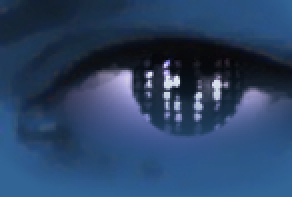


Computer Vision Group

Technische Universität München

Visual Navigation for Flying Robots

Lecture Notes Summer Term 2013

Lecturer: Dr. Jürgen Sturm

Teaching Assistants: Jakob Engel, Christian Kerl

http://vision.in.tum.de/teaching/ss2013/visnav2013

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
- Christian Kerl
- Cyrill Stachniss
- Daniel Cremers
- Frank Steinbrücker
- Friedrich Fraundorfer
- Georgio Grisetti
- Jan Peters
- Jakob Engel
- Jana Kosecka
- Jörg Müller
- Jürgen Hess
- Kai Arras
- Kai Wurm
- Korbinian Schmid
- 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

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

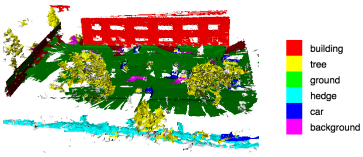
Welcome

Dr. Jürgen Sturm

Advertisement: Machine Learning for Computer Vision

Advertisement: Machine Learning for Computer Vision

- Lecture by Dr. Rudolph Triebel
- Starts Friday 26th April 9-11 am, weekly
- Exercise classes every other week
- Room 02.09.023
- Topics: Probabilistic Graphical Models, Conditional Random Fields, Kernel Methods, Gaussian Processes, Boosting, Random Forests, Clustering...
- Requirements: basic math (algebra, stochastic)
- More information: http://vision.in.tum.de/teaching/ss2013/ml_ss13



Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

Welcome

Dr. Jürgen Sturm

Organization

- Tue 10:15-11:45
  - Lectures, discussions
  - Lecturer: Jürgen Sturm
- Thu 14:00-15:30
  - Lab course, homework & programming exercises
  - Teaching assistant: Jakob Engel and Christian Kerl

Who Are We?

- Computer Vision group:
  1 Professor, 3 Postdocs, 11 PhD students
- Research topics:
  Motion estimation, 3D reconstruction, image segmentation, convex optimization, shape analysis
- My research goal:
  Apply solutions from computer vision to real-world problems in robotics.

Who Are You?

Goal of this Course

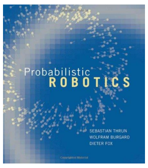
- Provide an overview on problems/approaches for autonomous quadrocopters
- Strong focus on vision as the main sensor
- Areas covered: Mobile Robotics and Computer Vision
- Hands-on experience in lab course

Course Website

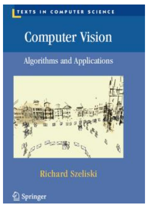
- Course Website: http://vision.in.tum.de/teaching/ss2013/visnav2013
  - Announcements
  - Schedule
  - Slides
  - Recordings
  - Exercises
- We need your feedback to improve this course!
- Let us know when you have ideas for improvement, find mistakes, …

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

Camera-based localization and mapping

Advanced topics

Lab Course

- Jakob Engel and Christian Kerl
- Thu 14:00 – 15:30
- Room 02.05.014
- Alternation of exercises and robot lab:
  - Exercises: every two weeks, discussion of homework, **participation is required**
  - Robot lab: in weeks without exercises, help with quadrocopter programming, participation recommended

Exercises

- Exercise sheets contain both theoretical and programming problems
- 3 exercise sheets + 1 mini-project
- Deadline: before lecture (Tue 10:15)
- Hand in by email (visnav2013@vision.in.tum.de)

Group Assignment and Schedule

- 5 Parrot Ardrones
- 30 students in the course, 3 students per group → 10 groups
- Either use lab computers or bring own laptop (recommended)
- List for groups and robot schedule
- You have to sign up for a team before May 2nd (team list in lab room)
- After May 2nd, remaining places will be given to students on waiting list

Lab Course

- Starts this Thursday (room 02.05.014)
- Introduction to ROS and the Ardrone
- If you bring your own laptop:
  - Pre-install ROS
  - http://www.ros.org/wiki/ROS/Installation
- If not:
  - Jakob and Christian will provide you with user accounts for the lab machines

VISNAV2013: Team Assignment

Team Name					
Student Name					
Student Name					
Student Name					

Team Name					
Student Name					
Student Name					
Student Name					

VISNAV2013: 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)

Thursday	Ardrone 1	Ardrone 2	Ardrone 3	Ardrone 4	Ardrone 5
2pm – 4pm					
4pm – 6pm					

Safety Warning

- Quadrocopters are dangerous objects
- Read the instructions 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)

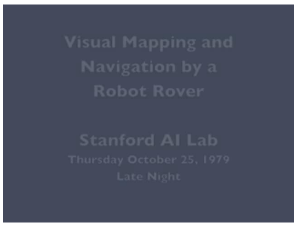
History

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)

Stanford Cart (1961-80)



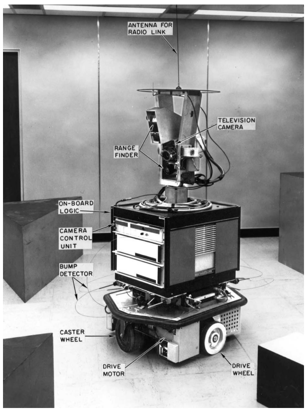
Visual Mapping and Navigation by a Robot Rover

Stanford AI Lab

Thursday October 25, 1979

Late Night

Shakey the Robot (1966-1972)



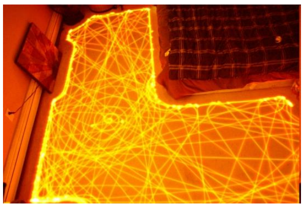
Shakey the Robot (1966-1972)

Rhino and Minerva (1998-99)

- Museum tour guide robots
- University of Bonn and CMU
- Deutsches Museum, Smithsonian Museum

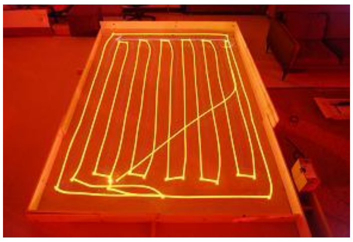
Roomba (2002)

- 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)

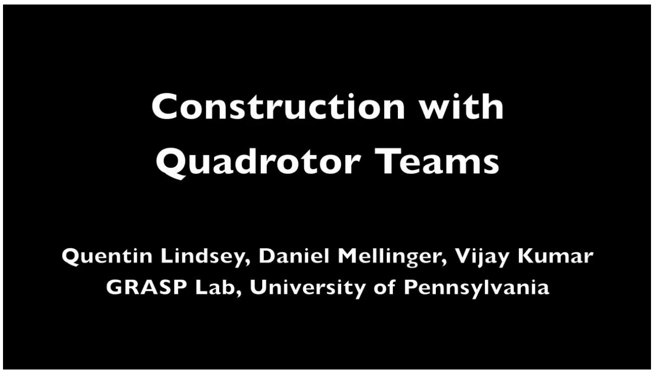
Quadrocopters (2001-)

Aggressive Maneuvers (2010)

Precise Aggressive Maneuvers for Autonomous Quadrotors

Daniel Mellinger, Nathan Michael, Vijay Kumar
GRASP Lab, University of Pennsylvania

Autonomous Construction (2011)



Construction with Quadrotor Teams

Quentin Lindsey, Daniel Mellinger, Vijay Kumar
GRASP Lab, University of Pennsylvania

Mapping with a Quadrocopter (2011)

Our Own Recent Work (2011-)

- Visual odometry (Frank Steinbrücker, Christian Kerl)
- Camera-based navigation (Jakob Engel)
- 3D Reconstruction (Erik Bylow)

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

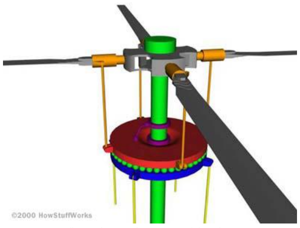
For more, see http://multicopter.org/wiki/Multicopter_Table

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 thrust
  - controlled by changing the speeds of rotation

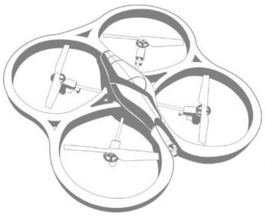
Helicopter

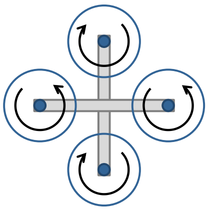
- Swash plate adjusts pitch of propeller cyclically, controls pitch and roll
- Yaw is controlled by tail rotor



©2000 HowStuffWorks

Quadrocopter

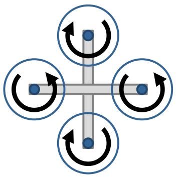




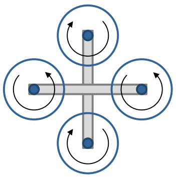
Keep position:

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

Quadrocopter: Basic Motions

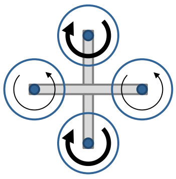


Ascend

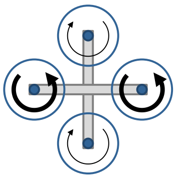


Descend

Quadrocopter: Basic Motions

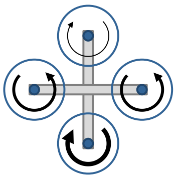


Turn Left

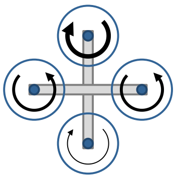


Turn Right

Quadrocopter: Basic Motions

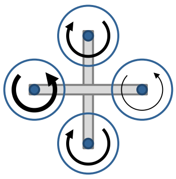


Accelerate
Forward

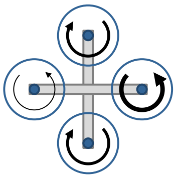


Accelerate
Backward

Quadrocopter: Basic Motions



Accelerate
to the Right



Accelerate
to the Left

Autonomous Flight

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

Challenges

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

Roboticist 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?

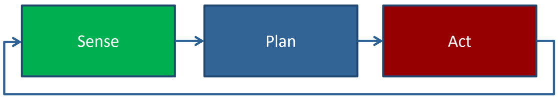
Robot Hardware/Components

- 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
- Current trends (since mid 2000s)
  - Big data
  - Cloud computing

Classical / hierarchical paradigm

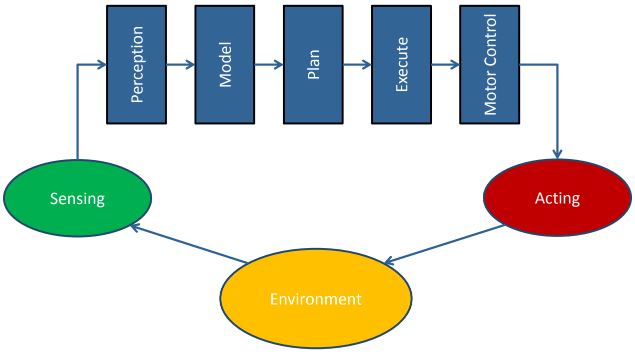


- Inspired by methods from Artificial Intelligence (70's)
- Focus on automated reasoning and knowledge representation
- STRIPS (Stanford Research Institute Problem Solver): Perfect world model, closed world assumption
- Shakey: Find boxes and move them to designated positions

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

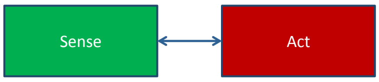


Characteristics of hierarchical paradigm

Good old-fashioned Artificial Intelligence (GOFAI):

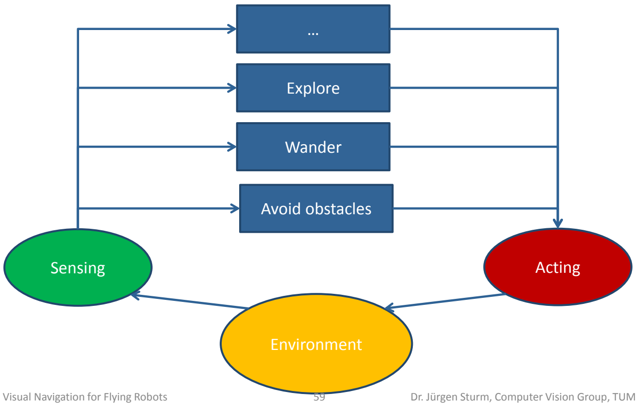
- Symbolic approaches
- Robot perceives the world, plans the next action, acts
- All data is inserted into a single, global world model
- Sequential data processing

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



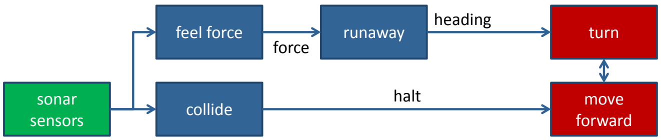
Characteristics of Reactive Paradigm

- 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)

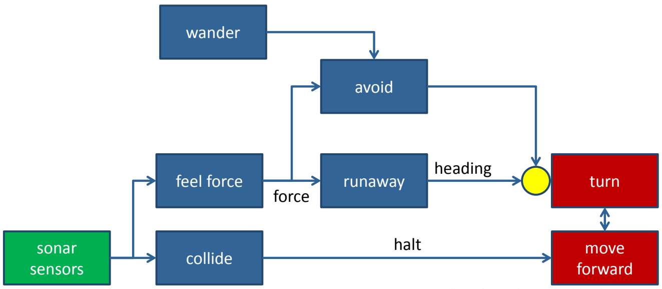
Subsumption Architecture

- 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

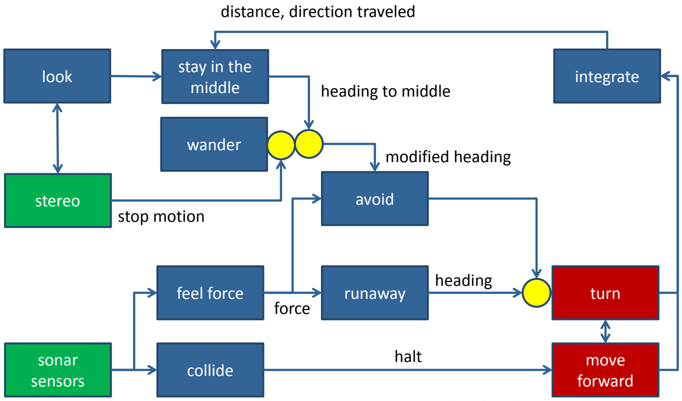
Level 1: Avoid



Level 2: Wander



Level 3: Follow Corridor



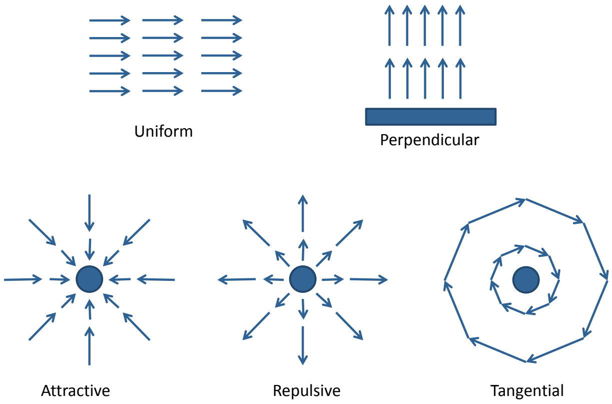
Roomba Robot

- Exercise: Model the behavior of a Roomba robot.

