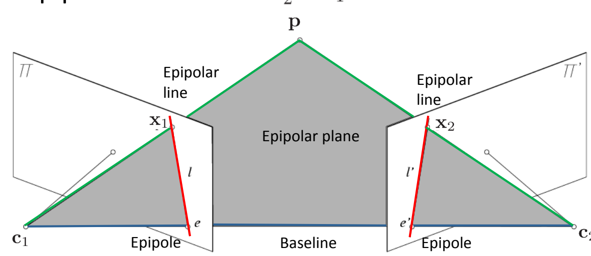
Stereo Correspondence Constraints

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



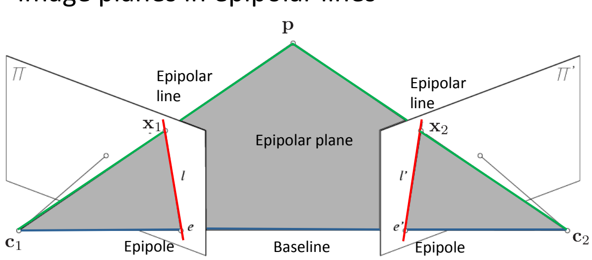
Reminder: Epipolar Geometry

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



Epipolar Plane

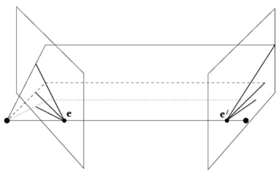
- All epipolar lines intersect at the epipoles
- An epipolar plane intersects the left and right image planes in epipolar lines

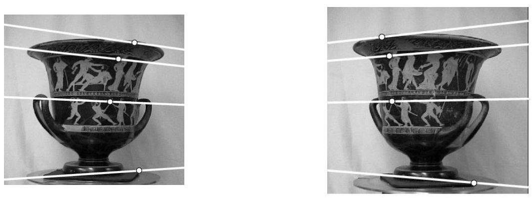


Epipolar Constraint

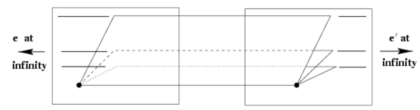
- This is useful because it reduces the correspondence problem to a 1D search along an epipolar line

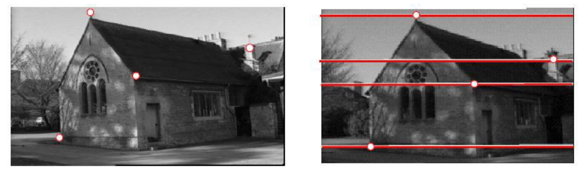
Example: Converging Cameras





Example: Parallel Cameras



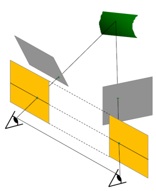


Rectification

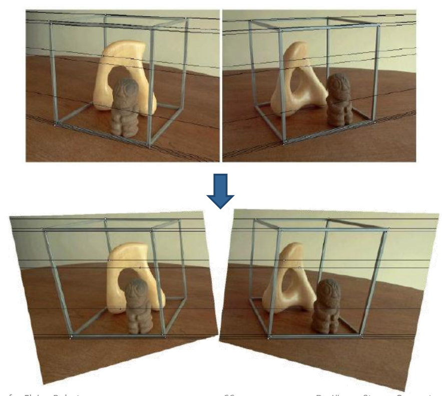
- In practice, it is convenient if the image scanlines (rows) are the epipolar lines

→ Reproject image planes onto a common plane parallel to the baseline (two 3x3 homographies)

- Afterwards pixel motion is horizontal

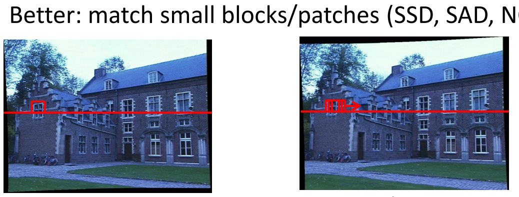


Example: Rectification



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)



left image

right image

Block Matching Algorithm

Input: Two images and camera calibrations

Output: Disparity (or depth) image

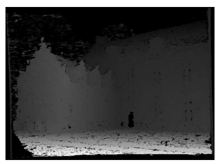
Algorithm:

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

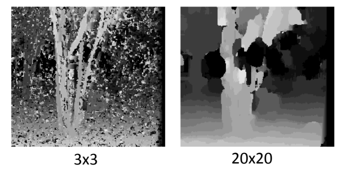
- Input

- Output



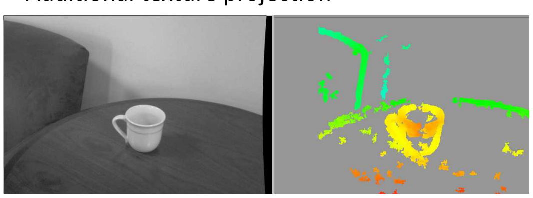
What is the Influence of the Block Size?

- 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)

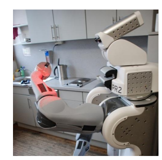


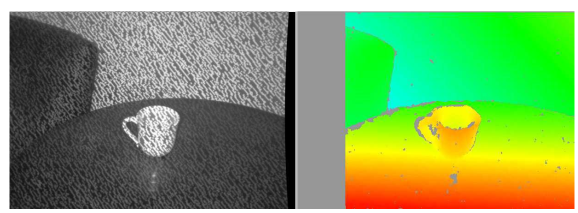
Problems with Stereo

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



Example: PR2 Robot with Projected Texture Stereo





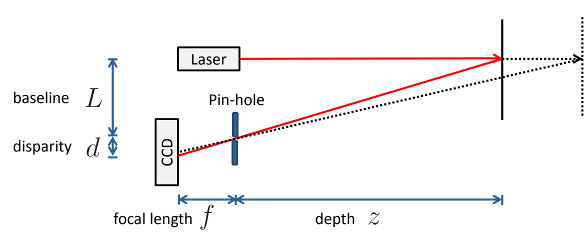
Laser Triangulation

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

Laser Triangulation

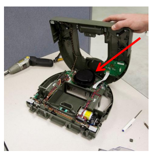
- Function principle

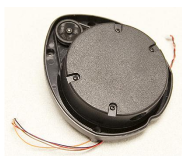


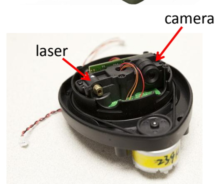
- Depth triangulation $z = f\frac{L}{d}$
  (note: same for stereo disparities)

Example: Neato XV-11

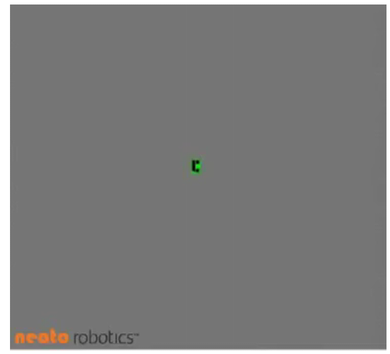
- K. Konolige, “A low-cost laser distance sensor”, ICRA 2008
- Specs: 360deg, 10Hz, 30 USD





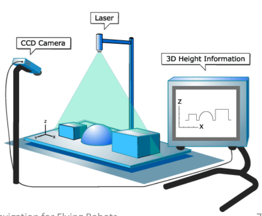


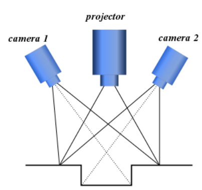
How Does the Data Look Like?



Laser Triangulation

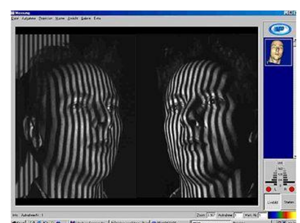
- Stripe laser + 2D camera
- Often used on conveyer belts (volume sensing)
- Large baseline gives better depth resolution but more occlusions → use two cameras





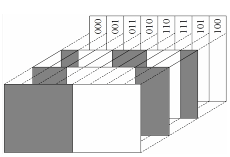
Structured Light

- Multiple stripes / 2D pattern
- Data association more difficult



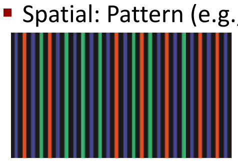
Structured Light

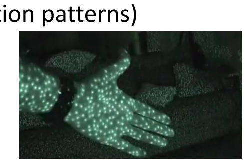
- Multiple stripes / 2D pattern
- Data association more difficult
- Coding schemes
  - Temporal: Coded light



Structured Light

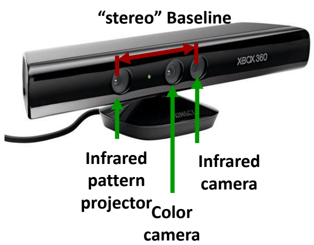
- Multiple stripes / 2D pattern
- Data association more difficult
- Coding schemes
  - Temporal: Coded light
  - Wavelength: Color
  - Spatial: Pattern (e.g., diffraction patterns)





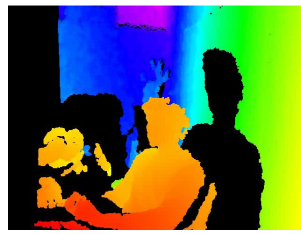
Sensor Principle of Kinect

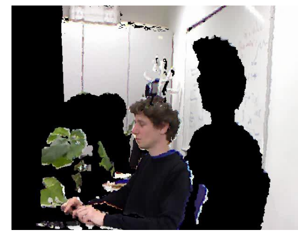
- Kinect projects a diffraction pattern (speckles) in near-infrared light
- CMOS IR camera observes the scene



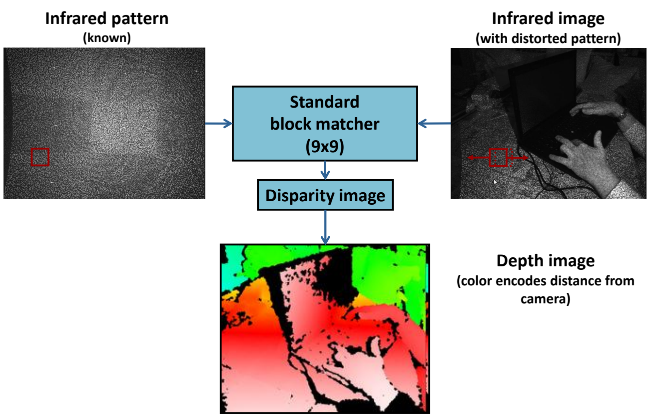
Example Data

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





Sensor Principle of Kinect



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

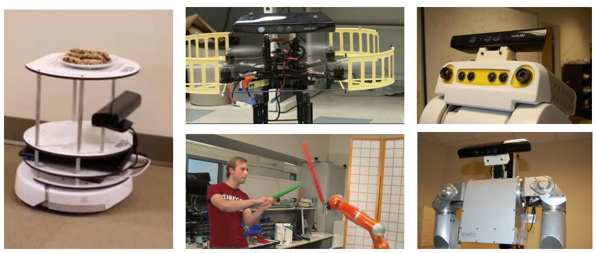
- Infrared camera has 640x480 @ 30 Hz
  - Depth correlation runs on FPGA
  - 11-bit depth image
  - 0.8m – 5m range
  - Depth sensing does not work in direct sunlight (why?)
- RGB camera has 640x480 @ 30 Hz
  - Bayer color filter
- Four 16-bit microphones with DSP for beam forming @ 16kHz
- Requires 12V (for motor), weighs 500 grams
- Human pose recognition runs on Xbox CPU and uses only 10-15% processing power @30 Hz
(Paper: http://research.microsoft.com/apps/pubs/default.aspx?id=145347)

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?

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

Place Recognition, ICP, and Dense Reconstruction

Dr. Jürgen Sturm

Exercise Sheet 5

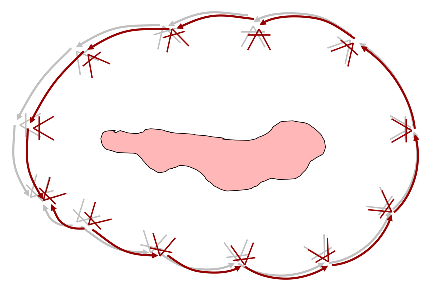
- Prepare mid-term presentation
- Proposed structure: 3 slides
  1. Remind people who you are and what you are doing (can be same slide as last time)
  2. Your work/achievements so far (video is a plus)
  3. Your plans for the next two weeks
- Hand in slides before July 3, 10am

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

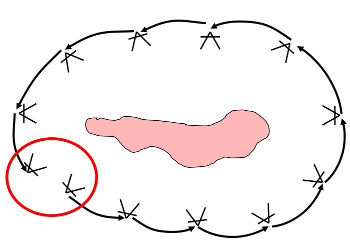
Remember: Loop Closures

- Use loop-closures to minimize the drift / minimize the error over all constraints



Loop Closures

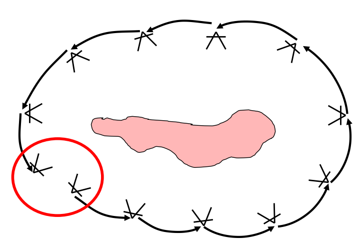
How can we detect loop closures efficiently?



Loop Closures

How can we detect loop closures efficiently?

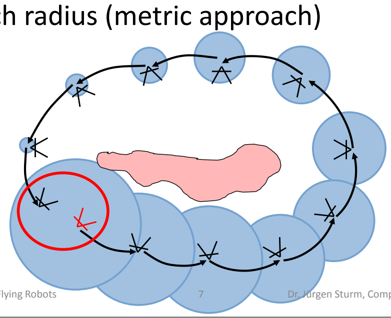
1. Compare with all previous images $O(n)$ (not efficient)



Loop Closures

How can we detect loop closures efficiently?

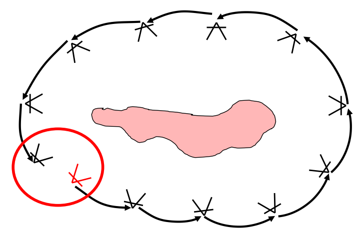
2. Use motion model and covariance to limit search radius (metric approach)



Loop Closures

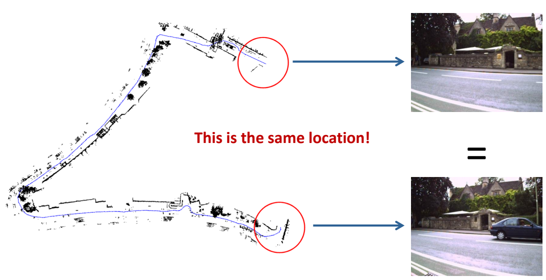
How can we detect loop closures efficiently?

3. Appearance-based place recognition (using bag of words)

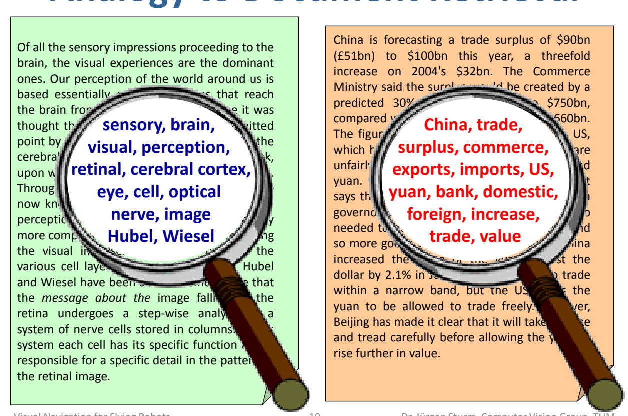


Appearance-based Place Recognition

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



Analogy to Document Retrieval



Object/Scene Recognition

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

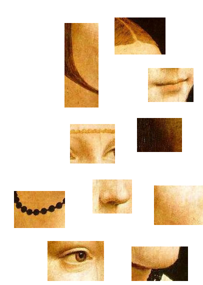

object


bag of visual words

Bag of Visual Words

- Visual words = (independent) features


face


features

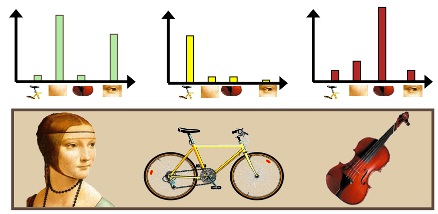
Bag of Visual Words

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

dictionary of visual words (codebook)

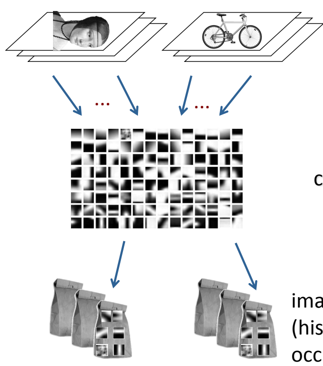
Bag of Visual Words

- Visual words = (independent) features
- Construct a dictionary of representative words
- Represent the image based on a histogram of word occurrences (bag)

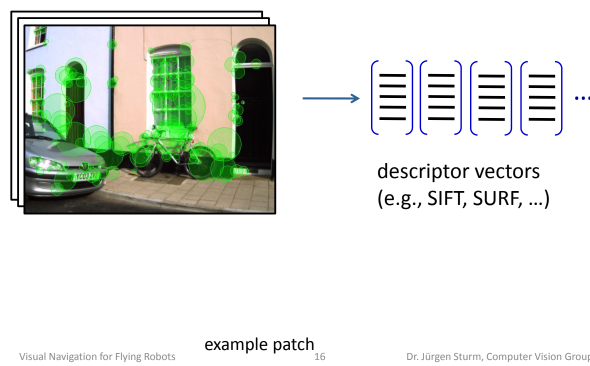


Each detected feature is assigned to the closest entry in the codebook

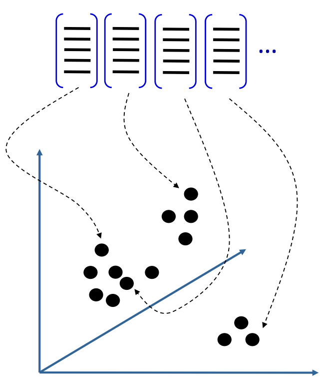
Overview



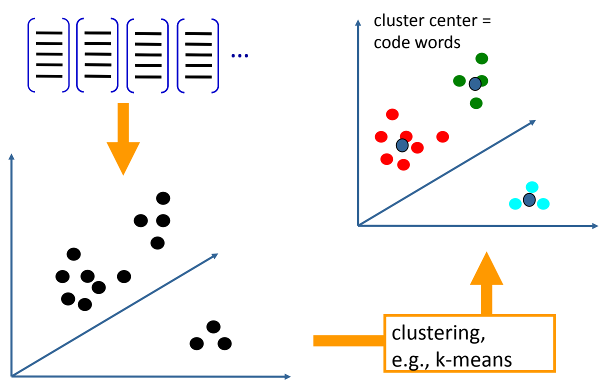
Learning the Dictionary



Learning the Dictionary



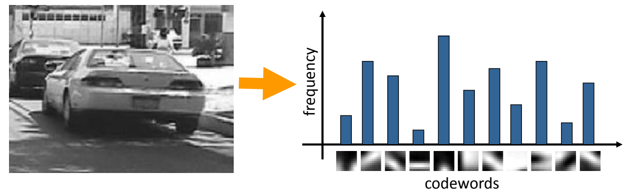
Learning the Dictionary



Learning the Visual Vocabulary

Example Image Representation

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

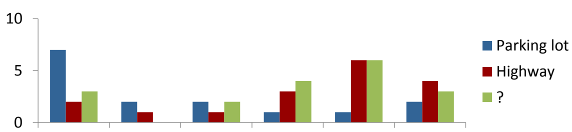


Object/Scene Recognition

- Compare histogram of new scene with those of known scenes, e.g., using
  - simple histogram intersection

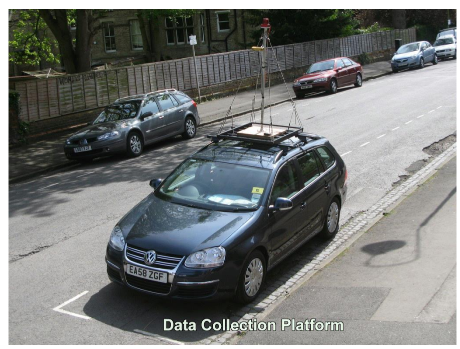
$$score(\mathbf{p}, \mathbf{q}) = \sum \min(p_i, q_i)$$

  - naïve Bayes
  - more advanced statistical methods



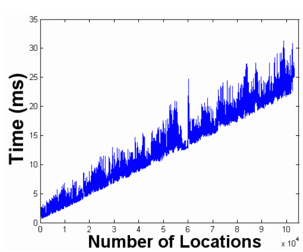
Example: FAB-MAP

[Cummins and Newman, 2008]



Timing Performance

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



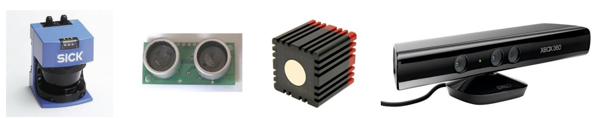
Summary: Bag of Words

[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)

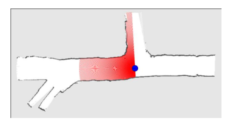
Laser-based Motion Estimation

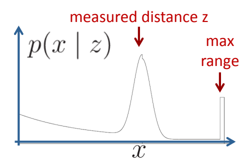
- So far, we looked at motion estimation (and place recognition) from **visual** sensors
- Today, we cover motion estimation from **range** sensors
  - Laser scanner (laser range finder, ultrasound)
  - Depth cameras (time-of-flight, Kinect …)



Laser Scanner

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





Laser-based Motion Estimation

How can we best align two laser scans?

Laser-based Motion Estimation

How can we best align two laser scans?

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

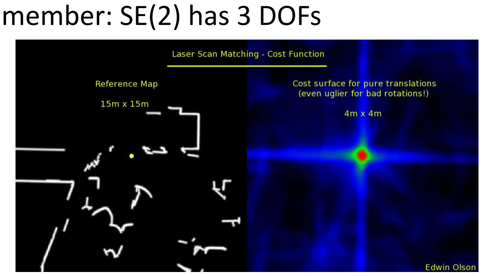
Exhaustive Search

- Convolve first scan with sensor model

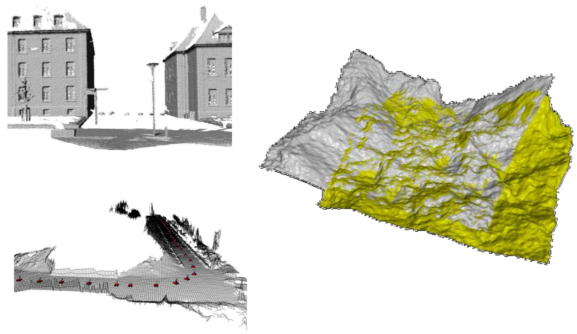
- Sweep second scan over likelihood map, compute correlation and select best pose

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)

$$P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$$
$$Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$$

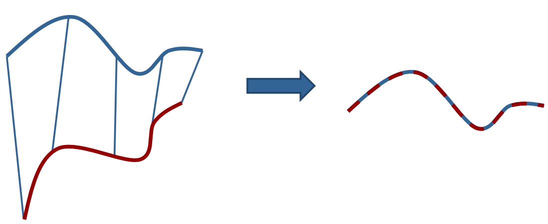
- **Wanted:** Translation $\mathbf{t}$ and rotation R that minimize the sum of the squared error

$$E(R, \mathbf{t}) = \frac{1}{n} \sum_{i=1}^{n} ||\mathbf{p}_i - R\mathbf{q}_i - \mathbf{t}||^2$$

where $\mathbf{p}_i$ and $\mathbf{q}_i$ 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

$$\bar{\mathbf{p}} = \frac{1}{n} \sum_i \mathbf{p}_i \qquad \bar{\mathbf{q}} = \frac{1}{n} \sum_i \mathbf{q}_i$$

- 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…

Known Correspondences

- Decompose the matrix

$$W = \sum_i (\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{q}_i - \bar{\mathbf{q}})^\top = USV^\top$$

  using singular value decomposition (SVD)

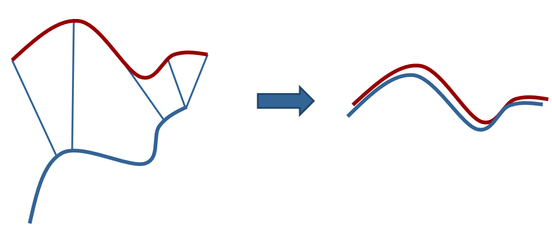
- **Theorem**
  If $\operatorname{rank} W = 3$, the optimal solution of $E(R, \mathbf{t})$ is unique and given by

$$R = UV^\top$$
$$\mathbf{t} = \bar{\mathbf{p}} - R\bar{\mathbf{q}}$$

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

Unknown Correspondences

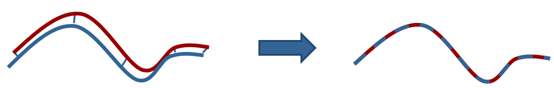
- If the correct correspondences are not known, it is generally impossible to determine the optimal transformation in one step



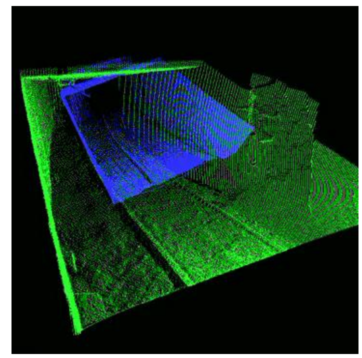
ICP Algorithm

[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

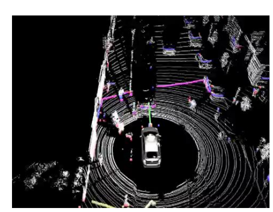
- Various aspects of performance
  - Speed
  - Stability (local minima)
  - Tolerance w.r.t. noise and/or outliers
  - Basin of convergence (maximum initial misalignment)
- Choice depends on data and application

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

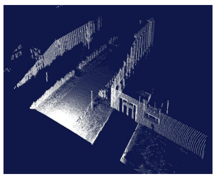
- Density of points usually depends on the distance to the sensor → no uniform distribution
- Can lead to a bias in ICP



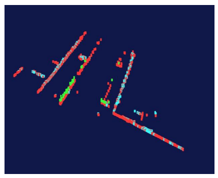
Feature-based Sampling

Detect interest points (same as with images)

- Decrease the number of correspondences
- Increase efficiency and accuracy
- Requires pre-processing

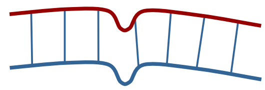


3D Scan (~200.000 Points)

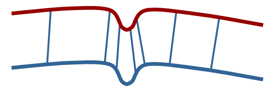


Extracted Features (~5.000 Points)

Normal-Space Sampling



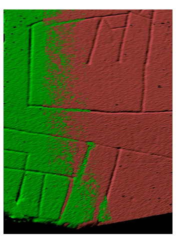
Uniform sampling



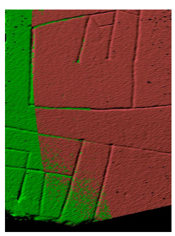
Normal-space sampling

Example: Normal-Space Sampling

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



Random sampling



Normal-space sampling

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

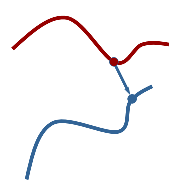
Finding Correspondences

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

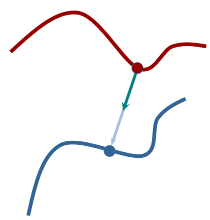
- Find closest point in the other point set
- Distance threshold



- Closest-point matching generally stable, but slow and requires pre-processing

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

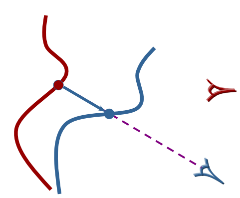
Speeding Up Correspondence Search

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)

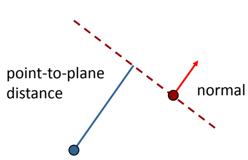
Projection-based Matching

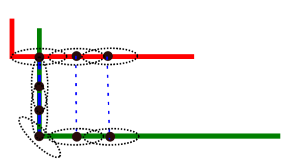
- Slightly worse performance per iteration
- Each iteration is one to two orders of magnitude faster than closest-point
- Requires point-to-plane error metric



Error Metrics

- Point-to-point
- Point-to-plane lets flat regions slide along each other



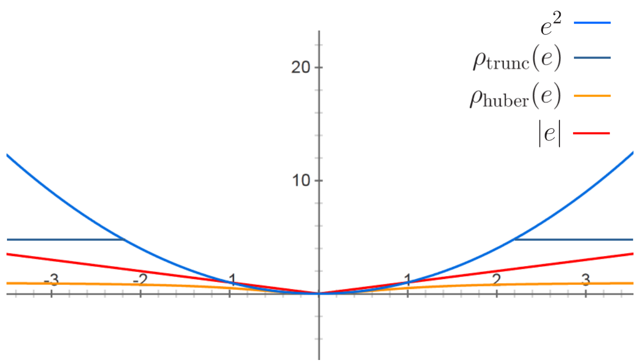


- Generalized ICP: Assign individual covariance to each data point [Segal, 2009]

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?