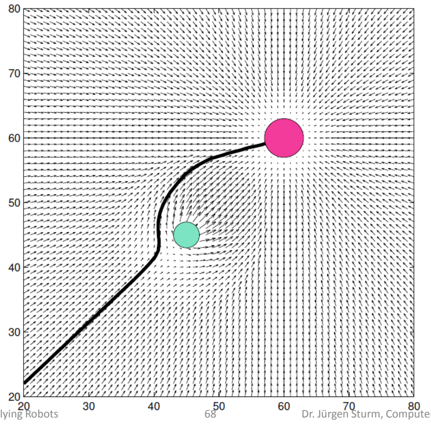
Navigation with Potential Fields

- 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

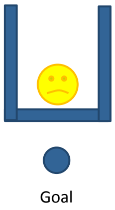


Corridor Following Robot

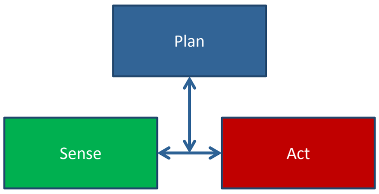
- Level 1 (collision avoidance)
  add repulsive fields for the detected obstacles
- Level 2 (wander)
  add a uniform field into a (random) direction
- Level 3 (corridor following)
  replaces the wander field by three fields (two perpendicular, one parallel to the walls)

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

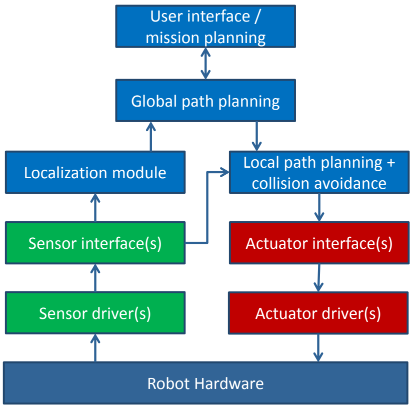
Best Practices for Robot Architectures

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

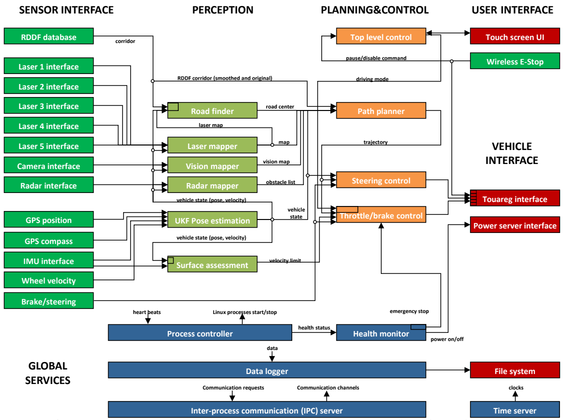
Robotic Middleware

- 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

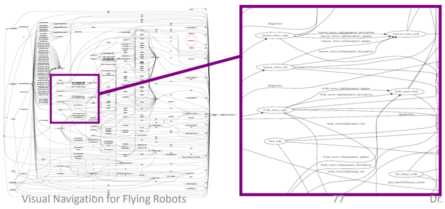


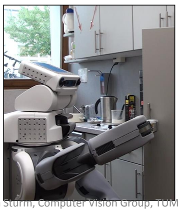
Stanley's Software Architecture



PR2 Software Architecture

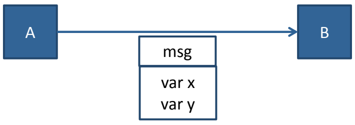
- Two 7-DOF arms, grippers, torso, 2-DOF head
- 7 cameras, 2 laser scanners
- Two 8-core CPUs, 3 network switches
- 73 nodes, 328 message topics, 174 services



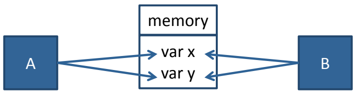


Communication Paradigms

- Message-based communication



- Direct (shared) memory access

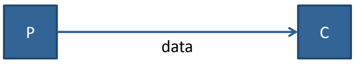


Forms of Communication

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

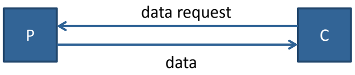
Push

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



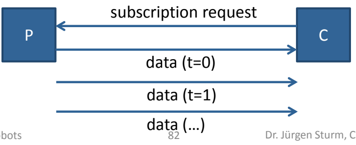
Pull

- 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



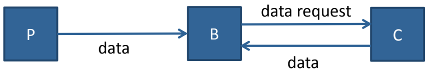
Publisher/Subscriber

- 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, …)



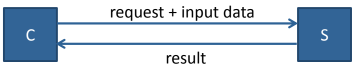
Publish to Blackboard

- 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

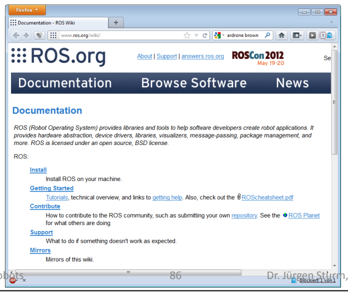


Actions (Preemptive Tasks)

- 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



Documentation

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

ROS:

Install
Install ROS on your machine.

Getting Started
Tutorials, technical overview, and links to getting help. Also, check out the ROScheatsheet.pdf

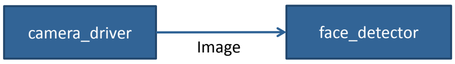
Contribute
How to contribute to the ROS community, such as submitting your own repository. See the ROS Planet for what others are doing

Support
What to do if something doesn't work as expected.

Mirrors
Mirrors of this wiki.

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



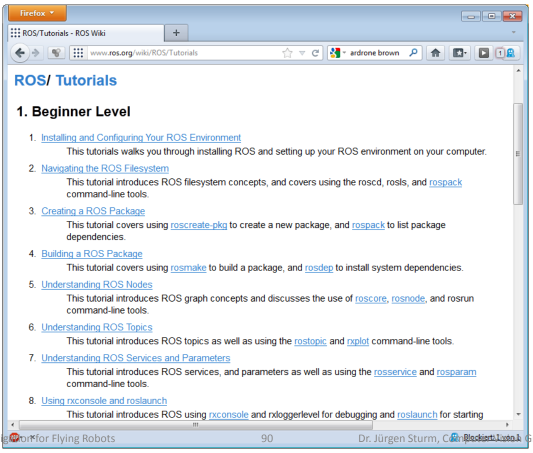
Software Management

- 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



ROS/ Tutorials

1. Beginner Level

1. Installing and Configuring Your ROS Environment
   This tutorials walks you through installing ROS and setting up your ROS environment on your computer.
2. Navigating the ROS Filesystem
   This tutorial introduces ROS filesystem concepts, and covers using the roscd, rosls, and rospack command-line tools.
3. Creating a ROS Package
   This tutorial covers using roscreate-pkg to create a new package, and rospack to list package dependencies.
4. Building a ROS Package
   This tutorial covers using rosmake to build a package, and rosdep to install system dependencies.
5. Understanding ROS Nodes
   This tutorial introduces ROS graph concepts and discusses the use of roscore, rosnode, and rosrun command-line tools.
6. Understanding ROS Topics
   This tutorial introduces ROS topics as well as using the rostopic and rxplot command-line tools.
7. Understanding ROS Services and Parameters
   This tutorial introduces ROS services, and parameters as well as using the rosservice and rosparam command-line tools.
8. Using rxconsole and roslaunch
   This tutorial introduces ROS using rxconsole and rxloggerlevel for debugging and roslaunch for starting

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: Lab Course

- Robot lab: room 02.05.14 (different room!)
- Lecture: room 02.09.23 (here)
- You have to sign up for a team before May 2nd (team list in student lab)
- After May 2nd, remaining places will be given to students on waiting list
- First exercise sheet is due **next Tuesday 10am**

Today's Agenda

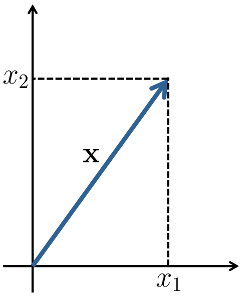
- Linear algebra
- 2D and 3D geometry
- Sensors
- First exercise sheet

Vectors

- Vector and its coordinates

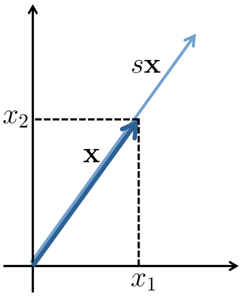
$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

- Vectors represent points in an n-dimensional space



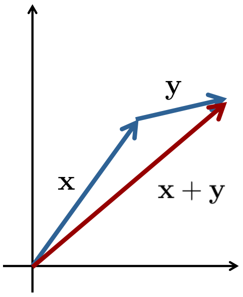
Vector Operations

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



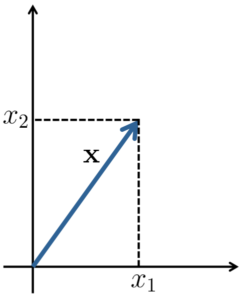
Vector Operations

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



Vector Operations

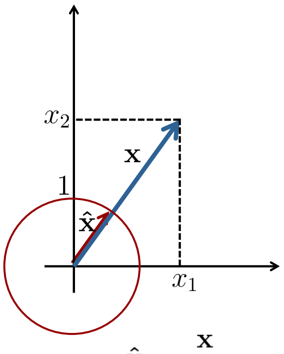
- Scalar multiplication
- Addition/subtraction
- **Length**
- Normalized vector
- Dot product
- Cross product



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

Vector Operations

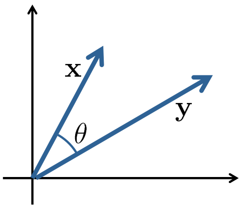
- Scalar multiplication
- Addition/subtraction
- Length
- **Normalized vector**
- Dot product
- Cross product



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

Vector Operations

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



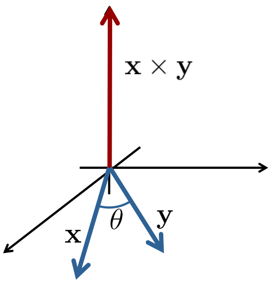
$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos\theta$$

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

$\mathbf{y}$ is lin. 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

$$\mathbf{x} \times \mathbf{y} = \begin{pmatrix} x_2 y_3 - x_3 y_2 \\ x_3 y_1 - x_1 y_3 \\ x_1 y_2 - x_2 y_1 \end{pmatrix}$$

- 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

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

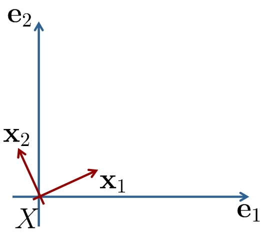
- First index refers to row
- Second index refers to column

Matrices

- Column vectors of a matrix

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

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



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

Matrices

- Row vectors of a matrix

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

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 $X = -X^\top (= \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix})$
- (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

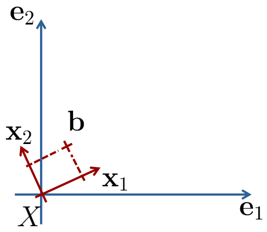
Matrix Operations

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

Matrix-Vector Multiplication

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

column vectors



- Geometric interpretation:
  A linear combination of the columns of A scaled by the coefficients of **b**
  → coordinate transf. from local to global frame

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 $C = AB$

$$= A \begin{pmatrix} \mathbf{b}_{*1} & \mathbf{b}_{*2} & \cdots \mathbf{b}_{*p} \end{pmatrix}$$

- 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

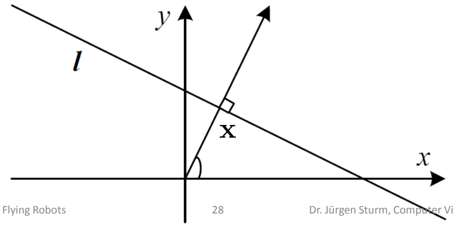
- Homogeneous vectors that differ only be scale represent the same 2D point
- Convert back to inhomogeneous coordinates by dividing through last element

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

- 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 $\bar{\mathbf{x}} \cdot \tilde{\mathbf{l}} = ax + by + c = 0$

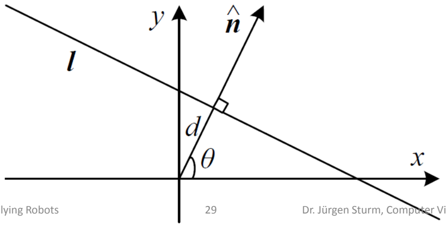


Geometric Primitives in 2D

- Normalized line equation vector

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

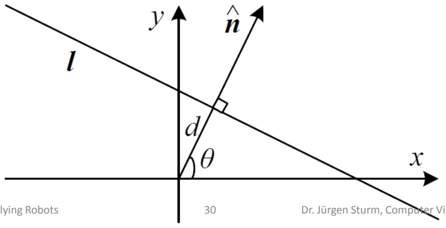
where d is the distance of the line to the origin



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

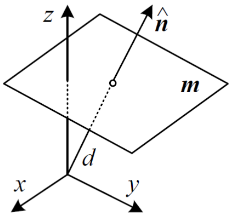
- Line joining two points $\tilde{\mathbf{l}} = \tilde{\mathbf{x}}_1 \times \tilde{\mathbf{x}}_2$
- Intersection point of two lines $\tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$

Geometric Primitives in 3D

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

Geometric Primitives in 3D

- 3D plane $\tilde{\mathbf{m}} = (a, b, c, d)^\top$
- 3D plane equation $\bar{\mathbf{x}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$
- 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



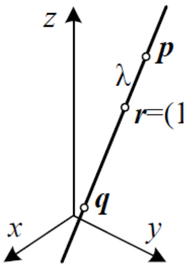
Geometric Primitives in 3D

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

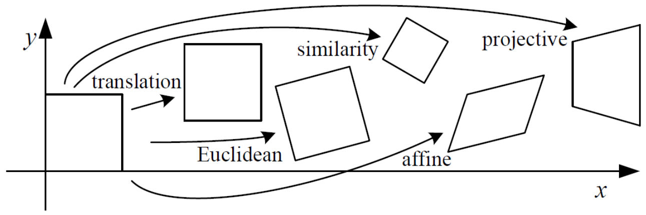
$$\lambda \in \mathbb{R}$$

- Line segment joining $\mathbf{p}, \mathbf{q}$:

$$0 \le \lambda \le 1$$



2D Planar Transformations



2D Transformations

- Translation

$$\mathbf{x}' = \mathbf{x} + \mathbf{t}$$

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

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

where $\mathbf{t} \in \mathbb{R}^2$ is the translation vector,
$\mathbf{I}$ is the identity matrix, and $\mathbf{0}$ is the zero vector

2D Transformations

- Translation

$$\mathbf{x}' = \mathbf{x} + \mathbf{t}$$

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

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

Question: How many DOFs has this transformation?

where $\mathbf{t} \in \mathbb{R}^2$ is the translation vector,
$\mathbf{I}$ is the identity matrix, and $\mathbf{0}$ is the zero vector

2D Transformations

- Rigid body motion or Euclidean transformation (rotation + translation)

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

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
- Lines remain lines :-)

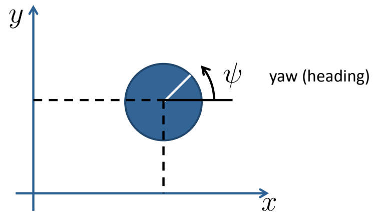
2D Transformations

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	⏢

Examples: Euclidean Transformations

Coordinate Transforms

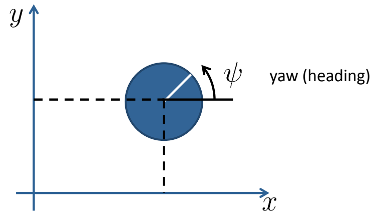
- Robot is located somewhere in space



Coordinate Transforms

- Robot is located somewhere in space

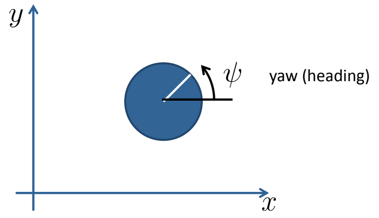
$$X = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} \cos\psi & -\sin\psi & x \\ \sin\psi & \cos\psi & y \\ 0 & 0 & 1 \end{pmatrix} \in \mathrm{SE}(2) \subset \mathbb{R}^{3x3}$$



Coordinate Transforms

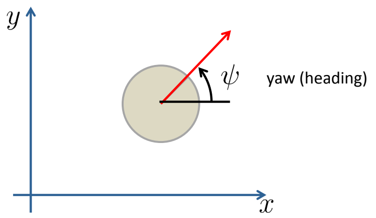
- Robot is located at x=0.7, y=0.5, yaw=45deg

$$X = \begin{pmatrix} \cos 45 & -\sin 45 & 0.7 \\ \sin 45 & \cos 45 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix}$$



Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves 1m forward
- What is its position afterwards?