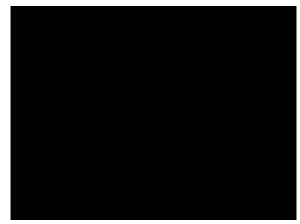
Robust Error Metrics



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

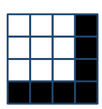
- ICP is a powerful algorithm for calculating the displacement between point clouds
- The overall speed depends most on the choice of matching algorithm
- ICP is (in general) only locally optimal → can get stuck in local minima

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 $\mathbf{m}$ using a grid
- Each cell is either free or occupied

$$\mathbf{m} = (m_1, \dots, m_n) \in \{\text{empty}, \text{occ}\}^n$$

- Robot maintains a belief $\text{Bel}(\mathbf{m})$ on map state

Goal: Estimate the belief from sensor observations

$$\text{Bel}(\mathbf{m}) = P(\mathbf{m} \mid \mathbf{z}_1, \dots, \mathbf{z}_t)$$

Occupancy Grid - Assumptions

- Map is static
- Cells have binary state (empty or occupied)
- All cells are independent of each other

- As a result, each cell m_i can be estimated independently from the sensor observations
- Will also drop index i (for the moment)

Mapping

- **Goal:** Estimate
$$\mathrm{Bel}(m) = P(m \mid z_1, \dots, z_n)$$
- How can this be computed?

Binary Bayes Filter

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

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

Binary Bayes Filter

- **Prior probability** that cell is occupied $P(m)$ (often 0.5)
- **Inverse sensor model** $P(m \mid z_t)$ is specific to the sensor used for mapping
- The **log-odds representation** can be used to increase speed and numerical stability

$$L(x) := \log \frac{p(x)}{p(\neg x)} = \log \frac{p(x)}{1 - p(x)}$$

Binary Bayes Filter using Log-Odds

- In each time step, compute

$$L(m \mid z_{1:t}) = \underbrace{L(m \mid z_{1:t-1})}_{\text{previous belief}} + \underbrace{L(m \mid z_t)}_{\text{inverse sensor model}} + \underbrace{L(m)}_{\text{map prior}}$$

- When needed, compute current belief as

$$\mathrm{Bel}_t(m) = 1 - \frac{1}{1 + \exp L(m \mid z_{1:t})}$$

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

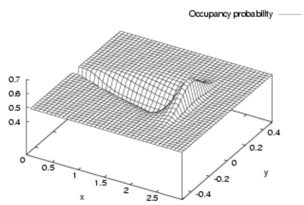
$$L'(m \mid z_{1:t}) = \max(\min(L(m \mid z_{1:t}), l_{\max}), l_{\min})$$

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?

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

Resulting Map

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?

Memory Consumption

- Consider we want to map a building with 40x40m at a resolution of 0.05cm
- How much memory do we need?

$$\left(\frac{40}{0.05}\right)^2 = 640.000 \text{ cells} = 4.88\text{mb}$$

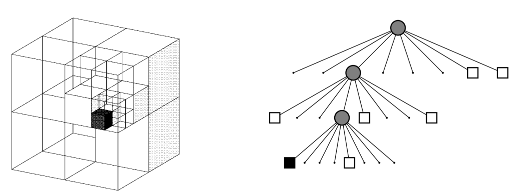
- And for 3D?

$$\left(\frac{40}{0.05}\right)^3 = 512.000.000 \text{ cells} = 3.8\text{gb}$$

- And what about a whole city?

Map Representation by Octtrees

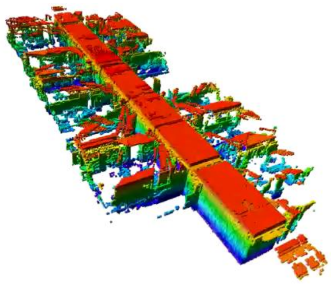
- Tree-based data structure
- Recursive subdivision of space into octants
- Volumes can be allocated as needed
- Multi-resolution



Example: OctoMap

[Wurm et al., 2011]

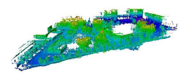
- Freiburg, building 79
  44 x 18 x 3 m^3, 0.05m resolution, 0.7mb on disk



Example: OctoMap

[Wurm et al., 2011]

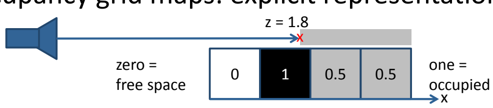
- Freiburg computer science campus
  292 x 167 x 28 m^3, 0.2m resolution, 2mb on disk



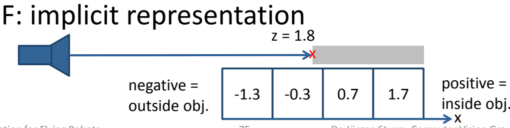
Signed Distance Field (SDF)

[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

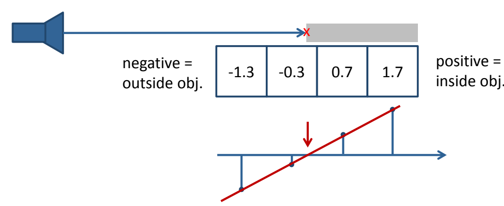


Signed Distance Field (SDF)

[Curless and Levoy, 1996]

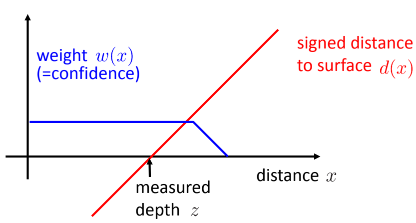
Algorithm:

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, …)

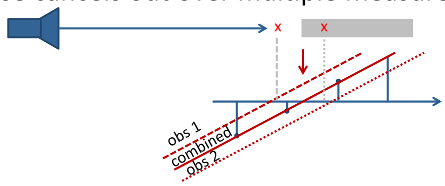
Data Fusion

- Each voxel cell x in the SDF stores two values
  - Weighted sum of signed distances $D_t(\mathbf{x})$
  - Sum of all weights $W_t(\mathbf{x})$
- When new range image arrives, update every voxel cell according to

$$D_{t+1}(\mathbf{x}) = D_t(\mathbf{x}) + w_{t+1}(\mathbf{x})d_{t+1}(\mathbf{x})$$
$$W_{t+1}(\mathbf{x}) = W_t(\mathbf{x}) + w_{t+1}(\mathbf{x})$$

Two Nice Properties

- Noise cancels out over multiple measurements

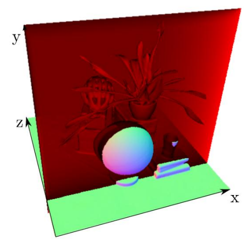


- Zero-crossing can be extracted at sub-voxel accuracy (least squares estimate)

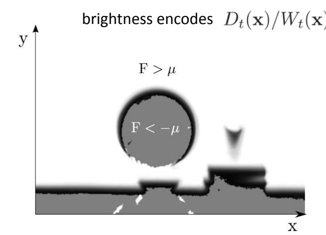
1D Example: $x^* = \dfrac{\sum D_t(x)x}{\sum W_t(x)x}$

SDF Example

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

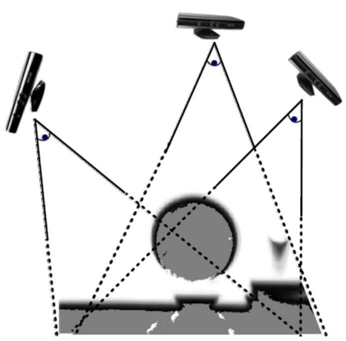


Surface with cross-section



SDF

SDF Fusion



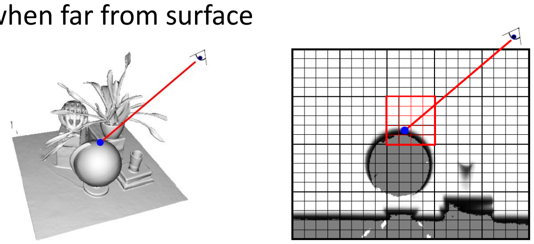
Visualizing Signed Distance Fields

Common approaches to iso surface extraction:

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

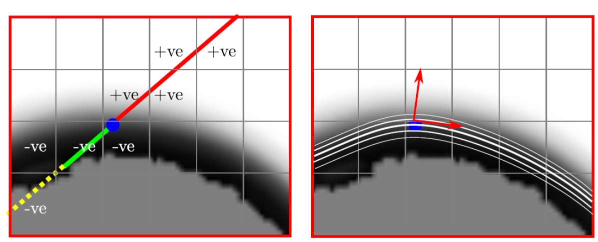
Ray Casting

- For each camera pixel, shoot a ray and search for the first zero crossing in the SDF
- Value in the SDF can be used to skip along when far from surface



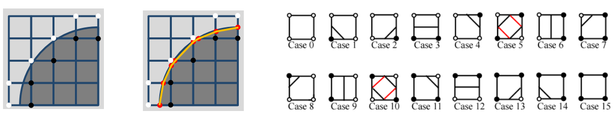
Ray Casting

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

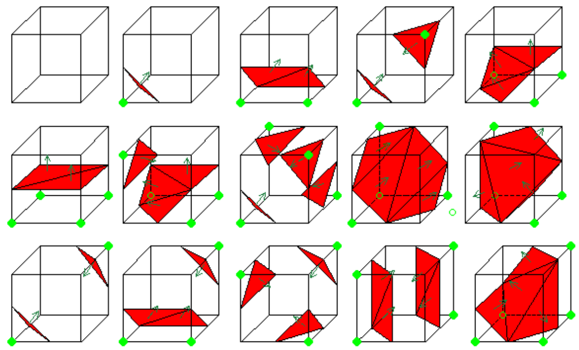


Marching Cubes

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



KinectFusion
[Newcombe et al., 2011]

- Projective ICP with point-to-plane metric
- Truncated signed distance function (TSDF)
- Ray Casting

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

Computer Vision Group
Prof. Daniel Cremers

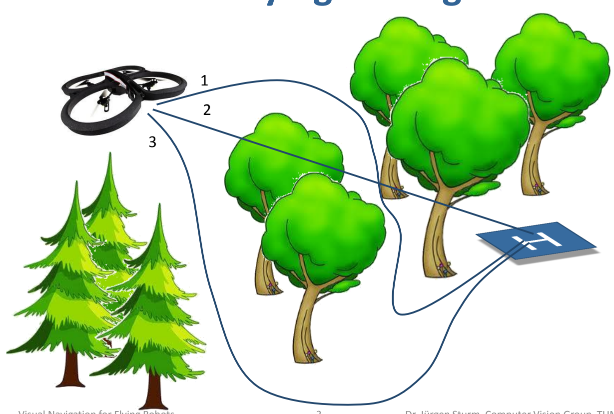
Technische Universität München

Visual Navigation for Flying Robots

Motion Planning

Dr. Jürgen Sturm

Motivation: Flying Through Forests



Motion Planning Problem

- Given obstacles, a robot, and its motion capabilities, compute collision-free robot motions from the start to goal.

Motion Planning Problem

What are good performance metrics?

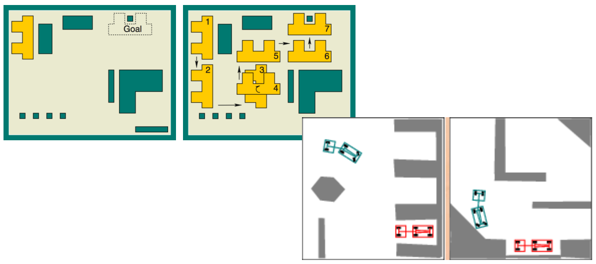
Motion Planning Problem

What are good performance metrics?

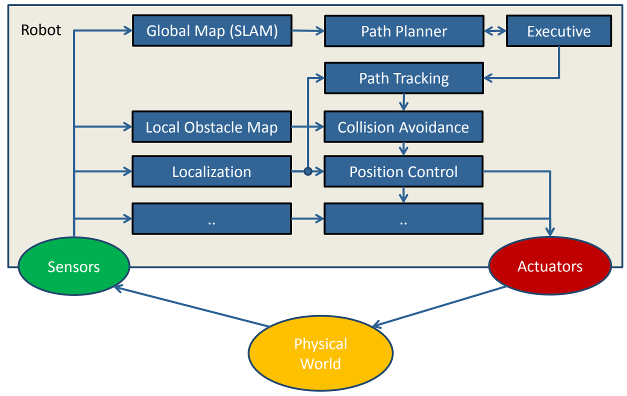
- Execution speed / path length
- Energy consumption
- Planning speed
- Safety (minimum distance to obstacles)
- Robustness against disturbances
- Probability of success
- …

Motion Planning Examples

Motion planning is sometimes also called the **piano mover's problem**



Robot Architecture



Agenda for Today

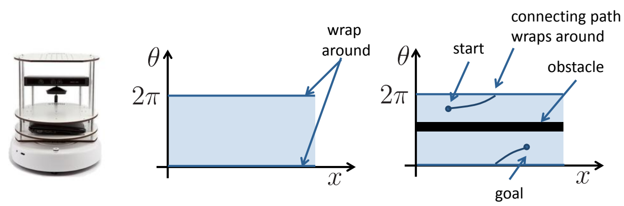
- Configuration spaces
- Roadmap construction
- Search algorithms
- Path optimization and re-planning
- Path execution

Configuration Space

- Work space
  - Typically 3D pose (position + orientation) → 6 DOF
- Configuration space
  - Reduced pose (position + yaw) → 4 DOF
  - Full pose → 6 DOF
  - Pose + velocity → 12 DOF
  - Joint angles of manipulation robot
  - …
- Planning takes place in **configuration space**

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

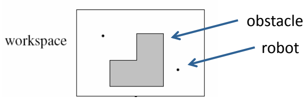
- Configuration space $C \subset \mathbb{R}^d$
- Configuration $\mathbf{q} \in C$
- Free space C_{free}
- Obstacle space C_{obs}

- Properties

$$C_{\text{free}} \cup C_{\text{obs}} = C$$
$$C_{\text{free}} \cap C_{\text{obs}} = \emptyset$$

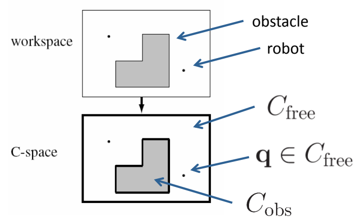
Free Space Example

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



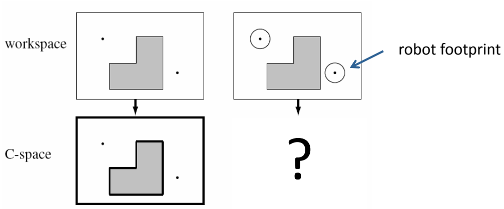
Example

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



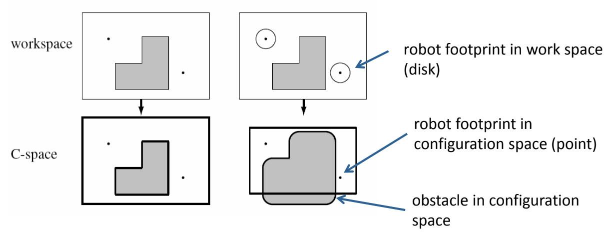
Example

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



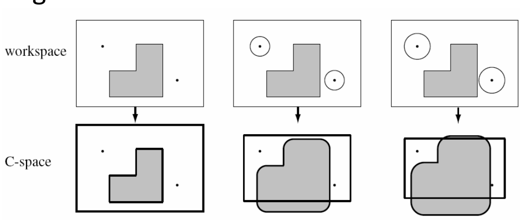
Example

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



Example

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



Computing the Free Space

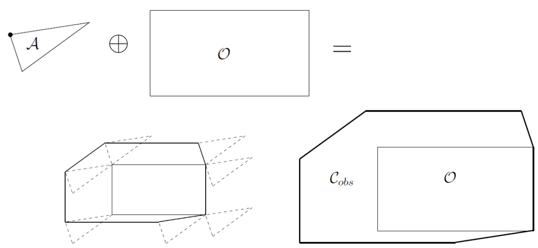
- Free configuration space is obtained by sliding the robot along the edge of the obstacle regions "blowing them up" by the robot radius
- This operation is called the **Minowski sum**

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

where $A, B \subset \mathbb{R}^d$

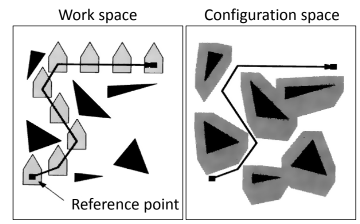
Example: Minowski Sum

- Triangular robot and rectangular obstacle



Example

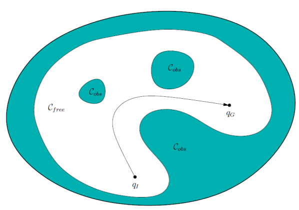
- Polygonal robot, translation only



- C-space is obtained by sliding the robot along the edge of the obstacle regions

Basic Motion Planning Problem

- **Given**
  - Free space C_{free}
  - Initial configuration $\mathbf{q}_I$
  - Goal configuration $\mathbf{q}_G$



- **Goal:** Find a continuous path

$$\tau : [0,1] \to C_{\text{free}}$$

with $\tau(0) = \mathbf{q}_I, \ \tau(1) = \mathbf{q}_G$

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)

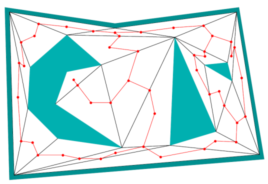
Combinatorial Methods

- 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

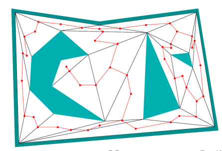
A **roadmap** is a graph in C_{free} where

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



(Desired) Properties of Roadmaps

- **Accessibility**
  From anywhere in C_{free}, it is easy to compute a path that reaches at least one of the vertices
- **Connectivity-preserving**
  If there exists a path between $\mathbf{q}_I$ and $\mathbf{q}_G$ in C_{free} then there must also exist a path in the road map



Roadmap Construction

We consider here three **combinatorial** methods:

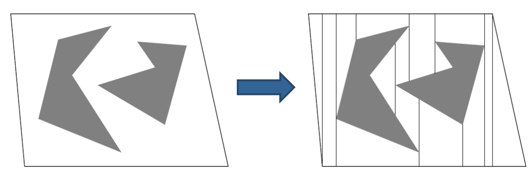
- Trapezoidal decomposition
- Shortest path roadmap
- Regular grid
- … but there are many more!

Afterwards, we consider two **sampling-based** methods:

- Probabilistic roadmaps (PRMs)
- Rapidly exploring random trees (RRTs)

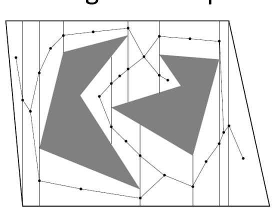
Roadmap Construction

- Decompose horizontally in convex regions using plane sweep
- Sort vertices in x direction. Iterate over vertices while maintaining a vertically sorted list of edges



Roadmap Construction

- Place vertices
  - in the center of each trapezoid
  - on the edge between two neighboring trapezoids
- Resulting road map

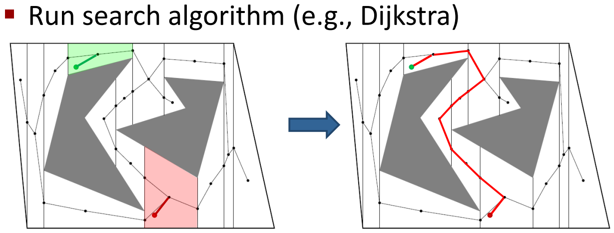


Quick check on properties:
- Accessibility
- Connectivity-preserving?

Example Query

Compute path from $\mathbf{q}_I$ to $\mathbf{q}_G$

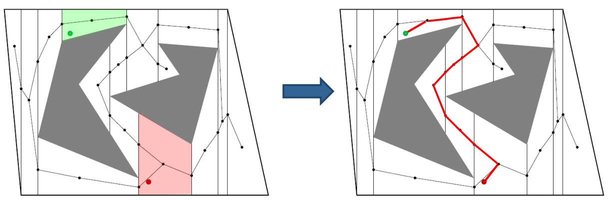
- Identify **start** and **goal** trapezoid
- Connect **start** and **goal** location to center vertex
- Run search algorithm (e.g., Dijkstra)



Properties of Trapezoidal Decomposition

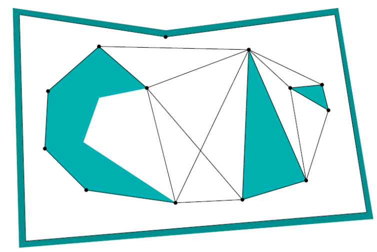
+ Easy to implement
+ Efficient computation
+ Scales to 3D

- Does not generate shortest path



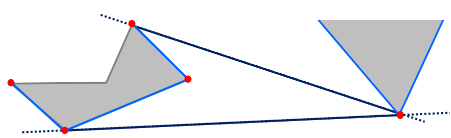
Shortest-Path Roadmap

- Contains all vertices and edges that optimal paths follow when obstructed
- Imagine pulling a tight string between q_I and q_G



Roadmap Construction

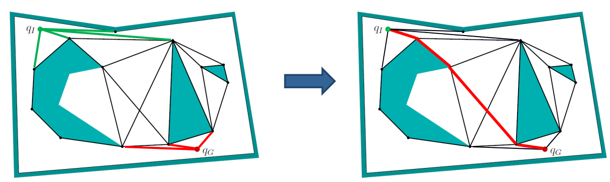
- Vertices = all sharp corners (>180deg, red)
- Edges
  1. Two consecutive sharp corners on the same obstacle (light blue)
  2. Bitangent edges (when line connecting two vertices extends into free space, dark blue)



Example Query

Compute path from q_I to q_G

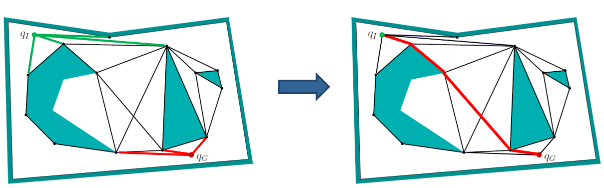
- Connect **start** and **goal** location to all visible roadmap vertices
- Run search algorithm (e.g., Dijkstra)



Example Query

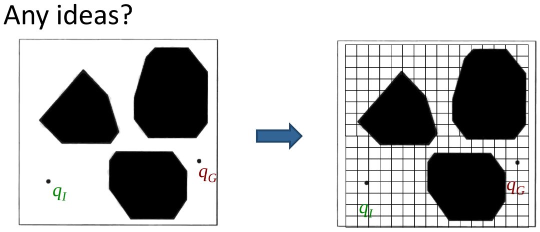
+ Easy to construct in 2D
+ Generates shortest paths

- Optimal planning in 3D or more dim. is NP-hard



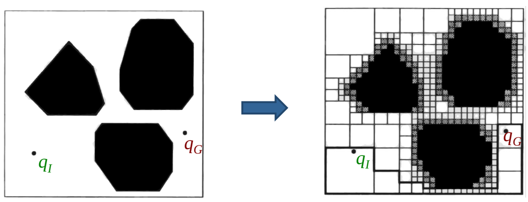
Approximate Decompositions

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



Approximate Decompositions

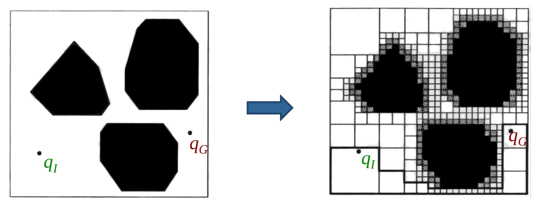
- Construct a regular grid
- Use quadtree/octtree to save memory
- Sometimes difficult to determine status of cell



Approximate Decompositions

+ Easy to construct
+ Most used in practice

- High number of tests



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_{free} and C_{obs} and leave the algorithm **in the dark** when exploring C_{free}
- The only light is provided by a **collision-detection algorithm** that probes C to see whether some configuration lies in C_{free}
- We will have a look at
  - Probabilistic road maps (PRMs)
  - Rapidly exploring random trees (RRTs)

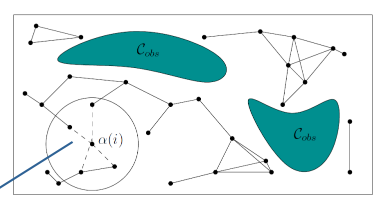
Probabilistic Roadmaps (PRMs)

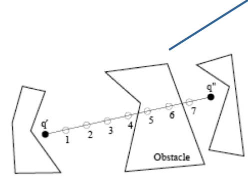
[Kavraki et al., 1992]

- **Vertex:** Take random sample from C, check whether sample is in C_{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



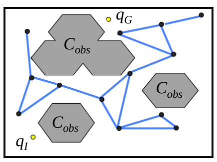


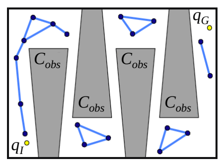
Step 3: Check edges for collisions, e.g., using discretized line search

Probabilistic Roadmaps

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

- Do not work well for some problems (e.g., narrow passages)
- Not optimal, not complete

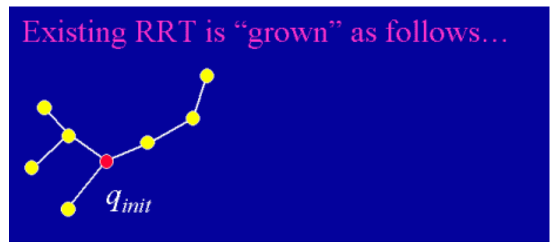




Rapidly Exploring Random Trees

[Lavalle and Kuffner, 1999]

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



Rapidly Exploring Random Trees

- **Algorithm**
  1. Initialize tree with first node $\mathbf{q}_I$
  2. Pick a random target location (every 100th iteration, choose $\mathbf{q}_G$)
  3. Find closest vertex in roadmap
  4. Extend this vertex towards target location
  5. Repeat steps until goal is reached

- Why not pick $\mathbf{q}_G$ every time?

Rapidly Exploring Random Trees

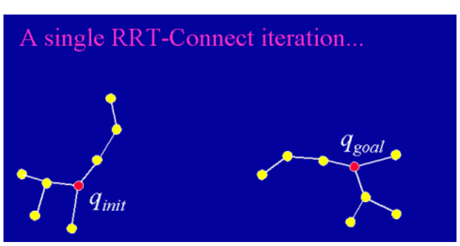
- **Algorithm**
  1. Initialize tree with first node $\mathbf{q}_I$
  2. Pick a random target location (every 100th iteration, choose $\mathbf{q}_G$)
  3. Find closest vertex in roadmap
  4. Extend this vertex towards target location
  5. Repeat steps until goal is reached

- Why not pick $\mathbf{q}_G$ every time?
- This will fail and run into C_{obs} instead of exploring

Rapidly Exploring Random Trees

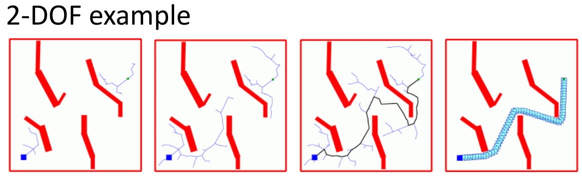
[Lavalle and Kuffner, 1999]

- **RRT:** Grow trees from start and goal location towards each other, stop when they connect

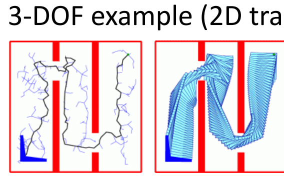


RRT Examples

- 2-DOF example

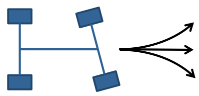


- 3-DOF example (2D translation + rotation)



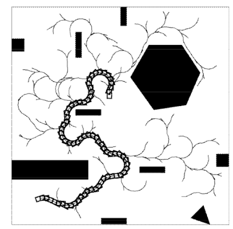
Non-Holonomic Robots

- 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)



Non-Holonomic Robots

- RRTs can naturally consider such constraints during tree construction
- Example: Car-like robot



Rapidly Exploring Random Trees

\+ Probabilistic. complete

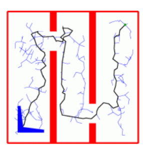
\+ Balance between greedy search and exploration

\+ 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, …)

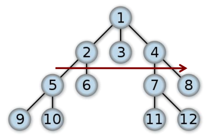
Search Algorithms

- **Given:** Graph G consisting of vertices and edges (with associated costs)
- **Wanted:** find the best (shortest) path between two vertices

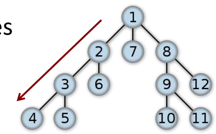
- What search algorithms do you know?