Vector Transformation

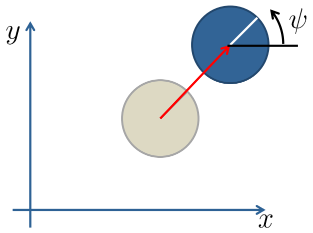
- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves 1m forward

Inhomogeneous coordinates

$$\mathbf{v}_{\text{local}} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Homogeneous coordinates

$$\tilde{\mathbf{v}}_{\text{local}} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$



Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves 1m forward

$$\tilde{\mathbf{v}}_{\text{global}} = X \tilde{\mathbf{v}}_{\text{local}}$$

Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves 1m forward

$$\begin{aligned} \tilde{\mathbf{v}}_{\text{global}} &= X \tilde{\mathbf{v}}_{\text{local}} \\ &= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \end{aligned}$$

Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves 1m forward

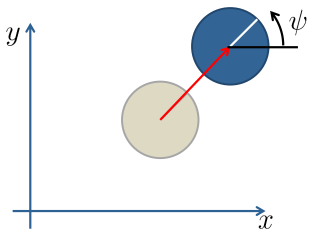
$$\begin{aligned} \tilde{\mathbf{v}}_{\text{global}} &= X \tilde{\mathbf{v}}_{\text{local}} \\ &= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 1.41 \\ 1.21 \\ 1 \end{pmatrix} \end{aligned}$$

Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves 1m forward

$$\tilde{\mathbf{v}}_{\text{local}} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

$$\tilde{\mathbf{v}}_{\text{global}} = \begin{pmatrix} 1.41 \\ 1.21 \\ 1 \end{pmatrix}$$



Vector Transformation

- We transformed local to global coordinates
- Sometimes we need to do the inverse
- How can we transform global coordinates into local coordinates?

$$\tilde{\mathbf{v}}_{\text{global}} = X \tilde{\mathbf{v}}_{\text{local}}$$

Vector Transformation

- We transformed local to global coordinates
- Sometimes we need to do the inverse
- How can we transform global coordinates into local coordinates?

$$\tilde{\mathbf{v}}_{\text{global}} = X \tilde{\mathbf{v}}_{\text{local}}$$

$$\tilde{\mathbf{v}}_{\text{local}} = X^{-1} \tilde{\mathbf{v}}_{\text{global}}$$

Inverse Transformations

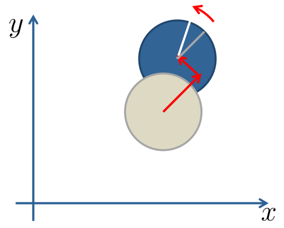
- We transformed local to global coordinates
- Sometimes we need to do the inverse
- How can we transform global coordinates into local coordinates?

$$\tilde{\mathbf{v}}_{\text{global}} = X \tilde{\mathbf{v}}_{\text{local}} = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tilde{\mathbf{v}}_{\text{local}}$$

$$\tilde{\mathbf{v}}_{\text{local}} = X^{-1} \tilde{\mathbf{v}}_{\text{global}} = \begin{pmatrix} R^\top & -R^\top \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tilde{\mathbf{v}}_{\text{global}}$$

Coordinate System Transformations

- Now consider a different motion
- Robot moves 0.2m forward, 0.1m sideways, and turns by 10deg



Coordinate System Transformations

- Robot moves 0.2m forward, 0.1m sideways, and turns by 10deg

$$U_1 = \begin{pmatrix} \cos 10 & -\sin 10 & 0.2 \\ \sin 10 & \cos 10 & 0.1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.98 & -0.17 & 0.2 \\ 0.17 & 0.98 & 0.1 \\ 0 & 0 & 1 \end{pmatrix}$$

Coordinate System Transformations

- After this motion, the robot pose (in the global frame) becomes

$$X_2 = XU$$

$$= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.98 & -0.17 & 0.2 \\ 0.17 & 0.98 & 0.1 \\ 0 & 0 & 1 \end{pmatrix} = \cdots$$

Coordinate System Transformations

Note: The order matters

- Move 1m forward, then turn 90deg left
- Turn 90deg left, then move 1m forward

$$AB \neq BA$$

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 Euclidean Transformtions

- Translation $\mathbf{t}$ has 3 degrees of freedom
- Rotation R has 3 degrees of freedom

$$X = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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 $R \in \mathrm{SO}(3)$
- What operations do we typically do with rotation matrices?

Rotation Matrix

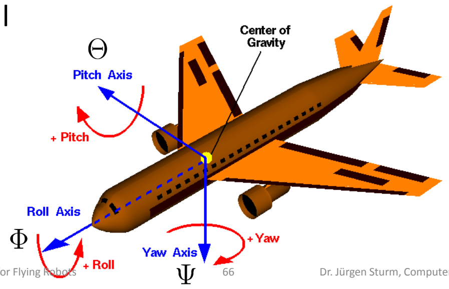
- Orthonormal 3x3 matrix

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

- Advantage: Can be easily concatenated and inverted (how?)
- Disadvantage: Over-parameterized (9 parameters instead of 3)

Euler Angles

- Product of 3 consecutive rotations (e.g., around X-Y-Z axes)
- Roll-pitch-yaw convention is very common in aerial



Roll-Pitch-Yaw Convention

- 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

$$\phi = \mathrm{Atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right)$$

$$\psi = -\mathrm{Atan2}\left(\frac{r_{21}}{\cos(\phi)}, \frac{r_{11}}{\cos(\phi)}\right)$$

$$\theta = \mathrm{Atan2}\left(\frac{r_{32}}{\cos(\phi)}, \frac{r_{33}}{\cos(\phi)}\right)$$

Euler Angles

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

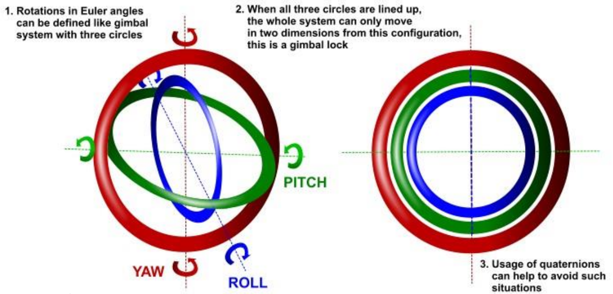
Euler Angles

- Euler angles (3 parameters)
  - Concatenation: convert to rotation matrix, multiply, convert back
  - Inverse: convert to rotation matrix, invert, convert back

$$\begin{aligned} &R_Z(\psi_1)R_Y(\theta_1)R_X(\phi_1) \cdot R_Z(\psi_2)R_Y(\theta_2)R_X(\phi_2) \\ &\neq R_Z(\psi_1+\psi_2)R_Y(\theta_1+\theta_2)R_X(\phi_1+\phi_2) \end{aligned}$$

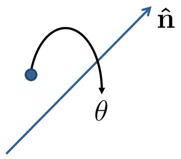
Gimbal Lock

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


1. Rotations in Euler angles can be defined like gimbal system with three circles
2. When all three circles are lined up, the whole system can only move in two dimensions from this configuration, this is a gimbal lock
PITCH
YAW
ROLL
3. Usage of quaternions can help to avoid such situations

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?)



Conversion

- Rodriguez' formula

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

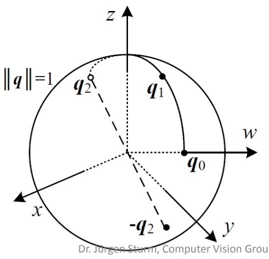
- Inverse

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

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

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 efficient
- 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)

$$\begin{aligned}\mathbf{q}^{-1} &= (\mathbf{v}, w)^{-1} \\ &= (\mathbf{v}, -w)\end{aligned}$$

Unit Quaternions

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

$$\mathbf{p}' = \mathbf{v}\bar{\mathbf{p}}\mathbf{q}^{-1}$$

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

- Relation to Axis/Angle representation

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

3D Orientations

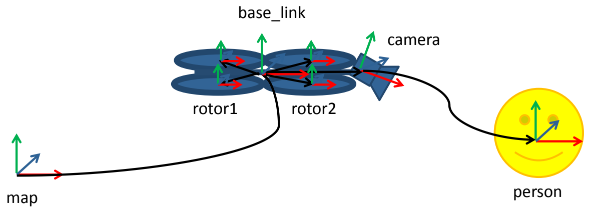
- **Note:** In general, it is very hard to "read" 3D orientations/rotations, no matter in what representation
- **Observation:** They are usually easy to visualize and can then be intuitively interpreted
- **Advice:** Use 3D visualization tools for debugging (RVIZ, libqglviewer, …)

C++ Libraries for Lin. Alg./Geometry

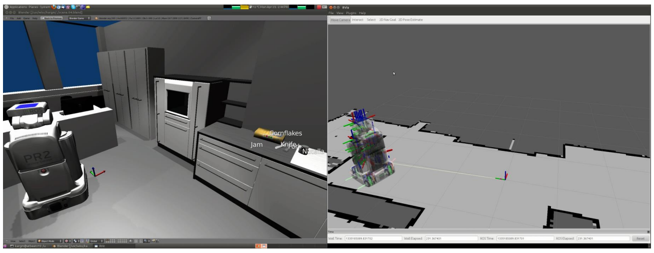
- Many C++ libraries exist for linear algebra and 3D geometry
- Typically conversion necessary
- Examples:
  - C arrays, std::vector (no linear alg. functions)
  - gsl (gnu scientific library, many functions, plain C)
  - boost::array (used by ROS messages)
  - Bullet library (3D geometry, used by ROS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



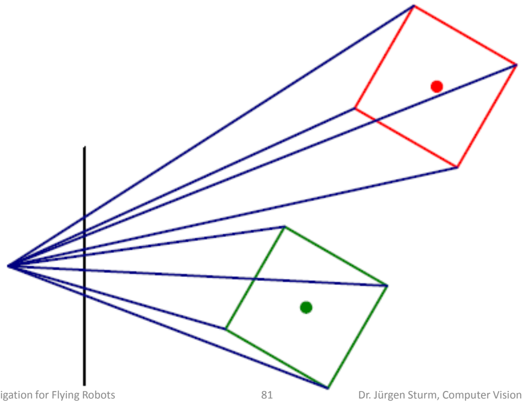
Example: Video from PR2



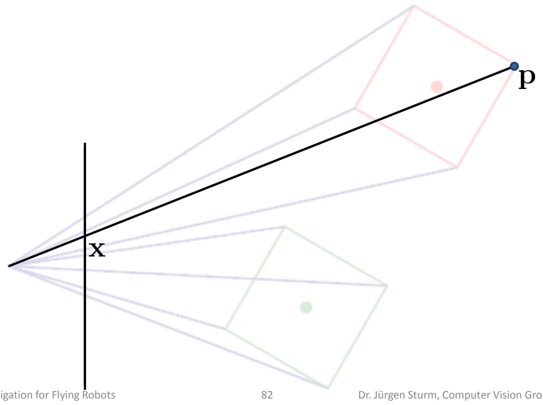
3D to 2D Projections

- 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

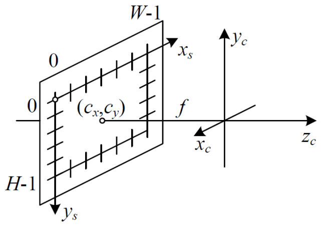
$$\tilde{\mathbf{x}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tilde{\mathbf{p}}$$

- 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)

Sensors

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

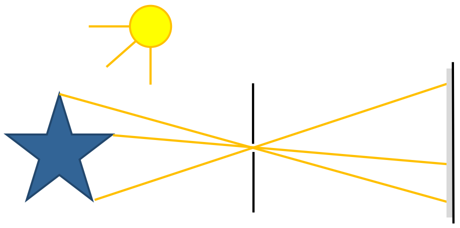
- Bandwidth or Frequency
- Delay
- Sensitivity
- Cross-sensitivity (cross-talk)
- Error (accuracy)
  - Deterministic errors (modeling/calibration possible)
  - Random errors
- Weight, power consumption, …

Let’s Have a Closer Look

- Cameras
- Gyroscope
- Accelerometers
- GPS
- Range sensors

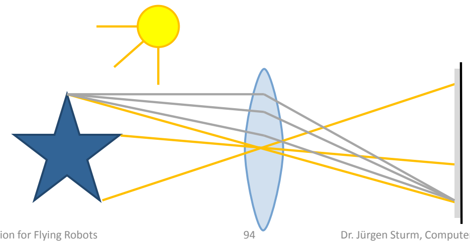
Pinhole Camera

- Lit scene emits light
- Film/sensor is light sensitive



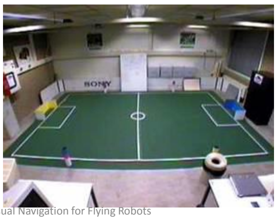
Lens Camera

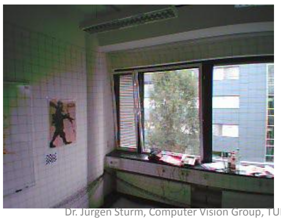
- Lit scene emits light
- Film/sensor is light sensitive
- A lens focuses rays onto the film/sensor



Real Cameras

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





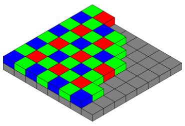
Radial Distortion

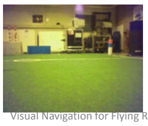
- Radial distortion of the image
  - Caused by imperfect lenses
  - Deviations are most noticeable for rays that pass through the edge of the lens
- Typically compensated with a low-order polynomial

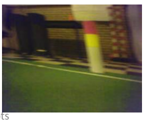
$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$
$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$

Digital Cameras

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

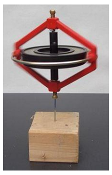


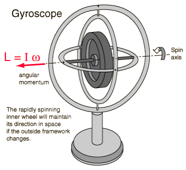




Mechanical Gyroscope

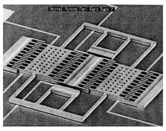
- Measures orientation (standard gyro) or angular velocity (rate gyro, needs integration for angle)
- Spinning wheel mounted in a gimbal device (can move freely in 3 dimensions)
- Wheel keeps orientation due to angular momentum (standard gyro)

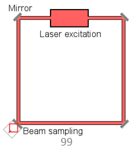


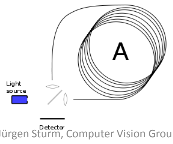

Gyroscope
L = I ω
angular momentum
Spin axis
The rapidly spinning inner wheel will maintain its direction in space if the outside framework changes.