Uninformed Search

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

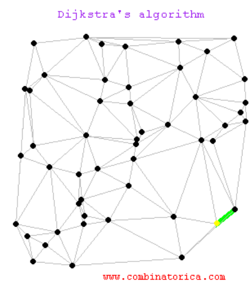


- **Depth-first**
  - Not complete in infinite spaces
  - Not optimal
  - Time $O(b^d)$
  - Space $O(bd)$
    (can forget explored subtrees)



Example: Dijkstra's Algorithm

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



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

- **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?
- Often a combination of
  - Path cost so far $g(n)$
  - Heuristic function $h(n)$
    (e.g., estimated distance to goal, but can also encode additional domain knowledge)

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)

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

Problems on A* on Grids

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…

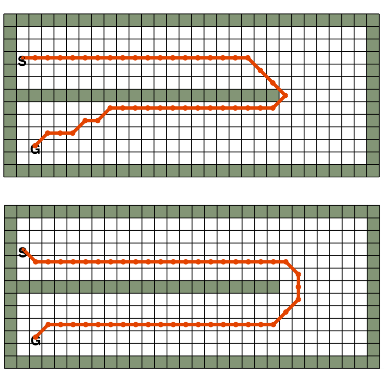
Map Smoothing

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

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

$$f(n) = g(n) \cdot p_{\mathrm{occ}}(n) + h(n)$$

Example: Map Smoothing

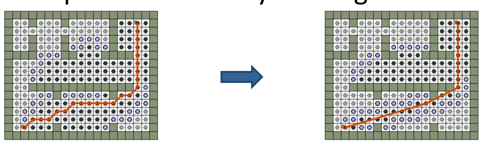


Path Smoothing

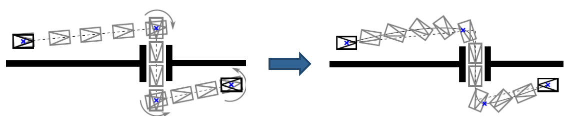
- **Problem:** Paths are aligned to grid structure (because they have to lie in the roadmap)
- Paths look unnatural and are sub-optimal
- **Solution:** Smooth the path after generation
  - Traverse path and find pairs of nodes with direct line of sight; replace by line segment
  - Refine initial path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

Example: 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**?

D* Search

- **Idea:** Incrementally repair path keeping its modifications local around robot pose
- Many variants:
  - D* (Dynamic A*) [Stentz, ICRA '94] [Stentz, IJCAI '95]
  - D* Lite [Koenig and Likhachev, AAAI '02]
  - Field D* [Ferguson and Stenz, JFR '06]

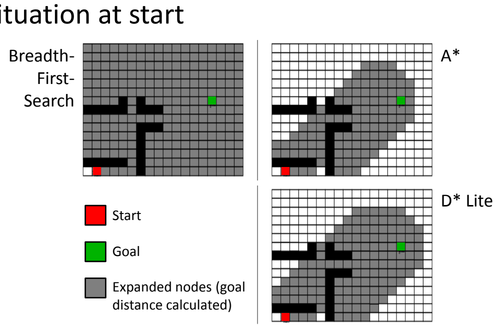
D* Search

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 **re-using previous solution**

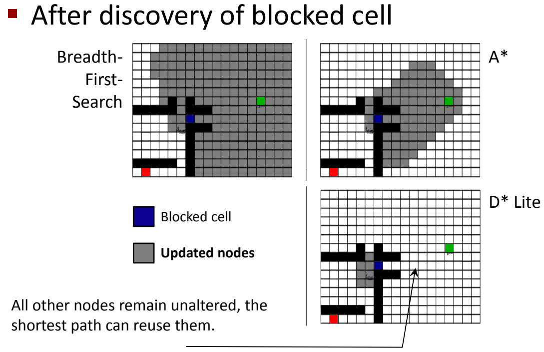
D* Example

- Situation at start



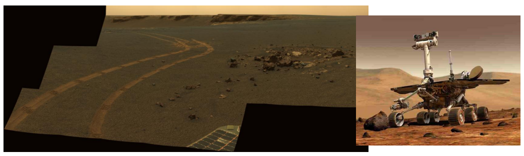
D* Example

- After discovery of blocked cell



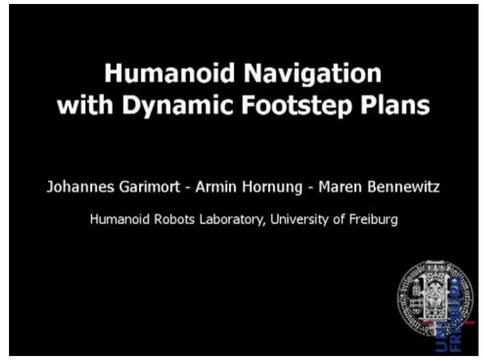
D* Search

- 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



D* Lite for Footstep Planning

[Garimort et al., ICRA '11]



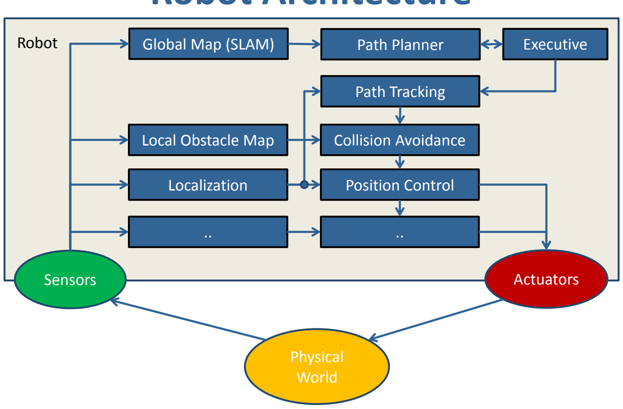
Real-Time Motion Planning

- 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?

Real-Time Motion Planning

- What is the maximum time needed to re-plan in case of an obstacle detection?
  In principle, re-planning with D* can take arbitrarily long
- What if the robot has to react quickly to unforeseen, fast moving objects?
  Need a collision avoidance algorithm that runs in constant time!
- Do we really need to re-plan for every obstacle on the way?
  Could trigger re-planning only if path gets obstructed (or robot predicts that re-planning reduces path length by p%)

Robot Architecture



Layered Motion Planning

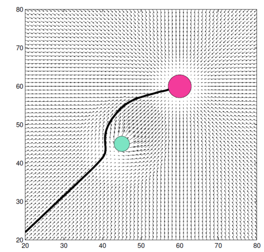
- An approximate **global planner** computes paths ignoring the kinematic and dynamic vehicle constraints (not real-time)
- An accurate **local planner** accounts for the constraints and generates feasible local trajectories in real-time (collision avoidance)

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

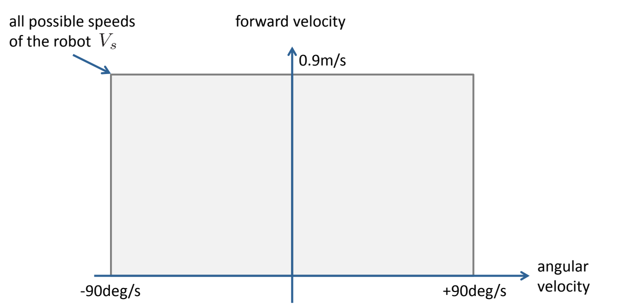
- Treat robot as a particle under the influence of a potential field
- **Pro:**
  - easy to implement
- **Con:**
  - suffers from local minima
  - no consideration of dynamic constraints



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

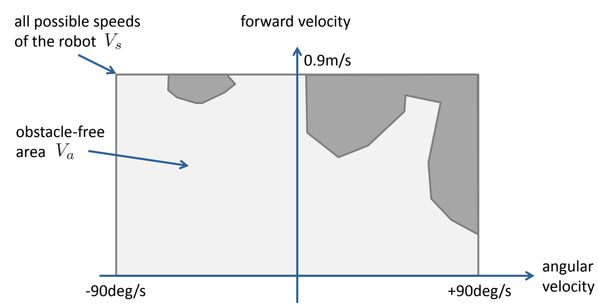
- Consider a 2D planar robot



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

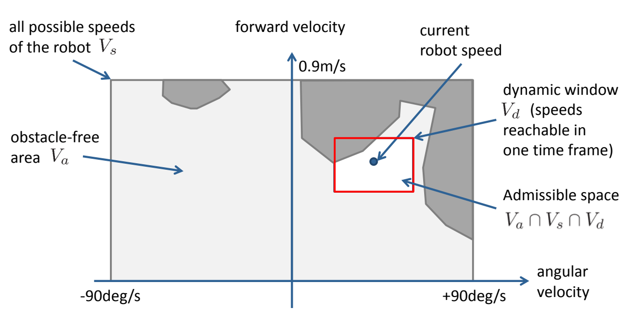
- Consider a 2D planar robot + 2D environment



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

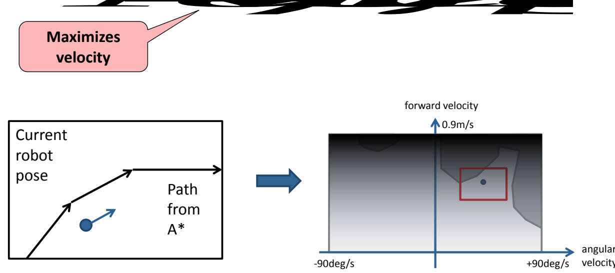
- Consider additionally dynamic constraints



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

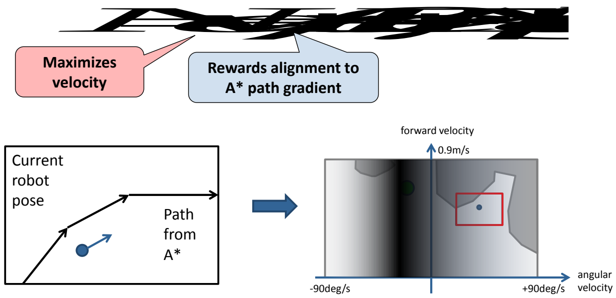
- Navigation function (potential field)



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

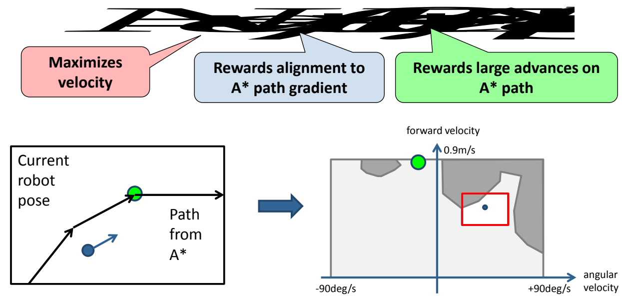
- Navigation function (potential field)



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

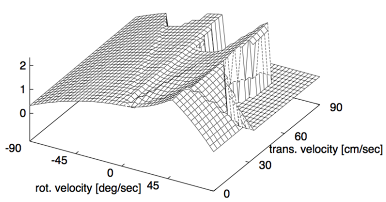
- Navigation function (potential field)



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Discretize dynamic window and evaluate navigation function (note: window has fixed size = real-time!)
- Find the maximum and execute motion

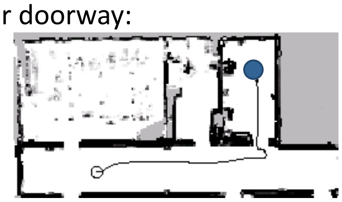


Example: Dynamic Window Approach

[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

Computer Vision Group
Prof. Daniel Cremers
Technische Universität München

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

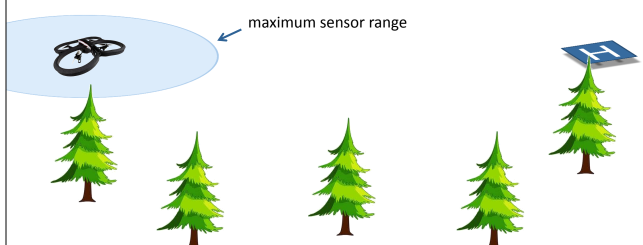
Agenda For Next Week

- **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

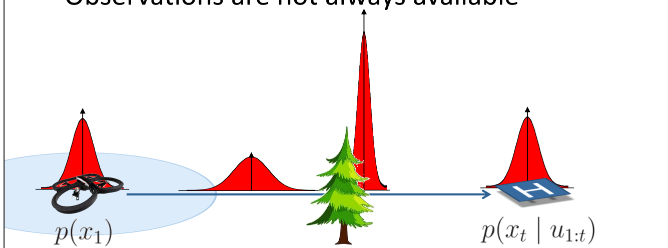
- Execution speed / path length
- Energy consumption
- Planning speed
- Safety (minimum distance to obstacles)
- **Robustness against disturbances**
- **Probability of success**
- …

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



Visual Navigation for Flying Robots 7 Dr. Jürgen Sturm, Computer Vision Group, TUM

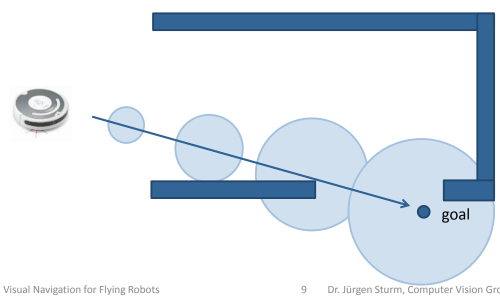
Solution 1: Shape The Environment To Decrease Uncertainty

- Assume a robot without sensors
- What is a good navigation plan?

Visual Navigation for Flying Robots 8 Dr. Jürgen Sturm, Computer Vision Group, TUM

Solution 1: Shape The Environment To Decrease Uncertainty

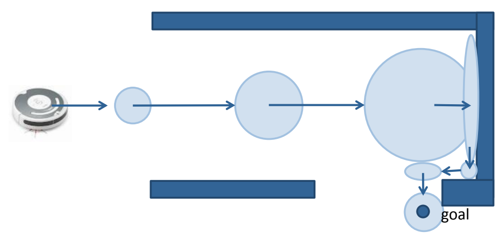
- Plan 1: Take the shortest path
- What is the probability of success of plan 1?



Visual Navigation for Flying Robots 9 Dr. Jürgen Sturm, Computer Vision Group, TUM

Solution 1: Shape The Environment To Decrease Uncertainty

- What is the probability of success of plan 2?