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

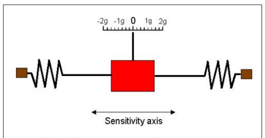
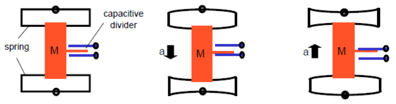
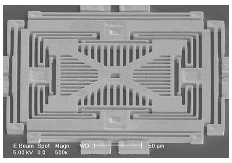


Fig. 1. Schematic structure of an accelerometer

MEMS Accelerometers

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

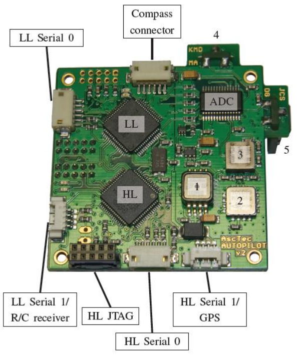




Inertial Measurement Unit

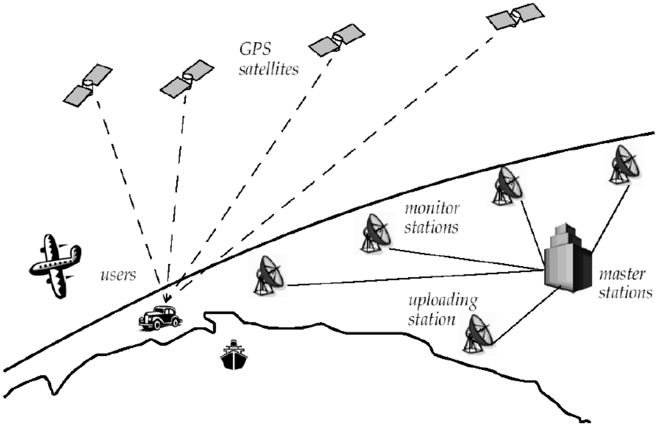
- 3-axes MEMS gyroscope
  - Provides angular velocity
  - Integrate for angular position
  - Problem: Drifts slowly over time (e.g., 1deg/hour), called the bias
- 3-axes MEMS accelerometer
  - Provides accelerations (including gravity)
- Can we use these sensors to estimate our position?

Example: AscTec Autopilot Board



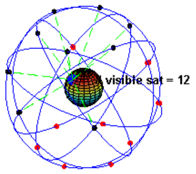
1:	pressure sensor	3:	yaw gyro
2:	acceleration sensor	4:	nick gyro
		5:	roll gyro

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

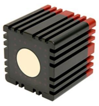
- Position from pseudorange
  - Requires measurements of 4 different satellites
  - Low accuracy (3-15m) but absolute
- Position from pseudorange + phase shift
  - Very precise (1mm) but highly ambiguous
  - Requires reference receiver (RTK/dGPS) to remove ambiguities

Range Sensors

- **Sonar**

- **Laser range finder**

- Time of flight camera



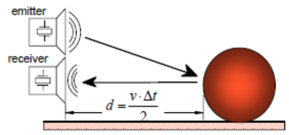
- Structured light
  (will be covered later)

Range Sensors

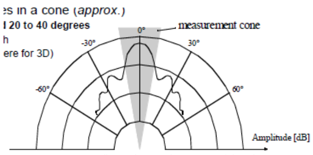
- Emit signal to determine distance along a ray
- Make use of propagation speed of ultrasound/light
- Traveled distance is given by $d = c \cdot t$
- Sound speed: 340m/s
- Light speed: 300.000km/s

Ultrasonic Range Sensors

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

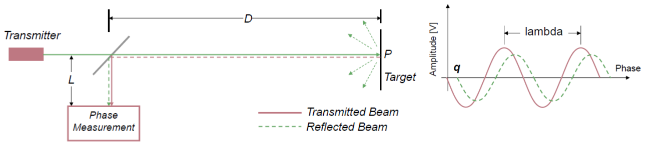


$$d = \frac{v \cdot \Delta t}{2}$$



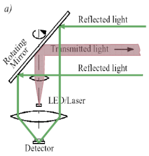
Laser Scanner

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

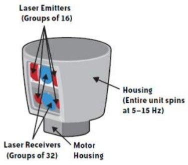


Laser Scanner

- 2D scanners



- 3D scanners


Laser Emitters (Groups of 16)
Housing (Entire unit spins at 5–15 Hz)
Laser Receivers (Groups of 32)
Motor Housing

Exercise Sheet 1

Coordinate Systems

- The pose of a robot can be described by 6 parameters:
  - Three-dimensional Cartesian coordinates
  - Three Euler angles roll, pitch, yaw.
- The state space of such a system is six-dimensional

$$\mathbf{x}_t = (x, y, z, \phi, \theta, \psi)^\top$$

- Robot makes sensor observations usually in its ego-centric frame (as seen by the robot)

Odometry Motion Model

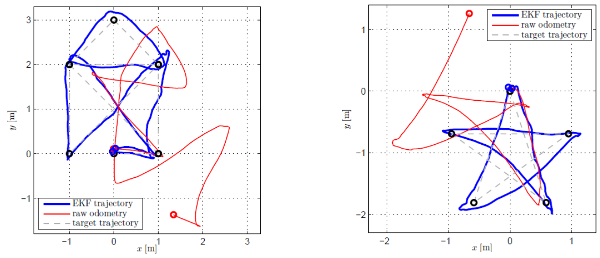
- In practice, one often finds two types of motion models:
  - **Odometry-based**
  - **Velocity-based** (**dead reckoning**)
- Odometry-based models are used when systems are equipped with distance sensors (e.g., wheel encoders).
- Velocity-based models have to be applied when no wheel encoders are given.

Dead Reckoning

- Mathematical procedure to determine the present location of a vehicle
- Achieved by calculating the current pose of the vehicle based on its velocities and the elapsed time

Dead Reckoning

- Estimating the position $\mathbf{x}_t$ based on the issued controls (or IMU readings) $\mathbf{u}_t$
- Integrate 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 $\mathbf{u}_t$ (in provided bag file)
  - Horizontal speed (vx/vy) in its local frame (!)
  - Height (z) in the global frame
  - Roll, Pitch, Yaw in the global frame
- Integrate these values to get robot pose $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t)$
  - Position (x/y/z) in the global frame
  - Orientation (e.g., r/p/y) in the global frame

Lessons Learned Today

- Linear algebra
- 2D/3D geometry
- Sensors
- Exercise sheet 1: Robot odometry
  - Due next Tuesday, 10am
  - Hand in via email to visnav2013@vision.in.tum.de

Computer Vision Group
Prof. Daniel Cremers

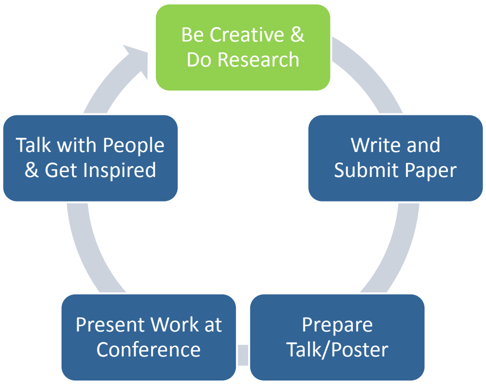
Technische Universität München

Visual Navigation for Flying Robots

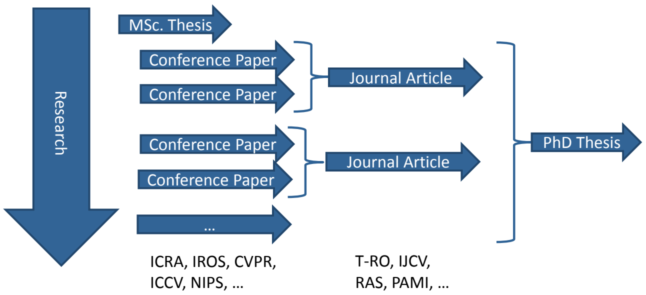
Probabilistic Models and State Estimation

Dr. Jürgen Sturm

Scientific Research


Be Creative & Do Research
Write and Submit Paper
Prepare Talk/Poster
Present Work at Conference
Talk with People & Get Inspired

Flow of Publications



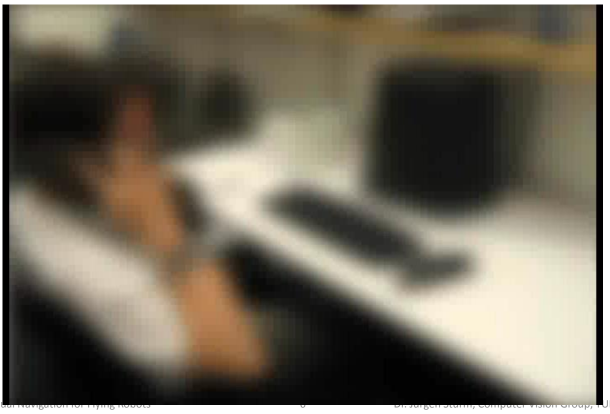
Important Conferences

- Robotics
  - International Conference on Robotics and Automation (ICRA) ➡ next week
  - International Conference on Intelligent Robots and Systems (IROS)
  - Robotics: Science and Systems (RSS)
- Computer Vision
  - International Conference on Computer Vision (ICCV)
  - Computer Vision and Pattern Recognition (CVPR)
  - German Conference on Computer Vision (GCPR)

ICRA: Preview

- Christian’s work got nominated for the “Best Vision Paper”
- Four sessions (with 6 papers each) on flying robots
- Interesting sessions:
  - Localization
  - Theory and Methods for SLAM
  - Visual Servoing
  - Sensor Fusion
  - Trajectory Planning for Aerial Robots
  - Novel Aerial Robots
  - Aerial Robots and Manipulation
  - Modeling & Control of Aerial Robots
  - Sensing for Navigation
  - ..
- Will report on interesting papers!

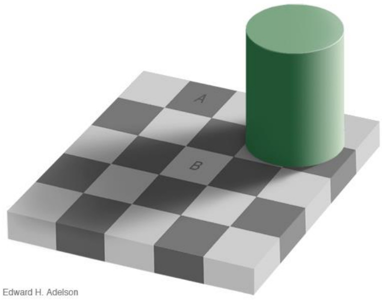
Perception



Perception

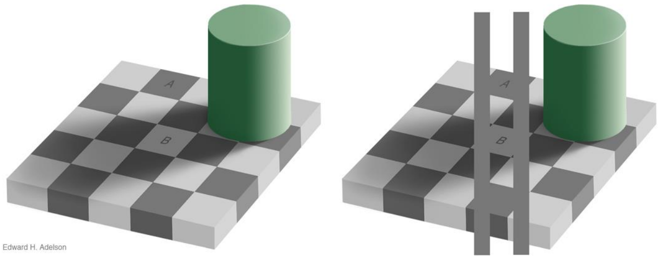
Perception

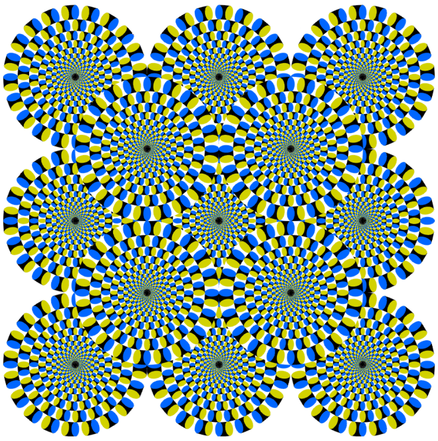
- Perception and models are strongly linked



Perception

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

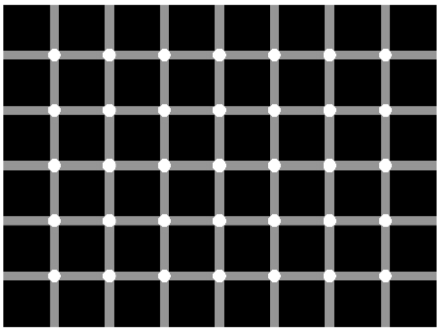




more on http://michaelbach.de/ot/index.html

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

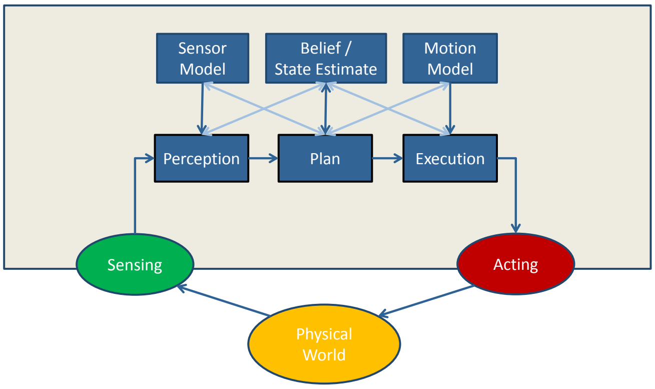
What parts of the world state are (most) relevant for a flying robot?

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

$$\underset{\uparrow\,\text{sensor reading}}{z} = \underset{\uparrow\,\text{observation function}}{h}(\underset{\uparrow\,\text{world state}}{x})$$

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

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

(Deterministic) Motion Model

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

- Update belief state according to motion model

$$\underset{\uparrow\,\text{current state}}{x'} = \overset{\text{transition function}\,\downarrow}{g}(\underset{\uparrow\,\text{previous state}}{x}, \overset{\text{executed action}\,\downarrow}{u})$$

Probabilistic Robotics

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

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)

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

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

Agenda for Today

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

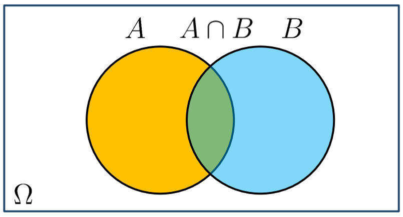
The Axioms of Probability Theory

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: $P(\text{Room}) =< 0.7, 0.2, 0.08, 0.02 >$

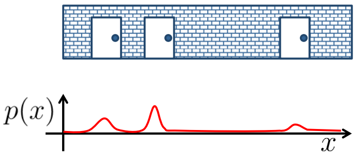
  $\text{Room} \in \{\text{office}, \text{corridor}, \text{lab}, \text{kitchen}\}$

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
  $$P(x \mid y) = P(x)$$

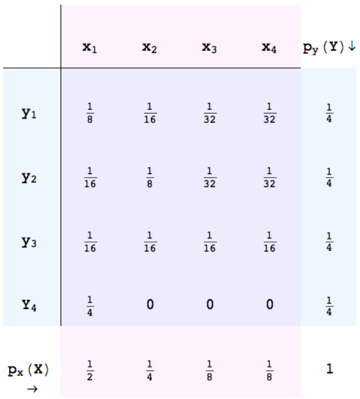
Conditional Independence

- 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

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

Estimation from Data

- Observations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^d$
- Sample Mean $\boldsymbol{\mu} = \frac{1}{n} \sum_i \mathbf{x}_i$
- Sample Covariance

$$\Sigma = \frac{1}{n-1} \sum_i (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top$$

The State Estimation Problem

We want to estimate the world state x from

1. Sensor measurements z and
2. 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

$$P(x \mid z) = \frac{\overset{\downarrow}{P(z \mid x)}\overset{\downarrow}{P(x)}}{\underset{\uparrow}{P(z)}}$$

prior on sensor observations

Bayes Formula

- Derivation of Bayes Formula

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

- Our usage

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

Bayes Formula

- Derivation of Bayes Formula

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

- Our usage

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

Normalization

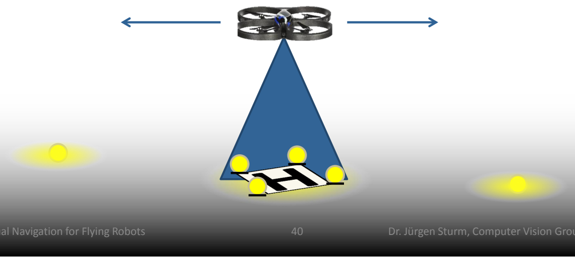
- Direct computation of $P(z)$ can be difficult
- Idea: Compute improper distribution, normalize afterwards
- Step 1: $L(x \mid z) = P(z \mid x)P(x)$
- Step 2: $P(z) = \sum_x P(z \mid x)P(x) = \sum_x L(x \mid z)$
  (Law of total probability)
- Step 3: $P(x \mid z) = L(x \mid z)/P(z)$

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)

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

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)

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

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

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 $p(x \mid z_1, z_2, \dots)$?
- Bayes formula gives us:

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

Recursive Bayesian Updates

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

Recursive Bayesian Updates

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

- **Markov Assumption:**

  z_n is independent of $z_1, \dots, z_{n-1}$ if we know x

Recursive Bayesian Updates

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

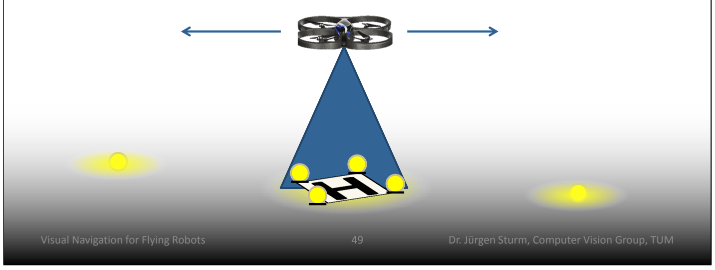
- **Markov Assumption:**

  z_n is independent of $z_1, \dots, z_{n-1}$ if we know x

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

Example: Sensor Measurement

- Quadrocopter seeks the landing zone
- Landing zone is marked with many bright lamps **and a visual marker**



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?