Visual Navigation for Flying Robots 10 Dr. Jürgen Sturm, Computer Vision Group, TUM

Solution 1: Shape The Environment To Decrease Uncertainty

- **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 conveyer belts)
  - …

Visual Navigation for Flying Robots 11 Dr. Jürgen Sturm, Computer Vision Group, TUM

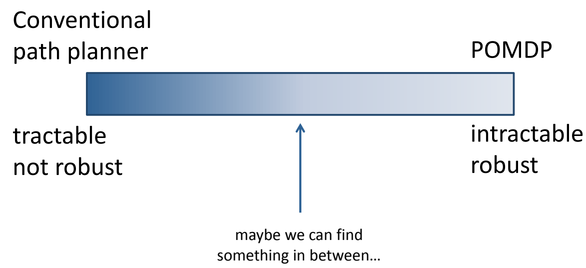
Solution 2: Add (More/Better) Sensors

Visual Navigation for Flying Robots 12 Dr. Jürgen Sturm, Computer Vision Group, TUM

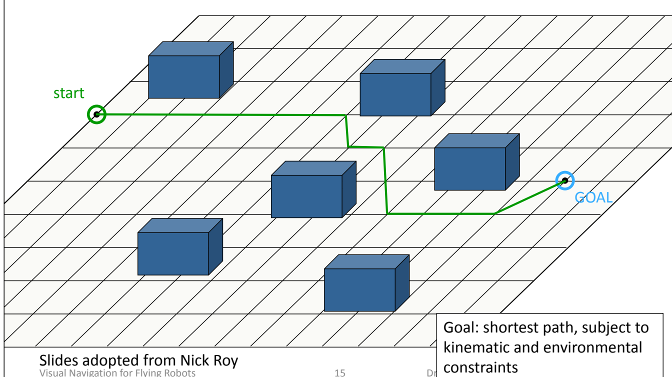
Solution 3: POMDPs

- 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

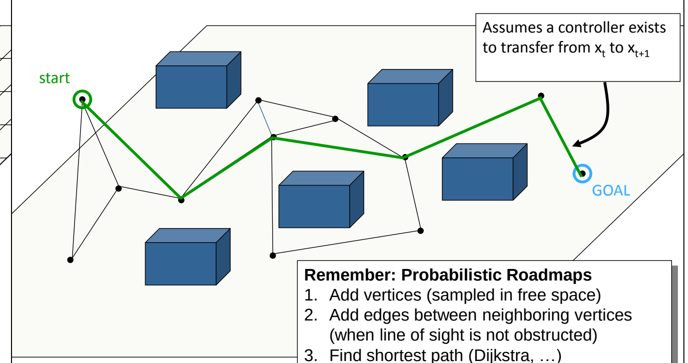


Remember: Motion Planning



Slides adopted from Nick Roy

Remember: Motion Planning in High-Dimensional Configuration Spaces



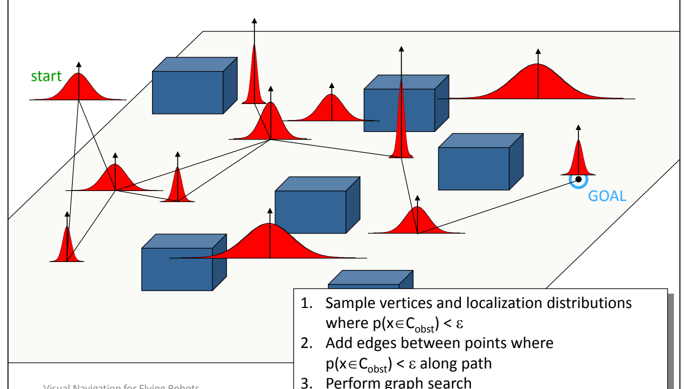
Slides adopted from Nick Roy

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

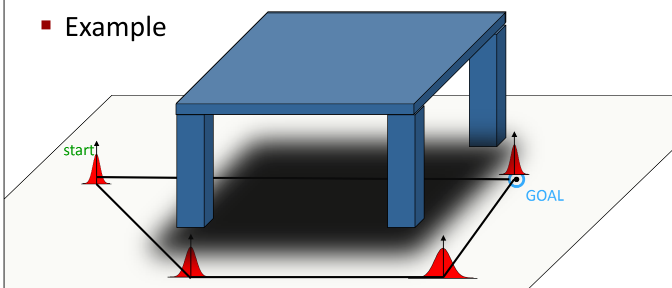
Motion Planning in Information Space

[Roy et al.]



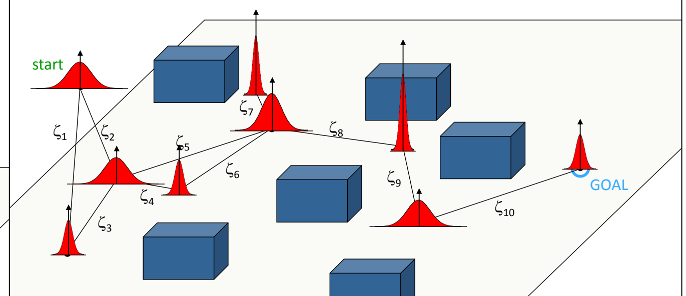
Motion Planning in Information Space

- **Problem:** Posterior distribution depends also on the path taken to the vertex
- Example



Belief Roadmap

[He et al., 2008]



1. Sample vertices from C_{free}, build graph and estimate belief dist. transfer functions
2. Propagate covariances by performing graph search

Slides adopted from Nick Roy

Planning in Information Spaces

[He et al., 2008]

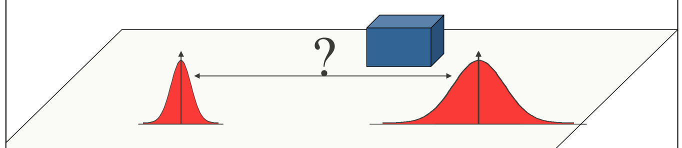
- **Given:** Roadmap
- **Goal:** Find path from start to goal nodes that results in minimum uncertainty at goal
- **Problem:** How can we estimate the belief distribution at the goal (efficiently)?

Planning in Information Spaces

[He et al., 2008]

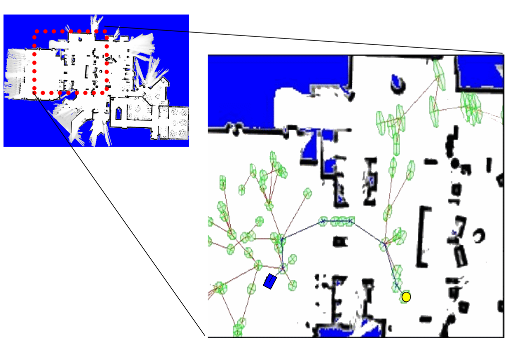
How can we propagate the belief distribution along an edge?

1. Sample waypoints, use forward simulation to compute full posterior
2. Linearize model and use Kalman filter



Example: Belief Roadmap

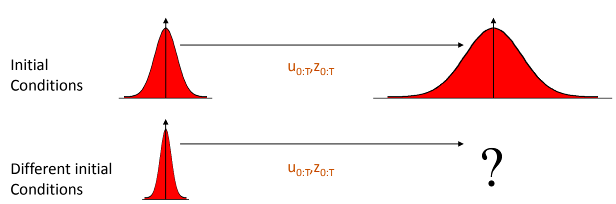
[He et al., 2008]



Belief Propagation

[He et al., 2008]

- The posterior distribution depends on the prior distribution



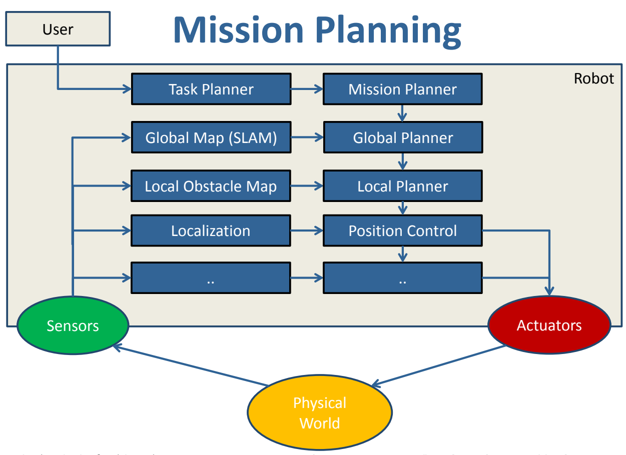
Planning in Information Spaces
[He et al., 2008]

- 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: ≈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



Mission Planning

- **Goal:** Generate and execute a plan to accomplish a certain (navigation) task
- Example tasks
  - Exploration
  - Coverage
  - Surveillance
  - Tracking
  - …

Task Planning

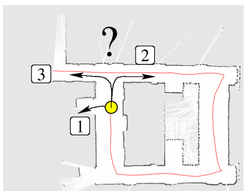
- **Goal:** Generate and execute a high level plan to accomplish a certain task
- Often symbolic reasoning (or hard-coded)
  - Propositional or first-order logic
  - Automated reasoning systems
  - Common programming languages: Prolog, LISP
- Multi-agent systems, communication
- Artificial Intelligence

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**

Exploration

- Choose the action that maximizes utility

$$a^* = \arg\max_{a \in A} U(m, a)$$

- **Question:** How can we define utility?

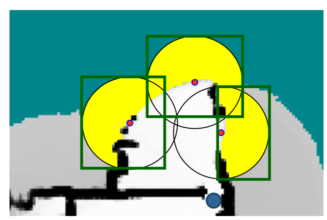
Example

- Where should the robot go next?

Maximizing the Information Gain

- Pick the action a that maximizes the **information gain** given a map m

$$a^* = \arg\max_{a \in A} IG(m, a)$$



Information Theory

- **Entropy** is a general measure for the uncertainty of a probability distribution
- Entropy = Expected amount of information needed to encode an outcome $X = x$

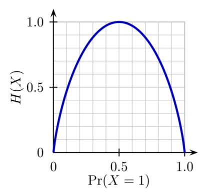
$$\begin{aligned} H(X) &= E(I(X)) \\ &= E(-\log p(X)) \\ &= -\sum_{i=1}^{n} p(x_i) \log p(x_i) \end{aligned}$$

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:



Example: Map Entropy

The overall entropy is the sum of the individual entropy values

Information Theory

- **Information gain** = Uncertainty reduction

$$IG(X, Y) = H(X) - H(X \mid Y)$$

- **Conditional entropy**

$$H(X \mid Y) = \sum_{i,j} p(x_i, y_j) \log \frac{p(y_j)}{p(x_i, y_j)}$$

Maximizing the Information Gain

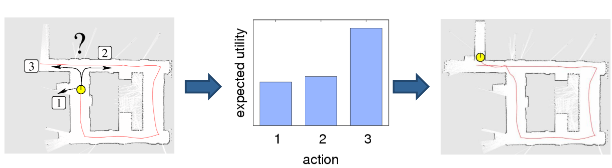
- To compute the information gain one needs to know the observations obtained when carrying out an action

$$a^* = \arg\max_{a \in A} IG(m, a)$$

- This quantity is not known! Reason about potential measurements

$$a^* = \arg\max_{a \in A} \int IG(m, z) p(z \mid a) \mathrm{d}z$$

Example



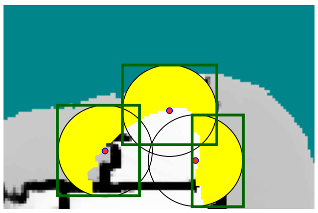
Exploration Costs

- 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

$$a^* = \arg\max_{a \in A} IG(m, a) - \alpha \cdot E(cost(m, a))$$

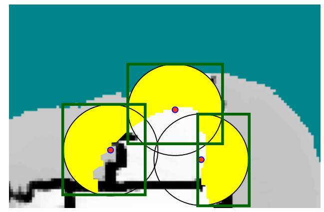
Exploration

- 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

- **Greedy strategy:** Select the candidate location with the highest utility, then repeat…



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

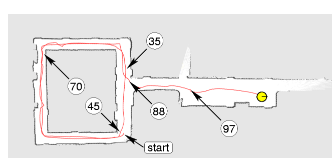
Example: Active Loop Closing

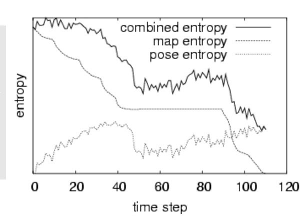
[Stachniss et al., 2005]

Example: Active Loop Closing

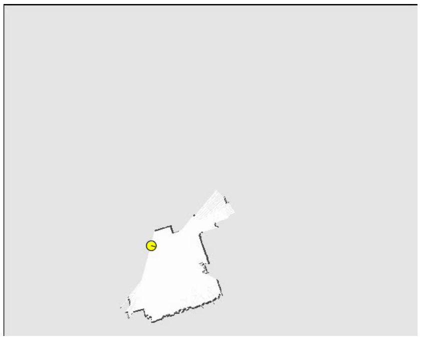
[Stachniss et al., 2005]

- Entropy evolution





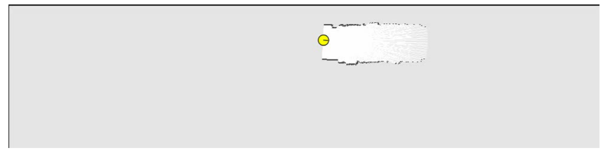
Example: Reduce uncertainty in map, path, and pose [Stachniss et al., 2005]



Selected target location

Corridor Exploration

[Stachniss et al., 2005]

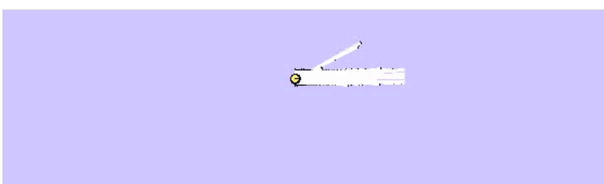


- The decision-theoretic approach leads to **intuitive behaviors:** "re-localize before getting lost"
- Some animals show a similar behavior (dogs marooned in the tundra of north Russia)

Multi-Robot Exploration

Given: Team of robots with communication

Goal: Explore the environment as fast as possible



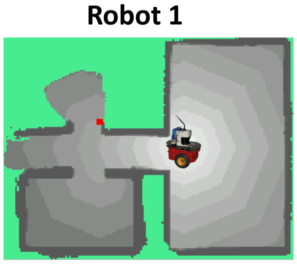
[Wurm et al., IROS 2011]

Complexity

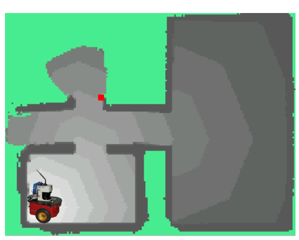
- Single-robot exploration in known, graph-like environments is in general **NP-hard**
- Proof: Reduce traveling salesman problem to exploration
- Complexity of multi-robot exploration is **exponential** in the number of robots

Motivation: Why Coordinate?