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

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?

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

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

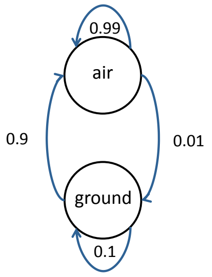
- To incorporate the outcome of an action $\mathbf{u}$ into the current state estimate ("belief"), we use the conditional pdf

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

- 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

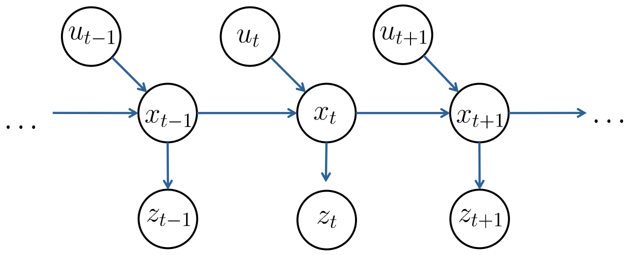
- Prior belief on robot state: $P(x = \text{ground}) = 1.0$ (robot is located on the ground)
- Robot executes “take-off” action
- What is the robot’s belief after one time step?

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

- 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:
    $$\mathbf{d}_t = (u_1, z_1, \dots, u_t, z_t)^\top$$
  - 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**

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

Bayes Filter

For each time step, do

1. Apply motion model

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

2. Apply sensor model

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

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

Bayes Filter

For each time step, do

1. Apply motion model

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

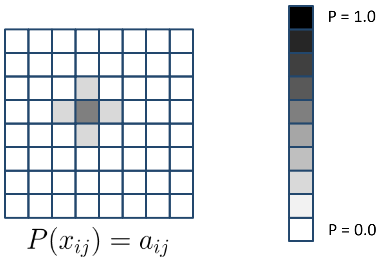
2. Apply sensor model

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

Second note: Bayes filter also works when actions and observations are asynchronous!

Example: Localization

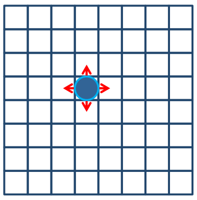
- Discrete state $x \in \{1, 2, \dots, w\} \times \{1, 2, \dots, h\}$
- Belief distribution can be represented as a grid
- This is also called a **histogram filter**



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

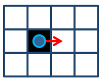
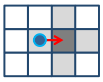
Example: Localization

- 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

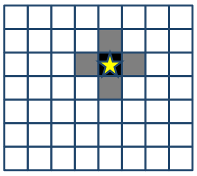
- Action $u \in \{\text{north}, \text{east}, \text{south}, \text{west}\}$
- Robot can move one cell in each time step
- Actions are not perfectly executed
- Example: move east

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

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

Example: Localization

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

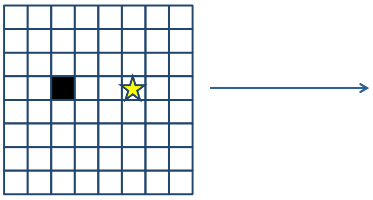


Example: Localization

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

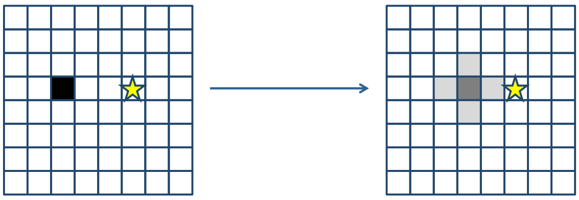
Example: Localization

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



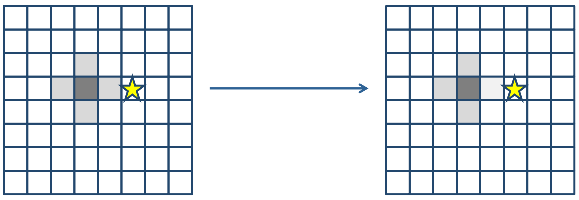
Example: Localization

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



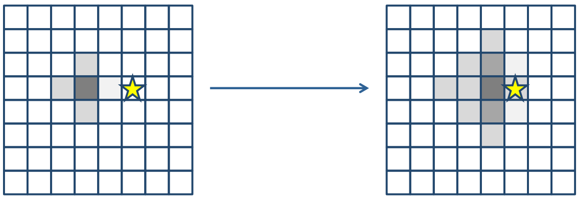
Example: Localization

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



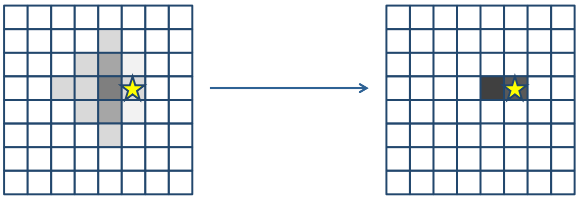
Example: Localization

- 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
- Question: Where is the robot?



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

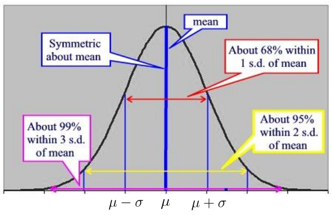
- Bayes filter with continuous states
- State represented with a normal distribution
- Developed in the late 1950's
- Kalman filter is very efficient (only requires a few matrix operations per time step)
- Applications range from economics, weather forecasting, satellite navigation to robotics and many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

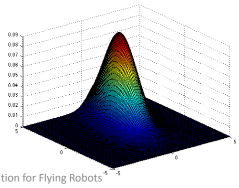
$$X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

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

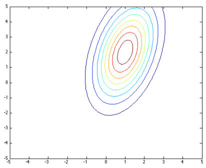
$$= \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

- Example: 2-dimensional normal distribution

pdf



iso lines



Properties of Normal Distributions

- Linear transformation → remains Gaussian

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

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

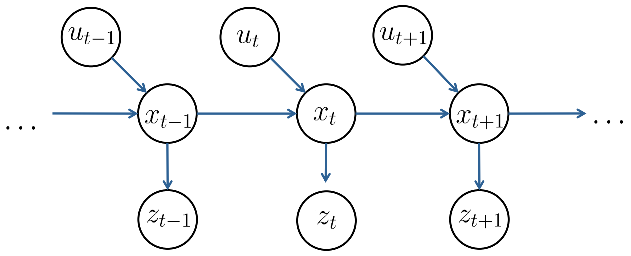
- Intersection of two Gaussians → remains Gaussian

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

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

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

$$x_t = A x_{t-1}$$

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

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

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

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

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

Linear Observations

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

$$z_t = C x_t$$

Linear Observations

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

$$z_t = C x_t + \delta_t$$

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

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

- Next state is also Gaussian (linear transformation)

$$x_t \sim \mathcal{N}(Ax_{t-1} + Bu_t, Q)$$

- Observations are also Gaussian

$$z_t \sim \mathcal{N}(Cx_t, R)$$

From Bayes Filter to Kalman Filter

For each time step, do

1. Apply motion model

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

From Bayes Filter to Kalman Filter

For each time step, do

1. Apply motion model

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

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

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

From Bayes Filter to Kalman Filter

For each time step, do

2. Apply sensor model

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

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

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

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

Kalman Filter

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

For each time step, do

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)
- Fixed time intervals Δt

Example

- Motion function

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

- Derivative of motion function

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

Example

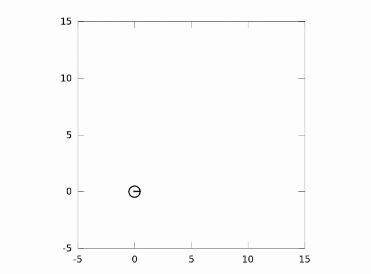
- Observation Function (→ Sheet 2)

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

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

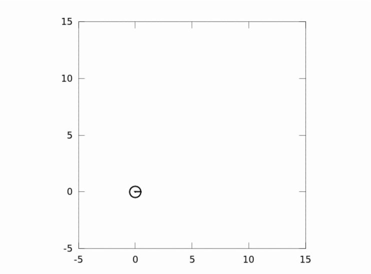
Example

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



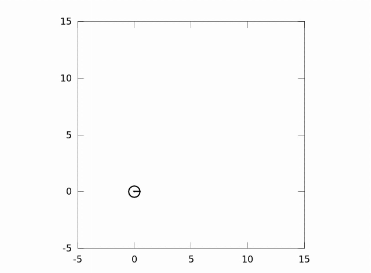
Example

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



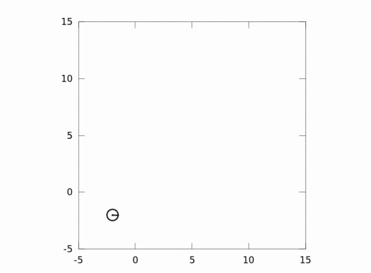
Example

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



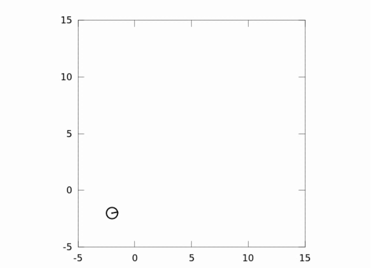
Example

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



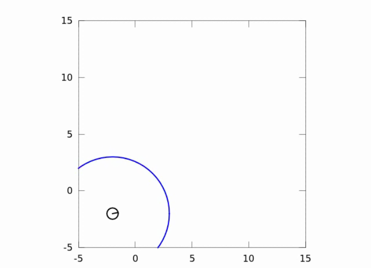
Example

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

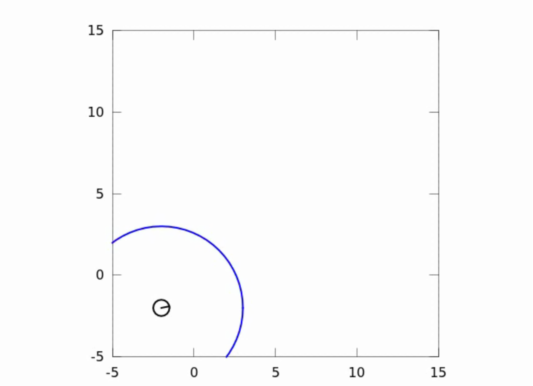


Example

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



Example



Lessons Learned Today

- 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

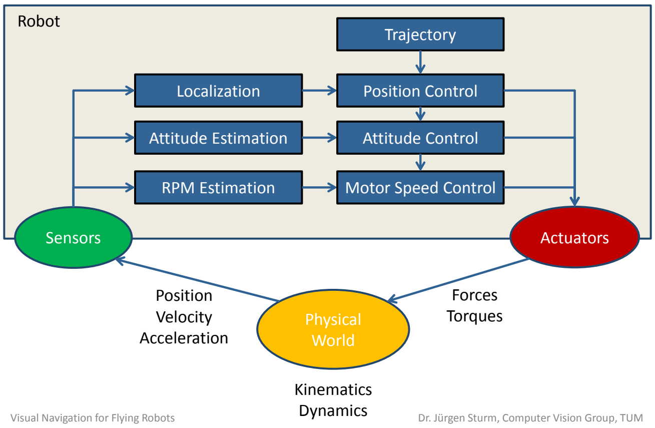
Technische Universität München

Visual Navigation for Flying Robots

Robot Control

Dr. Jürgen Sturm

Control Architecture



Robot

Trajectory

Localization

Position Control

Attitude Estimation

Attitude Control

RPM Estimation

Motor Speed Control

Sensors

Actuators

Physical World

Position
Velocity
Acceleration

Forces
Torques

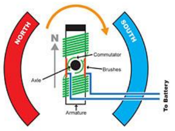
Kinematics
Dynamics

Agenda for Today

- Motors
- Motor Controllers
- Kinematics and Dynamics
- Linear Control

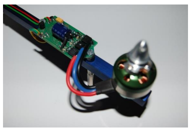
DC Motors

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

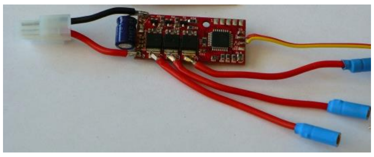


Brushless Motors

- Used in most quadrocopters
- Permanent magnets on the axis
- Electromagnets on the outside

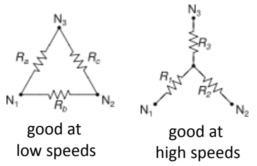


→ Does not require brushes (less maintenance)

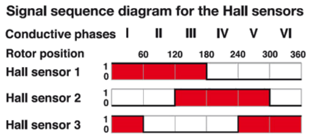


Brushless Motors

- Winding styles



- Hall sensor or EMF to detect rotation

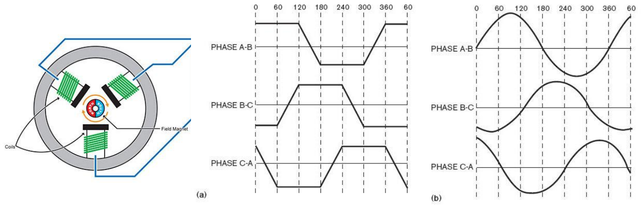


Signal sequence diagram for the Hall sensors

Conductive phases	I	II	III	IV	V	VI
Rotor position	60	120	180	240	300	360
Hall sensor 1	1	1	1	0	0	0
Hall sensor 2	0	0	1	1	1	0
Hall sensor 3	1	0	0	0	1	1

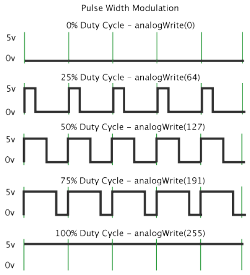
Motor Controller

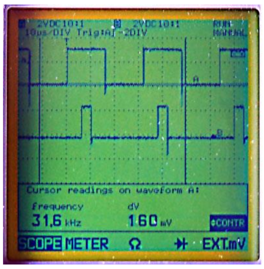
- Micro controller estimates rotation and generates PWM signal
- AC signal generator (inverter) generates motor phases



Pulse Width Modulation (PWM)

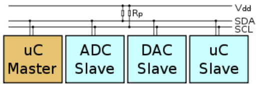
- Protocol used to control motor speed
- Remote controls typically output PWM

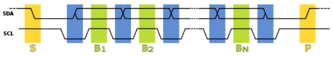




I2C Protocol

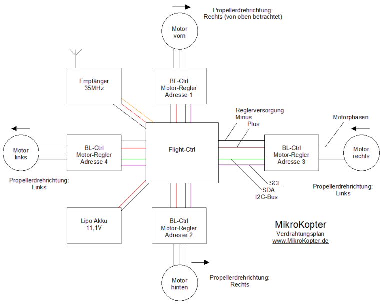
- Serial data line (SDA) + serial clock line (SCL)
- All devices connected in parallel
- 7-10 bit address, 100-3400 kbit/s speed
- Used by Mikrocopter for motor control

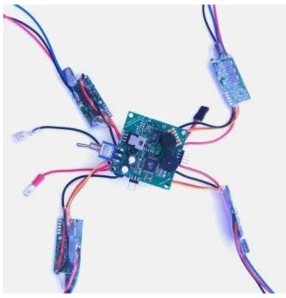




Attitude + Motor Controller Boards

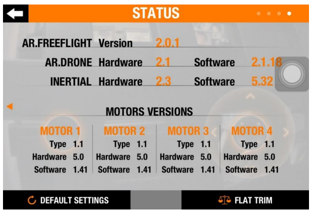
- Example: Mikrokopter Platform





Attitude + Motor Controller Boards

- Example: Ardrone Platform



STATUS

AR.FREEFLIGHT Version 2.0.1

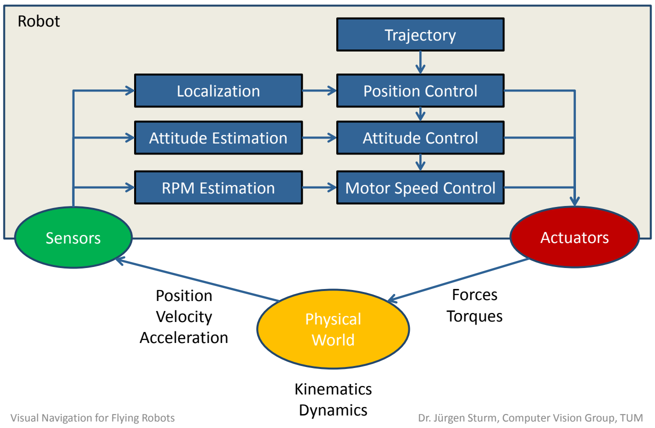
AR.DRONE Hardware 2.1 Software 2.1.18

INERTIAL Hardware 2.3 Software 5.32

MOTORS VERSIONS

MOTOR 1	MOTOR 2	MOTOR 3	MOTOR 4
Type 1.1	Type 1.1	Type 1.1	Type 1.1
Hardware 5.0	Hardware 5.0	Hardware 5.0	Hardware 5.0
Software 1.41	Software 1.41	Software 1.41	Software 1.41

Control Architecture



Robot

Trajectory

Localization

Position Control

Attitude Estimation

Attitude Control

RPM Estimation

Motor Speed Control

Sensors

Actuators

Position
Velocity
Acceleration

Physical World

Forces
Torques

Kinematics
Dynamics

Kinematics and Dynamics

- Kinematics
  - Describes the motion of rigid bodies
  - Position, velocity, acceleration
- 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 $u \in \mathbb{R}$
- Linear process model

$$\mathbf{x}_t = \begin{pmatrix} 1 & \Delta t & 0 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{pmatrix} \mathbf{x}_{t-1} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} u_t$$

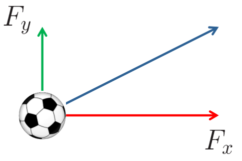
- Kalman filter
- How many states do we need for 3D?

Dynamics - Essential Equations

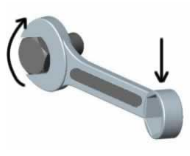
- Force (Kraft)

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

linear acceleration



- Torque (Drehmoment)



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

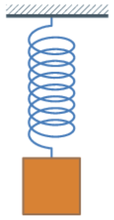
angular acceleration

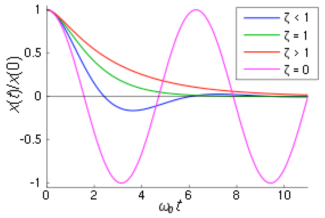
Forces

- Gravity $F_{\text{grav}} = mg$
- Friction
  - Stiction (static friction) $F_{\text{stiction}} = c_s \text{sign } \dot{x}$
  - Damping (viscous friction) $F_{\text{damping}} = D\dot{x}$
  - Air drag $F_{\text{airdrag}} = c_W A \frac{1}{2} \rho \dot{x}$
- Spring $F_{\text{spring}} = K(x - x_{\text{eq}})$
- Magnetic force
- …

Example: Spring-Damper System

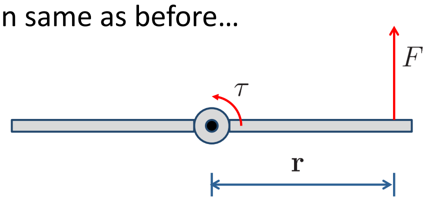
- Combination of spring and damper
- Forces $F = F_{\text{damping}} + F_{\text{spring}}$
- Resulting dynamics $m\ddot{x} = D\dot{x} + K(x - x_{\text{eq}})$





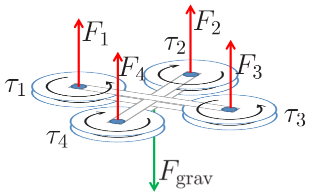
Torques

- 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…



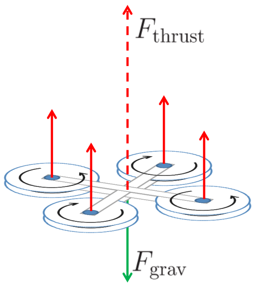
Dynamics of a Quadrocopter

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



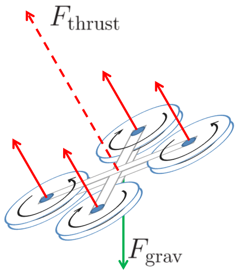
Vertical Acceleration

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



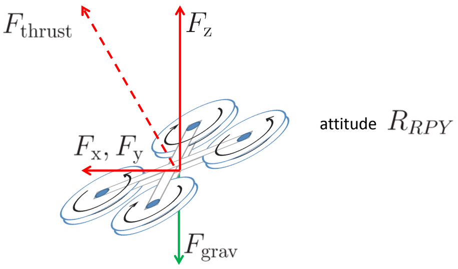
Vertical and Horizontal Acceleration

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



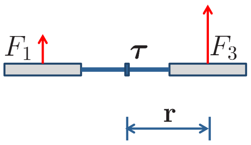
Vertical and Horizontal Acceleration

- 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)

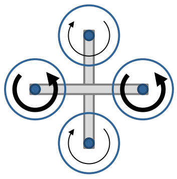
- Attitude changes when opposite motors generate unequal thrust
- Induced torque $\boldsymbol{\tau} = (F_1 - F_3) \times \mathbf{r}$
- Induced angular acceleration $\boldsymbol{\alpha} = J^{-1}\boldsymbol{\tau}$



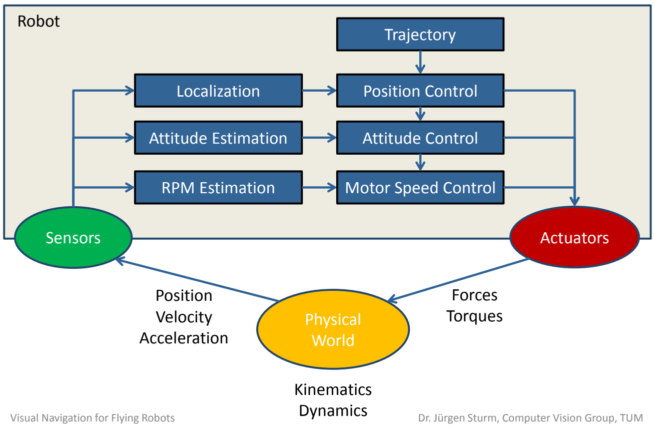
Side view of quadrocopter

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



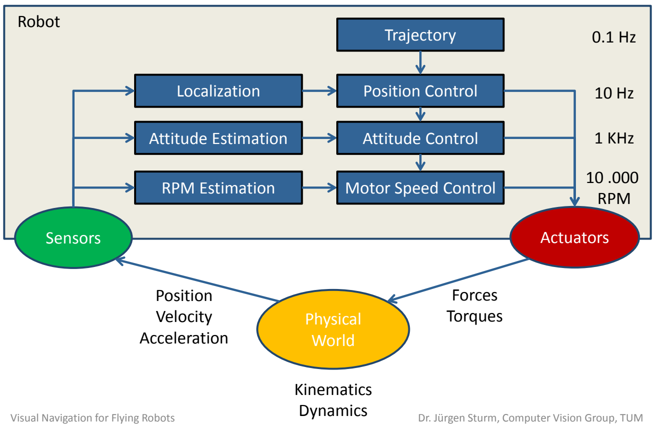
Assumptions of Cascaded Control

- 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)

Cascaded Control


Robot

Trajectory

0.1 Hz

Localization

Position Control

10 Hz

Attitude Estimation

Attitude Control

1 KHz

RPM Estimation

Motor Speed Control

10 .000 RPM

Sensors

Actuators

Position Velocity Acceleration

Physical World

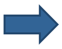
Forces Torques

Kinematics Dynamics

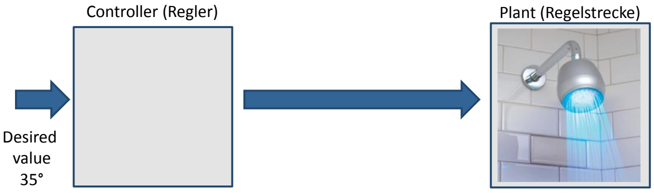
Feedback Control

- Given:
  - Goal state x_{des}
  - Measured state (feedback) z
- Wanted:
  - Control signal u to reach goal state
- How to compute the control signal?

Feedback Control - Generic Idea


Desired value 35°

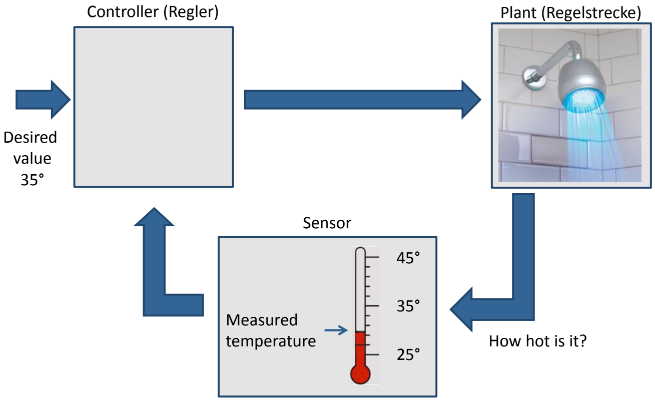
Feedback Control - Generic Idea


Controller (Regler)

Plant (Regelstrecke)

Desired value 35°

Feedback Control - Generic Idea


Controller (Regler)

Plant (Regelstrecke)

Desired value 35°

Sensor

Measured temperature

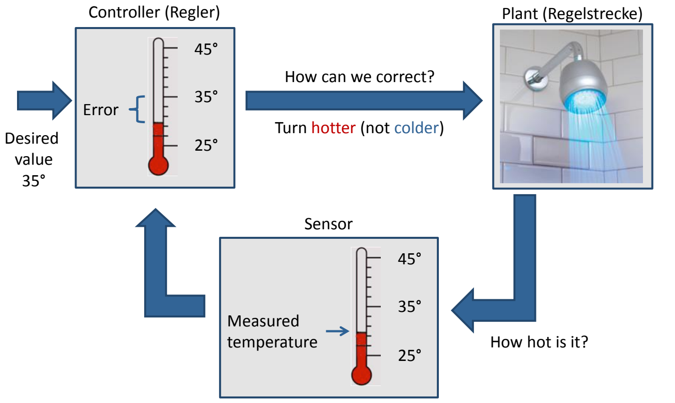
45°

35°

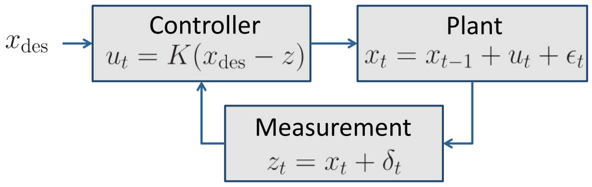
25°

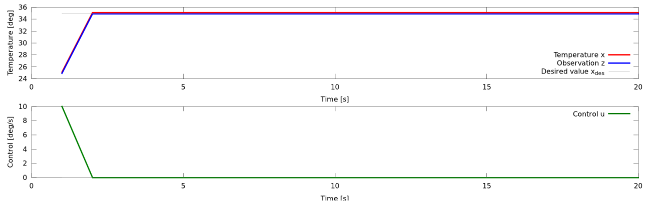
How hot is it?

Feedback Control - Generic Idea



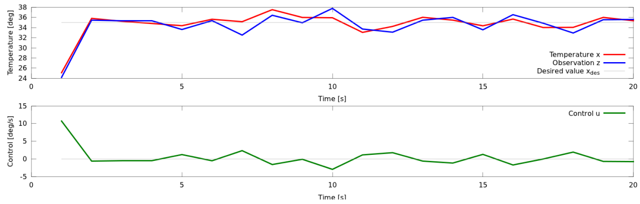
Feedback Control - Example





Measurement Noise

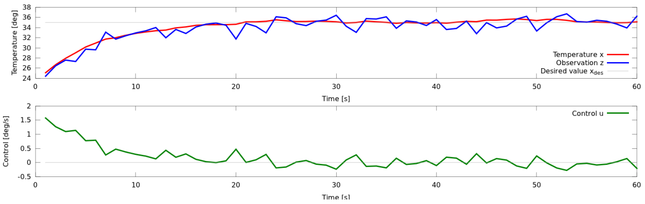
- What effect has noise in the measurements?



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

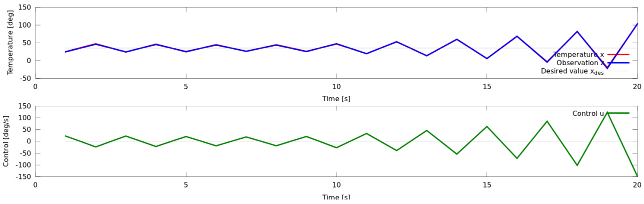
Proper Control with Measurement Noise

- Lower the gain… (K=0.15)



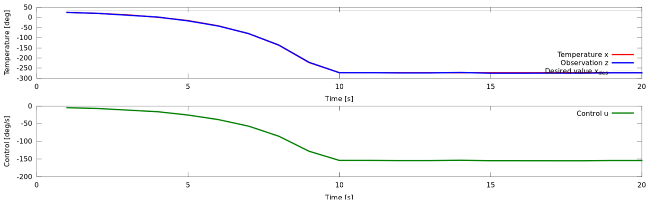
What do High Gains do?

- High gains are always problematic (K=2.15)



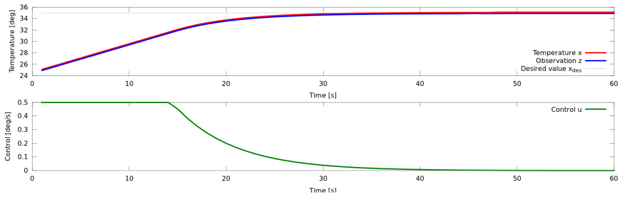
What happens if sign is messed up?