Robot 1



Robot 2



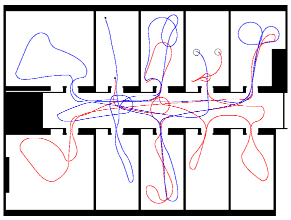
- Without coordination, two robots might choose the same exploration frontier

Levels of Coordination

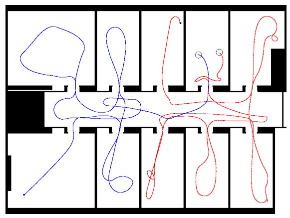
1. **No exchange of information**
2. **Implicit coordination**: Sharing a joint map
   - Communication of the individual maps and poses
   - Central mapping system
3. **Explicit coordination:** Determine better target locations to distribute the robots
   - Central planner for target point assignment
   - Minimize expected path cost / information gain / …

Typical Trajectories

Implicit coordination:

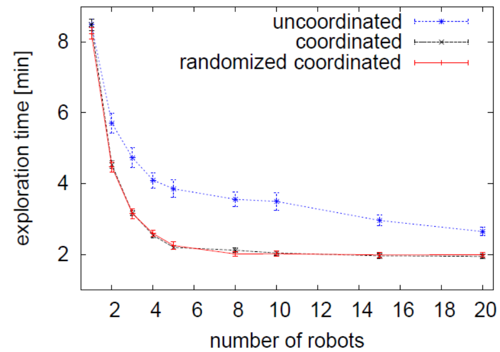


Explicit coordination:



Exploration Time

[Stachniss et al., 2006]



Coordination Algorithm

In each time step:

- Determine set of exploration targets
$$S = \{s_1, \dots, s_n\}$$
- 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

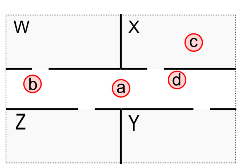
[Kuhn, 1955]

- 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

targets

robots	W	X	Y	Z
a	3	2	3	2
b	2	5	6	3
c	7	1	3	5
d	6	2	3	5

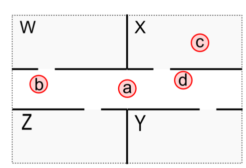


1. Compute the cost matrix (non-negative)

Hungarian Algorithm: Example

targets

robots	W	X	Y	Z
a	3	2	3	2
b	2	5	6	3
c	7	1	3	5
d	6	2	3	5

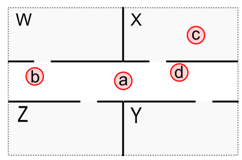


2. Find minimum element in each row

Hungarian Algorithm: Example

targets

robots	W	X	Y	Z	
a	3	2	3	2	2
b	2	5	6	3	2
c	7	1	3	5	1
d	6	2	3	5	2

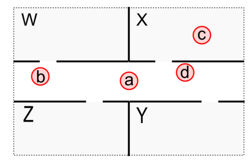


3. Subtract minimum from each row element

Hungarian Algorithm: Example

targets

robots	W	X	Y	Z
a	1	0	1	0
b	0	3	4	1
c	6	0	2	4
d	4	0	1	3
	0	0	1	0

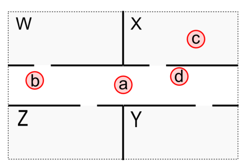


4. Find minimum element in each column

Hungarian Algorithm: Example

targets

robots	W	X	Y	Z
a	1	0	0	0
b	0	3	3	1
c	6	0	1	4
d	4	0	0	3
	0	0	1	0

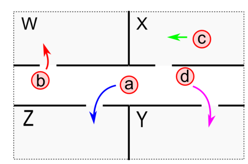


5. Subtract minimum from each column element

Hungarian Algorithm: Example

targets

robots	W	X	Y	Z
a	1	0	0	0
b	0	3	3	1
c	6	0	1	4
d	4	0	0	3



6a. Assign (if possible)

Hungarian Algorithm: Example

targets

robots	W	X	Y	Z
a	1	0	0	0
b	0	3	3	1
c	6	0	1	4
d	4	0	0	3

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

Hungarian Algorithm: Example

targets

robots	X	Y	X'	Y'
a	2	3	2	3
b	5	6	5	6
c	1	3	1	3
d	2	3	2	3

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

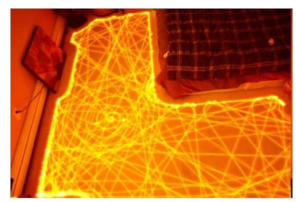
- Exploration aims at generating robot motions so that an **optimal map** is obtained
- **Coordination** reduces exploration time
- **Hungarian algorithm** efficiently solves the assignment problem (centralized, 1-step lookahead)
- Challenges (active research):
  - Limited bandwidth and **unreliable communication**
  - **Decentralized planning** and task assignment

Coverage Path Planning

- **Given:** Known environment with obstacles
- **Wanted:** The shortest trajectory that ensures complete (sensor) coverage

[images from Xu et al., ICRA 2011]

Coverage Path Planning



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

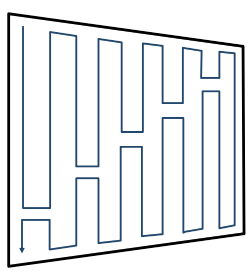
- What is a good coverage strategy?
- What would be a good cost function?
  - Amount of redundant traversals
  - Number of stops and rotations
  - Execution time
  - Energy consumption
  - Robustness
  - Probability of success
  - …

Coverage Path Planning

- 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

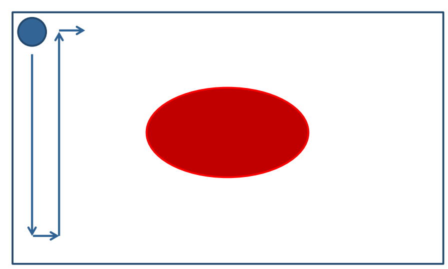
Coverage of Simple Shapes

- Approximately optimal solution often easy to compute for simple shapes (e.g., trapezoids)



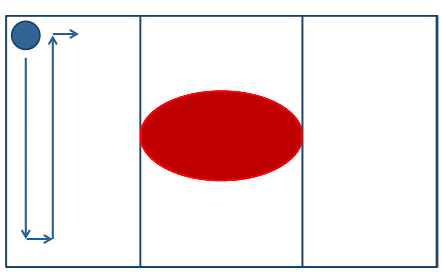
Idea

[Mannadiar and Rekleitis, ICRA 2011]



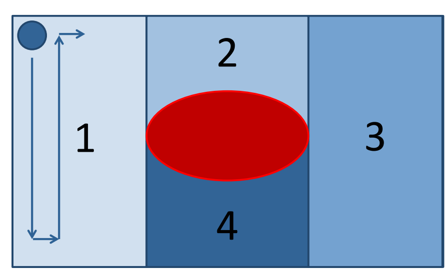
Idea

[Mannadiar and Rekleitis, ICRA 2011]



Idea

[Mannadiar and Rekleitis, ICRA 2011]



Coverage Based On Cell Decomposition

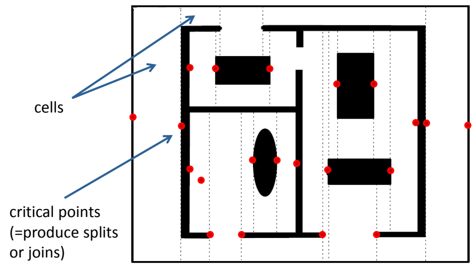
[Mannadiar and Rekleitis, ICRA 2011]

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]

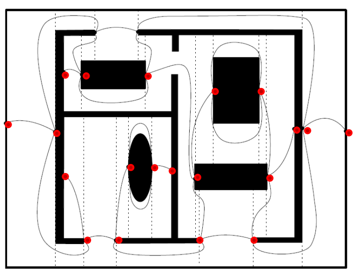
- Similar to trapezoidal decomposition
- Can be computed efficiently



Step 2: Build Reeb Graph

[Mannadiar and Rekleitis, ICRA 2011]

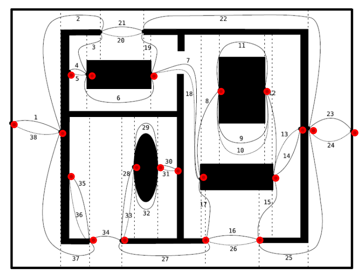
- Vertices = Critical points (that triggered the split)
- Edges = Connectivity between critical points



Step 3: Compute Euler Tour

[Mannadiar and Rekleitis, ICRA 2011]

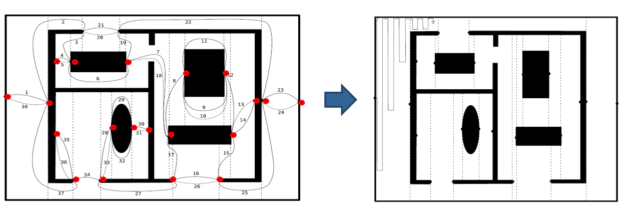
- Extend graph so that vertices have even order
- Compute Euler tour (linear time)



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

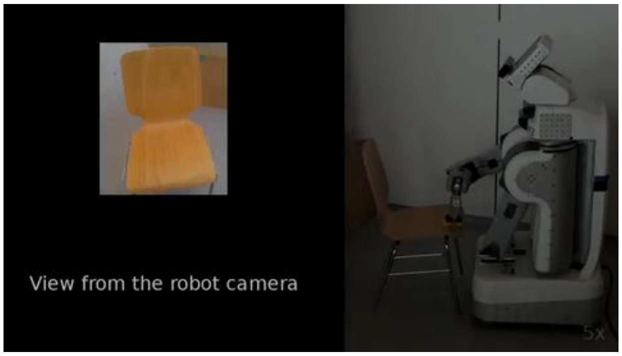
[Hess et al., IROS 2012]

- **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

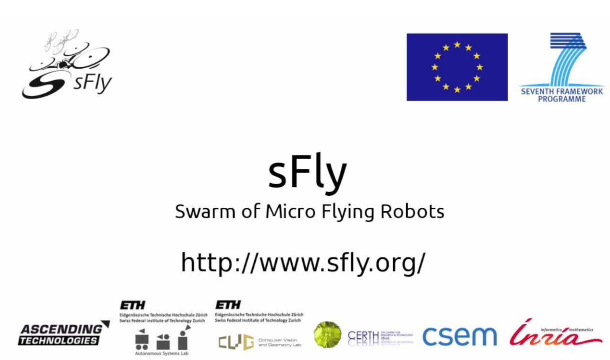
[Hess et al., IROS 2012]



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)



Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

Experimentation, Evaluation and Benchmarking

Dr. Jürgen Sturm

Agenda for Today

- Course Evaluation
- Scientific research: The big picture
- Best practices in experimentation
- Datasets, evaluation criteria and benchmarks
- Time for questions

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 – Methodology

1. Generate an idea
2. Develop an approach that solves the problem
3. Demonstrate the validity of your solution
4. Disseminate your results
5. At all stages: iteratively refine

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

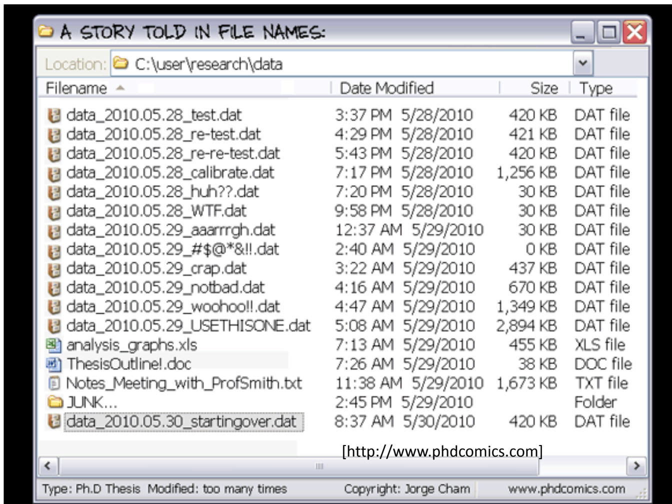
- Practitioner
  - Start programming
  - Realize that it is not going to work, start over, …
  - When it works, formalize it (try to find out why it works and what was missing before)
  - Empirically verify that it works
- Theorist
  - Formalize the problem
  - Find suitable method
  - (Theoretically) 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) **3-6 month**
  2. Submit PDF to an international conference or journal
  3. Paper will be peer-reviewed **3-6 month**
  4. Improve paper (if necessary)
  5. Give talk or poster presentation at conference **15 min.**
  6. Optionally: Repeat step 1-5 until PhD ☺ **3-5 years**



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)

Scientific Research

- This was the big picture
- Today's focus is on best practices in experimentation
- **What do you think are the (desired) properties of a good scientific experiment?**

What are the desired properties of a good scientific experiment?

- Reproducibility / repeatability
  - Document the experimental setup
  - Choose (and motivate) an your 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?