- Check K=-0.5

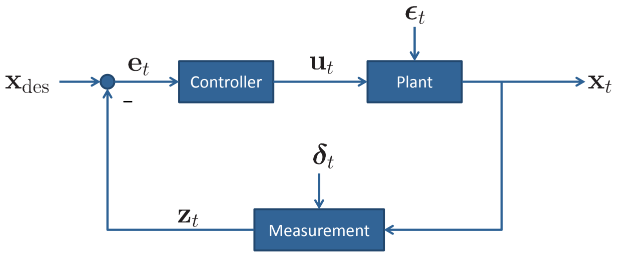


Saturation

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

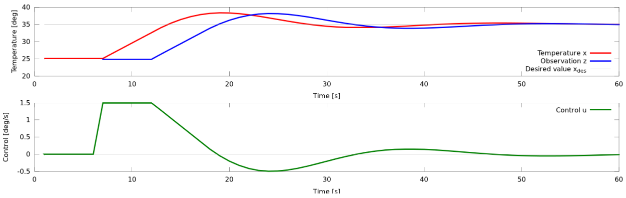


Block Diagram



Delays

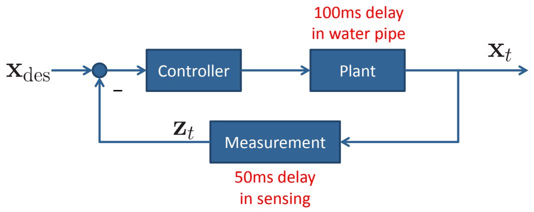
- In practice most systems have delays
- Can lead to overshoots/oscillations/de-stabilization



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

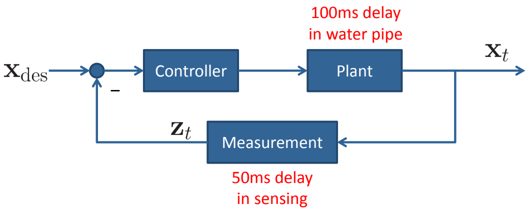
Delays

- What is the total dead time of this system?



Delays

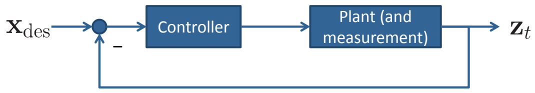
- What is the total dead time of this system?



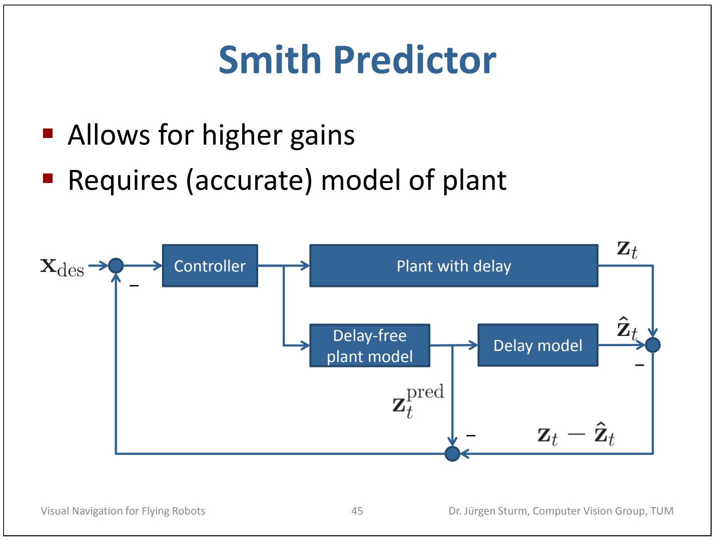
- Can we distinguish delays in the measurement from delays in actuation?

Delays

- What is the total dead time of this system?

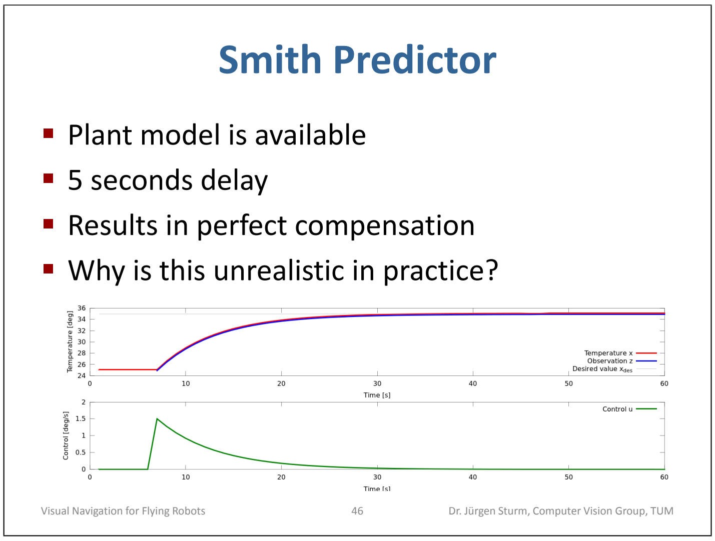


- Can we distinguish delays in the measurement from delays in actuation? No!



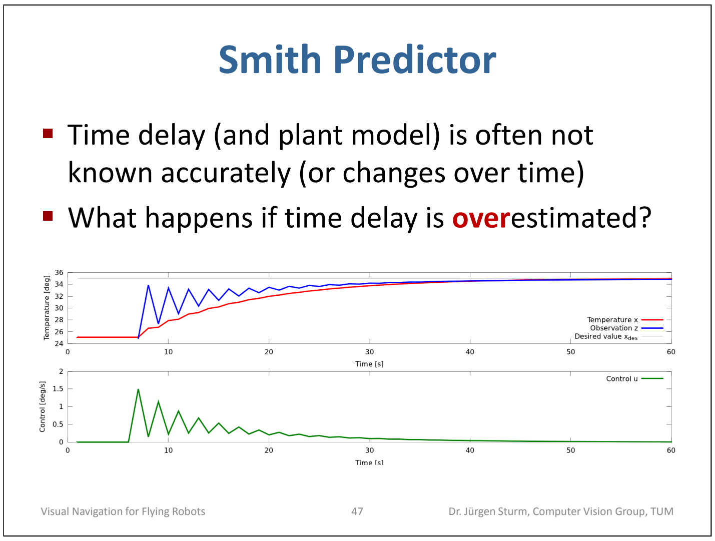
Smith Predictor

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



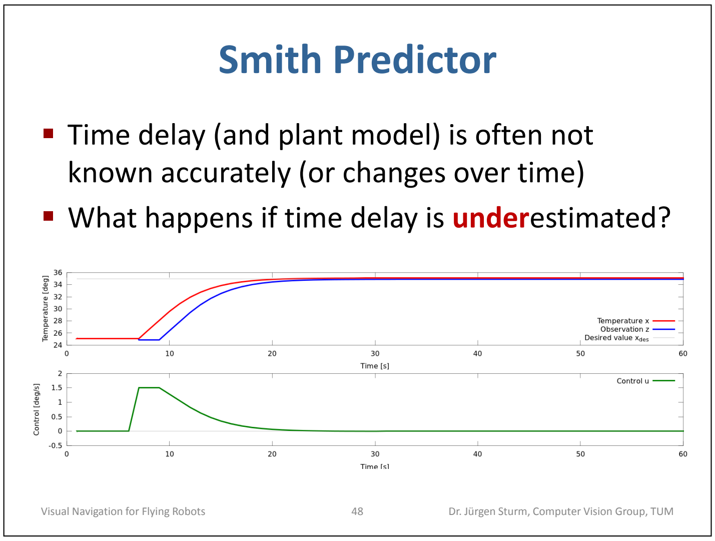
Smith Predictor

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



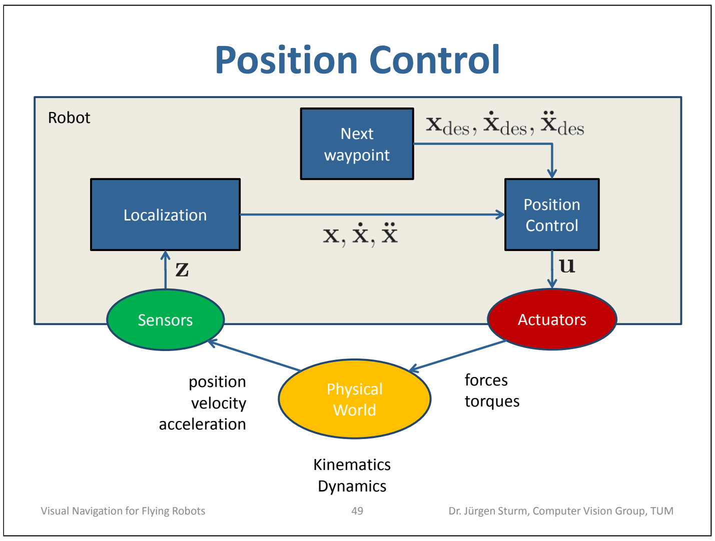
Smith Predictor

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

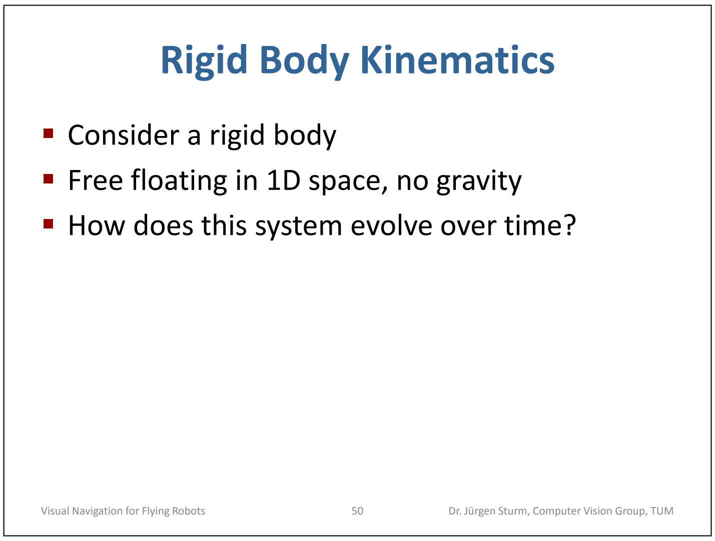


Smith Predictor

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



Position Control

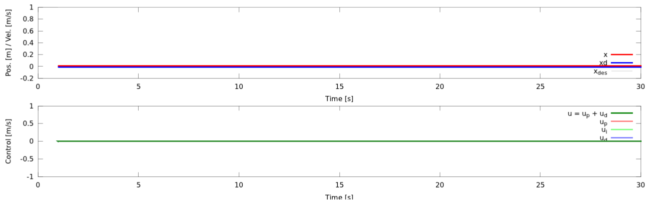


Rigid Body Kinematics

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

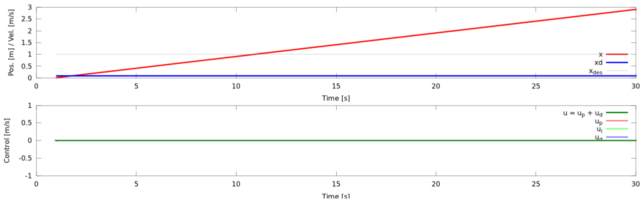
Rigid Body Kinematics

- 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?

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

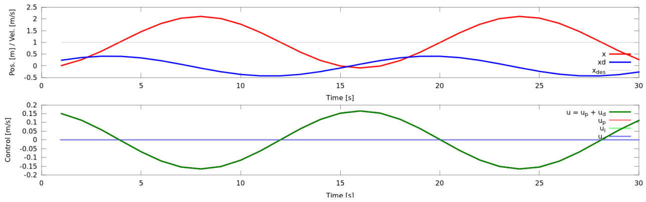
- This is called proportional control

P Control

- What happens for this control law?

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

- This is called proportional control

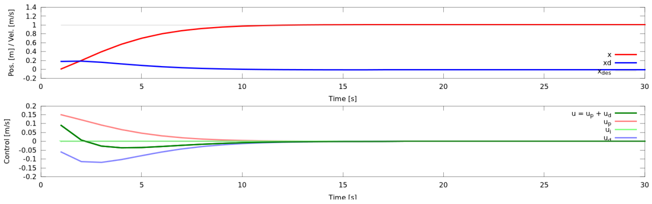


PD Control

- 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

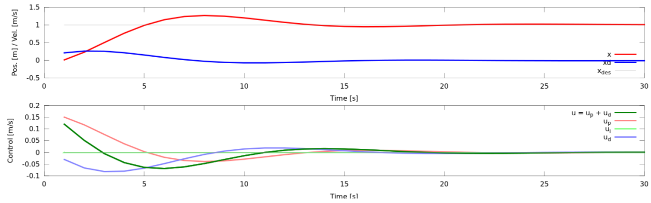


PD Control

- 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?

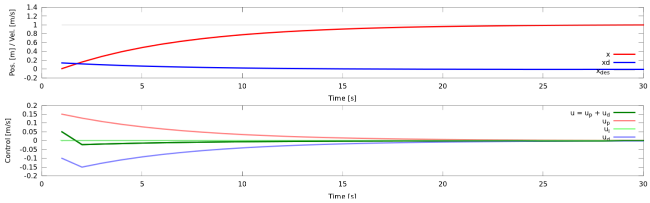


PD Control

- What happens for this control law?

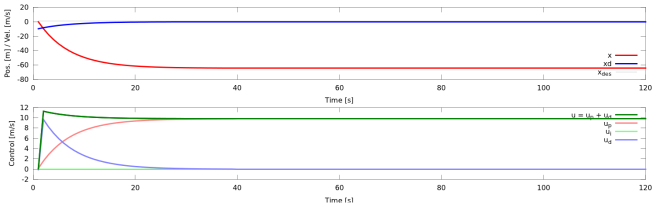
$$u_t = K_P(x_{\text{des}} - x_{t-1}) + K_D(\dot{x}_{\text{des}} - \dot{x}_{t-1})$$

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



PD Control

- What happens when we add gravity?

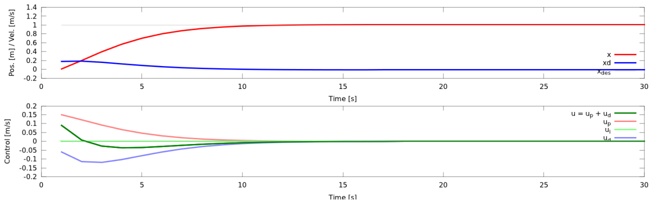


Gravity compensation

- Add as an additional term in the control law

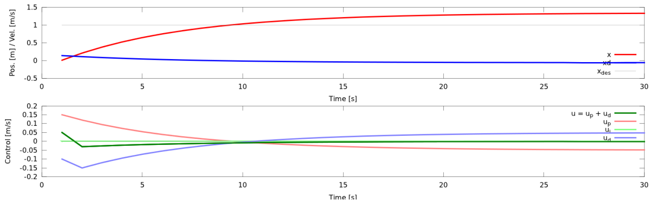
$$u_t = K_P(x_{\text{des}} - x_{t-1}) + K_D(\dot{x}_{\text{des}} - \dot{x}_{t-1}) + F_{\text{grav}}$$

- Any known (inverse) dynamics can be included



PD Control

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

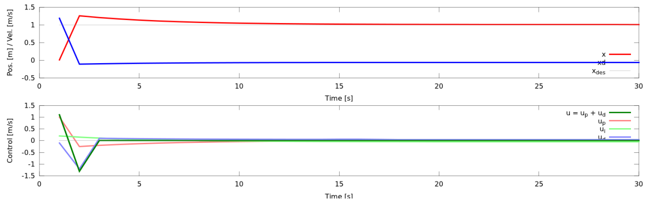


PID Control

- 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

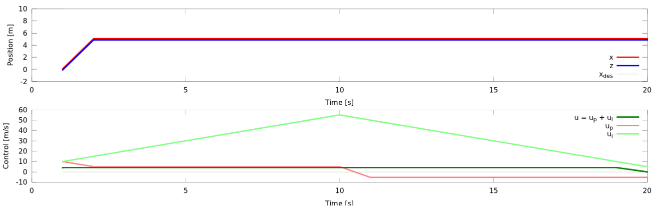
- Idea: Estimate the system error (bias) by integrating the error

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

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

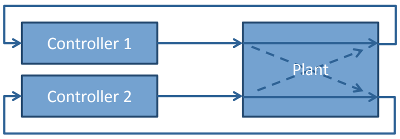
Example: Wind-up effect

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



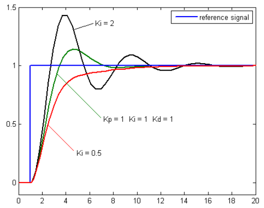
De-coupled Control

- 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?