Challenges

- 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

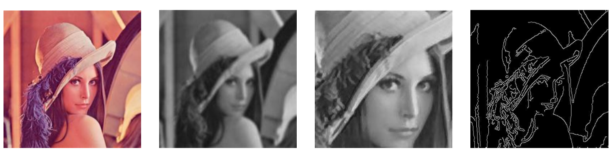
- Middlebury datasets (optical flow, stereo, …)
- Caltech-101, PASCAL (object recognition)
- Stanford bunny (3d reconstruction)

Robotics

- RoboCup competitions (robotic soccer)
- DARPA challenges (autonomous car)
- SLAM datasets

Image Denoising: Lenna Image

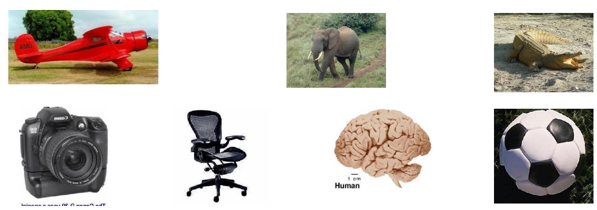
- 512x512 pixel standard image for image compression and denoising
- Lena Söderberg, Playboy magazine Nov. 1972
- Scanned by Alex Sawchuck at USC in a hurry for a conference paper



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



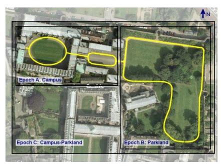
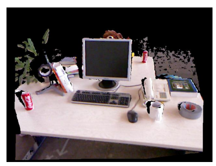
RoboCup Initiative

- 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]
- …

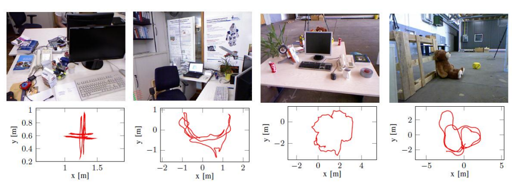
TUM RGB-D Dataset

[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, …



(a) fr1/xyz (b) fr1/room (c) fr2/desk (d) fr2/slam

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



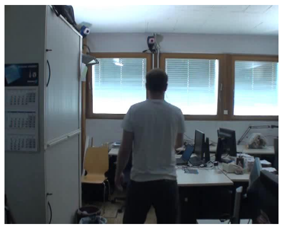
Calibration

Calibration of the overall system is not trivial:

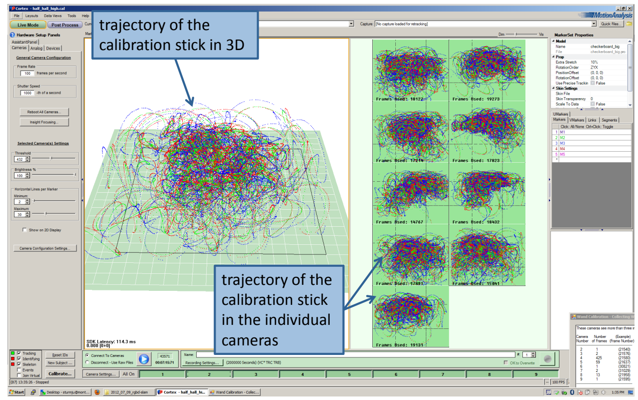
1. Mocap calibration
2. Kinect-mocap calibration
3. Time synchronization

Calibration Step 1: Mocap

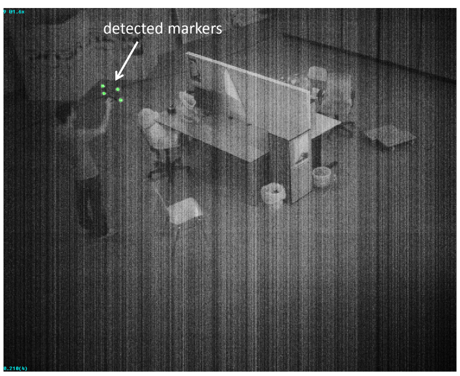
- Need at least 2 cameras for position fix
- Need at least 3 markers on object for full pose
- Calibration stick for extrinsic calibration



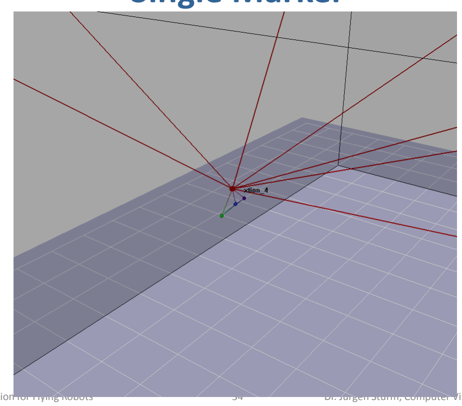
Calibration Step 1: Mocap



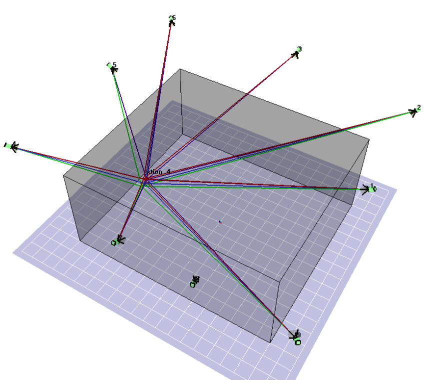
Example: Raw Image from Mocap



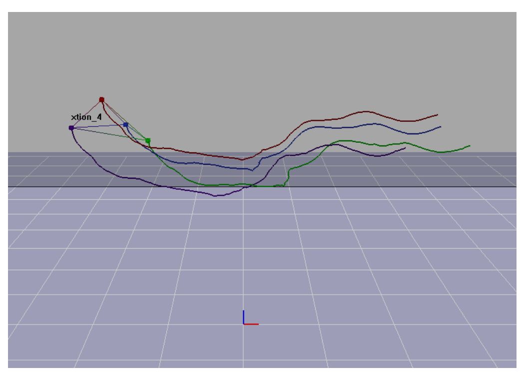
Example: Position Triangulation of a Single Marker



Example: Tracked Object (4 Markers)

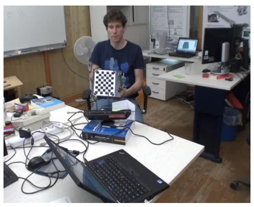
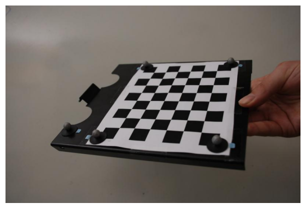


Example: Recorded Trajectory



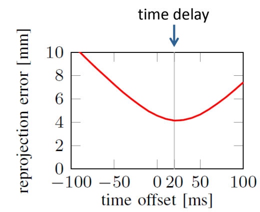
Calibration Step 2: Mocap-Kinect

- 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

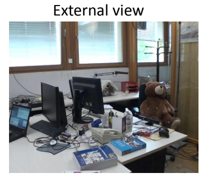



Calibration - Validation

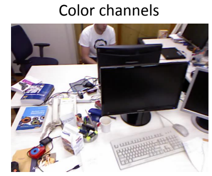
- Intrinsic calibration
- Extrinsic calibration color + depth
- Time synchronization color + depth
- Mocap system slowly drifts (need re-calibration every hour)
- Validation experiments to check the quality of calibration
  - 2mm length error on 2m rod across mocap volume
  - 4mm RMSE on checkerboard sequence

Example Sequence: Freiburg1/XYZ

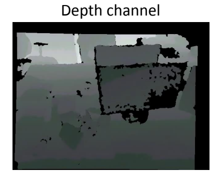
External view



Color channels

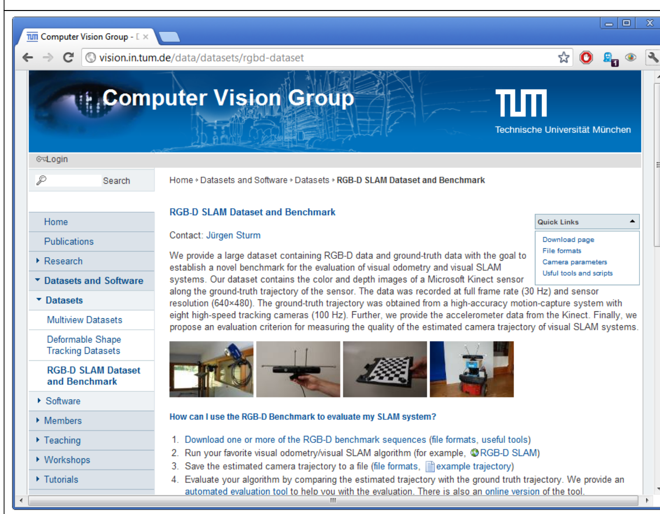


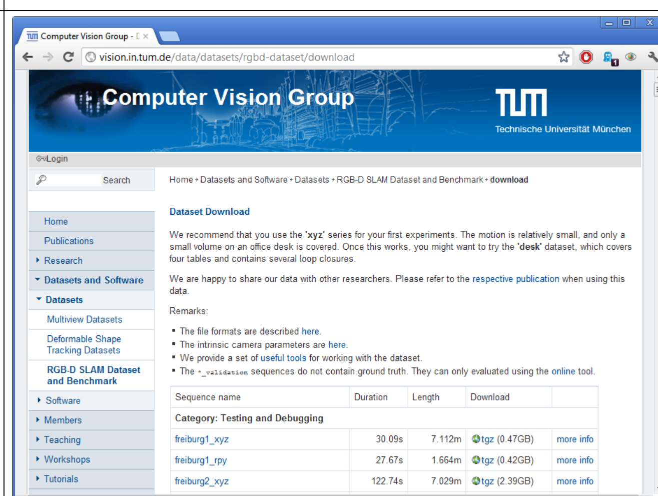
Depth channel

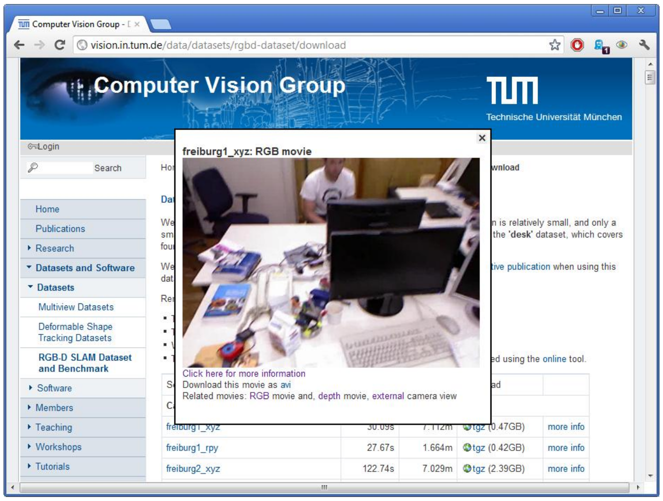


Sequence description (on the website):

"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. "







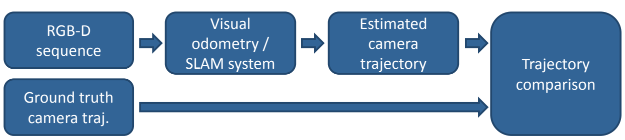
Dataset Website

- In total: 39 sequences (19 with ground truth)
- One ZIP archive per sequence, containing
  - Color and depth images (PNG)
  - Accelerometer data (timestamp ax ay az)
  - Trajectory file (timestamp tx ty ty qx qy qz qw)
- Sequences also available as ROS bag and MRPT rawlog

http://vision.in.tum.de/data/datasets/rgbd-dataset

What Is a Good Evaluation Metric?

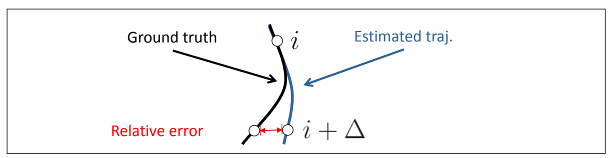
- Compare camera trajectories
  - Ground truth trajectory $Q_1, \dots, Q_n \in \mathrm{SE}(3)$
  - Estimate camera trajectory $P_1, \dots, P_n \in \mathrm{SE}(3)$
- Two common evaluation metrics
  - Relative pose error (drift per second)
  - Absolute trajectory error (global consistency)



Relative Pose Error (RPE)

- Measures the (relative) **drift**
- Recommended for the evaluation of visual odometry approaches

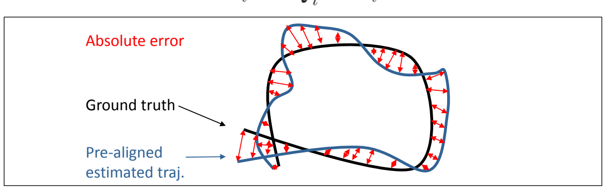
$$\underbrace{E_i}_{\text{Relative error}} := \underbrace{\left(Q_i^{-1} Q_{i+\Delta}\right)^{-1}}_{\text{True motion}} \underbrace{\left(P_i^{-1} P_{i+\Delta}\right)}_{\text{Estimated motion}}$$



Absolute Trajectory Error (ATE)

- Measures the **global error**
- Requires pre-aligned trajectories
- Recommended for SLAM evaluation

$$E_i := Q_i^{-1} S P_i$$

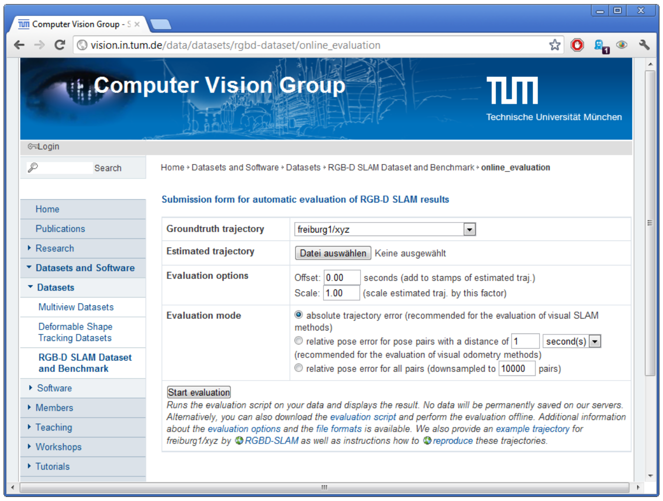


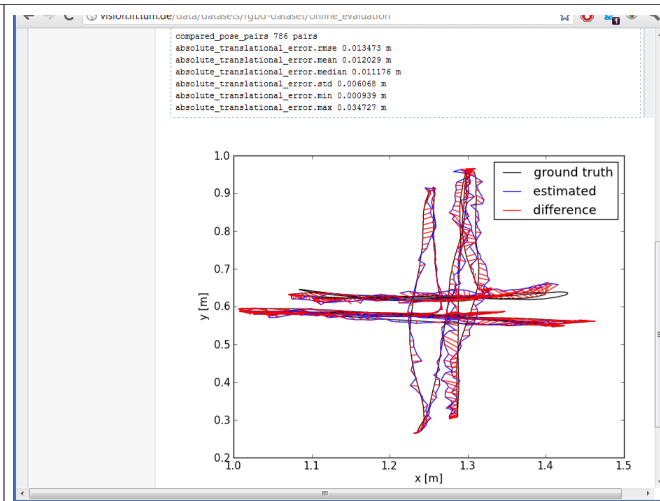
Evaluation metrics

- Average over all time steps

$$\mathrm{RMSE}(E_{1:n}) := \left(\tfrac{1}{m} \textstyle\sum_{i=1}^{m} \|trans(E_i)\|^2\right)^{1/2}$$

- 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 (!)

Time for Questions