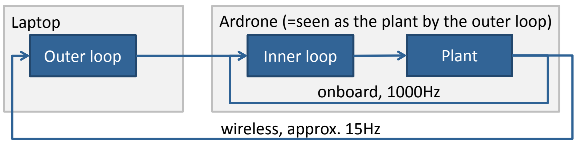
- Gains too large: overshooting, oscillations
- Gains too small: long time to converge
- Heuristic methods exist
- In practice, often tuned manually



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

$$\mathbf{x}_{\text{inner}} = \begin{pmatrix}\underbrace{\omega_x \quad \omega_y \quad \omega_z}_{\substack{\text{attitude}\\ \text{(measured using gyro +}\\ \text{accelerometer)}}} & \underbrace{z}_{\substack{\text{altitude}\\ \text{(measured using ultrasonic}\\ \text{distance sensor + attitude)}}}\end{pmatrix}^\top$$

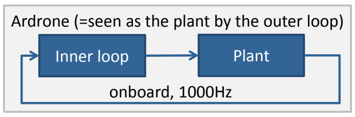
Ardrone: Inner Control Loop

- 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

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

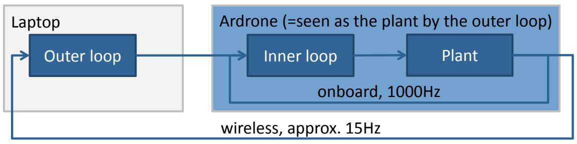


Ardrone: Outer Control Loop

- Outer loop sees inner loop as a plant (black box)
- Plant input: roll, pitch, yaw rate, z velocity

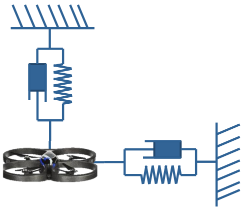
$$\mathbf{u}_{\text{outer}} = \begin{pmatrix}\omega_x & \omega_y & \dot{\omega}_z & \dot{z}\end{pmatrix}^\top$$

- 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



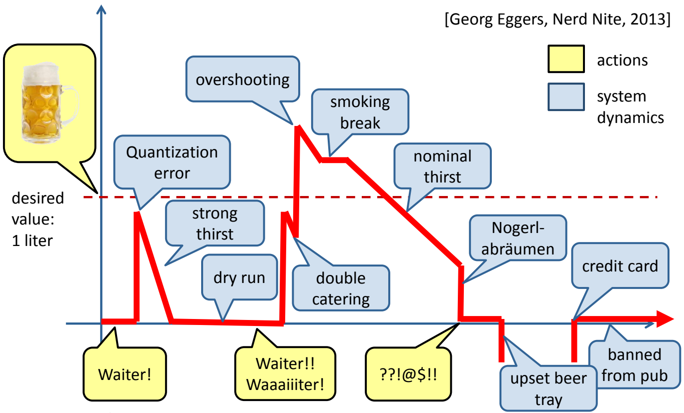
PID Control – Summary

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

PID Control – Beergarden Example

[Georg Eggers, Nerd Nite, 2013]



Advanced Control Techniques

What other control techniques do exist?

- Adaptive control
- Robust control
- Optimal control
- 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

$$J = \sum_{k=0}^{\infty} \left(x_k^T Q x_k + u_k^T R u_k \right)$$

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

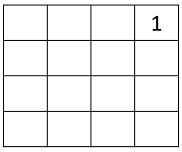
Linear Quadratic Regulator

- Advantage:
  Cost matrix has an intuitive interpretation
- Disadvantage:
  Typically no closed form solution
- Often solved numerically
- Only for small planning horizon

Reinforcement Learning

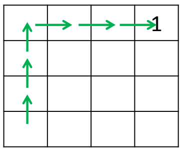
- Note that in principle, any cost function can be used
- Sometimes, it is easier to specify a reward function $r(x_t, u_t)$
- Example:

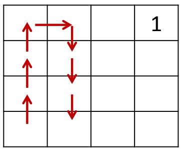
$$r(x_t, u_t) = \begin{cases} 1 & \text{if } x_{\text{des}} = x_t \\ 0 & otherwise \end{cases}$$



Reinforcement Learning

- Reward function $r(x_t, u_t)$
- Episode (=trajectory) $\tau = (x_1, u_1, \dots, x_n, u_n)$
- Reward of an episode $R(\tau) = \sum_t r(x_t, u_t)$

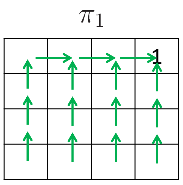


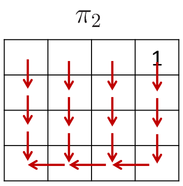


Reinforcement Learning

- A policy (=controller) defines which action to take in a particular state

$$\pi(x) = u$$





Policy Evaluation

- The optimal policy maximizes the expected future reward
- How can we estimate the expected future reward?
- How can we find the optimal policy?
- Closer look at two methods
  - Q learning
  - Policy gradient methods

Reinforcement Learning

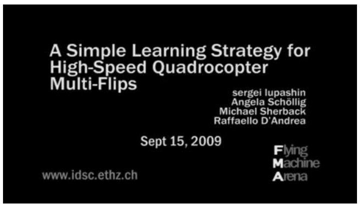
Q learning

- Learn the value function for each state-action pair
- Needs compact representation of value function (e.g., neural networks)
- Examples: TD-Gammon (mid-90’s), Inverted pendulum, …

Reinforcement Learning

Policy gradient methods

- Policy is parameterized
- Analytic gradient typically not available
- Simulation-based optimization



A Simple Learning Strategy for High-Speed Quadrocopter Multi-Flips

sergei lupashin
Angela Schöllig
Michael Sherback
Raffaello D'Andrea

Sept 15, 2009

www.idsc.ethz.ch

Flying Machine Arena

[Lupashin et al., ICRA 2010]

Reinforcement Learning

Policy gradient methods

- Policy is parameterized
- Analytic gradient typically not available
- Simulation-based optimization

Learning to follow a trajectory

Quadrocopters improve over time

IDSC
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

[Schoellig et al., ACC 2012]

Inverse Reinforcement Learning

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

Lessons Learned Today

- Brushless Motors
- Motor Controllers
- Cascaded Control
- PID Control
- Advanced Control Techniques

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

Visual Motion Estimation

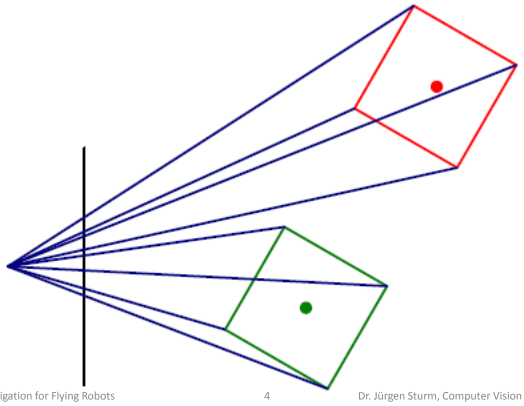
Dr. Jürgen Sturm

Motivation

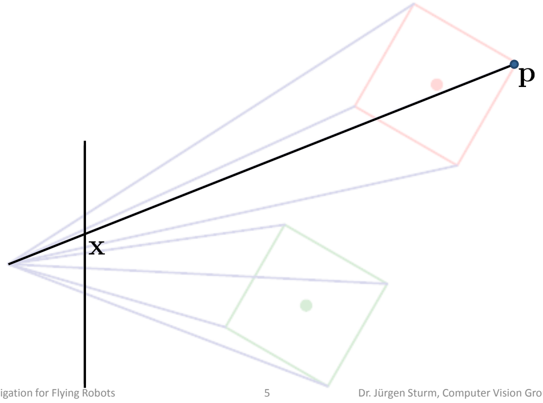
Visual Motion Estimation

- Quick geometry recap
- Image filters
- 2D image alignment
- Corner detectors
- Kanade-Lucas-Tomasi tracker
- 2D motion estimation
- Interesting research papers from ICRA and ROSCon

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

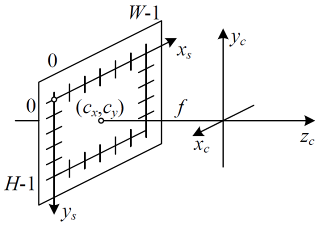
$$\tilde{\mathbf{x}} = \lambda \bar{\mathbf{x}} = \mathbf{p}$$

- 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 (located in 1m distance from origin)
- 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 $\mathbf{x} \in [0, W-1] \times [0, H-1] \subset \mathbb{N}_0^2$ (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

$$f : \Omega \mapsto \mathbb{R}^3$$

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

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)

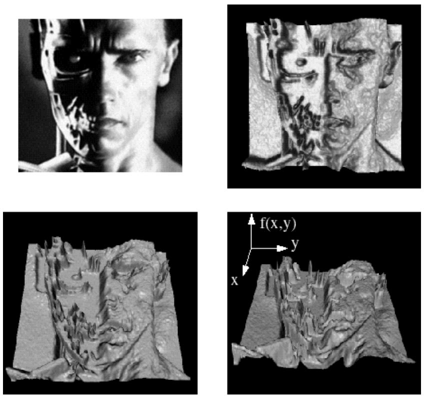
often (column,row)

j →

i ↓

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)

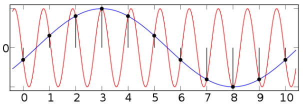
$$V = B^{\frac{1}{\gamma}} \text{ with } \gamma = 2.2$$

- 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

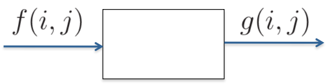


Rolling Shutter

- 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



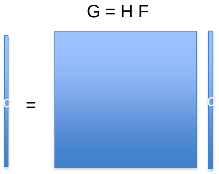
- Example tasks:
de-noising, (de-)blurring, computing derivatives, edge detection, …

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

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

$=$

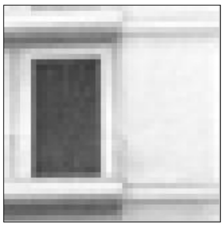
?	?	?	?	?	?	?	?
?	-5	9	-9	21	-12	10	?
?	-29	18	24	4	-7	5	?
?	-50	40	142	-88	-34	10	?
?	-41	41	264	-175	-71	0	?
?	-24	37	349	-224	-120	-10	?
?	-23	33	360	-217	-134	-23	?
?	?	?	?	?	?	?	?

Important Filters

- **Impulses**
- **Shifts**
- Blurring and de-blurring
  - **Gaussian**
  - Bilateral filter
  - Motion blur
- Edges
  - **Finite difference filter**
  - Derivative filter
  - Oriented filters
  - Gabor filter
- …

Impulse

$$g(i,j) = f \underset{\substack{\uparrow \\ \text{convolution} \\ \text{operator}}}{*} h = \sum_{k,l} h(i-k,j-l)f(k,l)$$



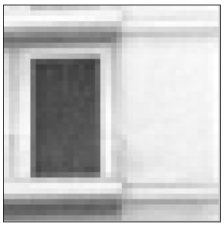
$f(i,j)$

$*$

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

$h(i,j)$

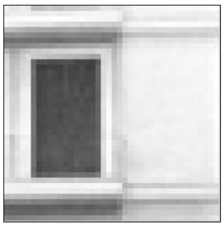
$=$



$g(i,j)$

Image shift (translation)

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



$f(i,j)$

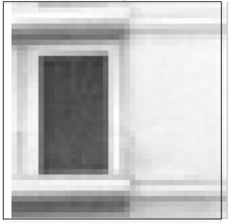
$*$

0	0	0	0	0
0	0	0	0	0
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0

$h(i,j)$

$=$

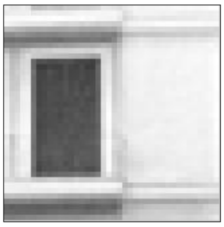
2 pixels →



$g(i,j)$

Image rotation

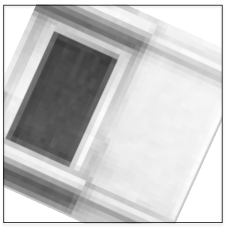
$$g(i,j) = f * h = \sum_{k,l} h(i-k,j-l)f(k,l)$$



$f(i,j)$

$*$? $=$

$h(i,j)$

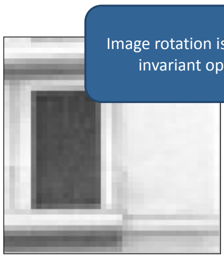


$g(i,j)$

Image rotation

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

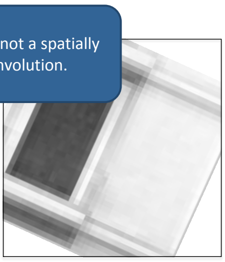
Image rotation is a linear operator (why?), but not a spatially invariant operation (why?). There is no convolution.



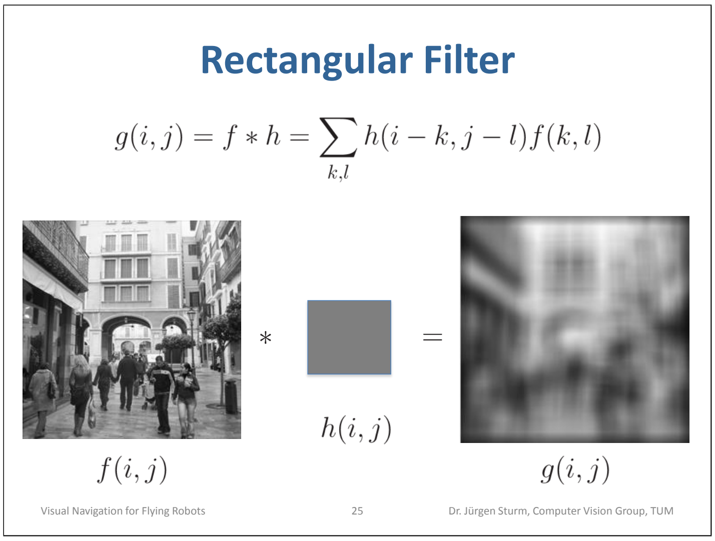
$f(i,j)$

$*$? $=$

$h(i,j)$



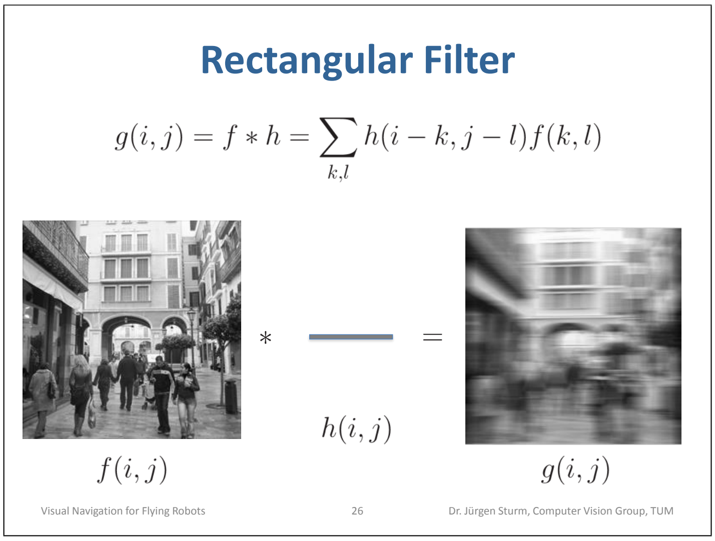
$g(i,j)$



Rectangular Filter

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

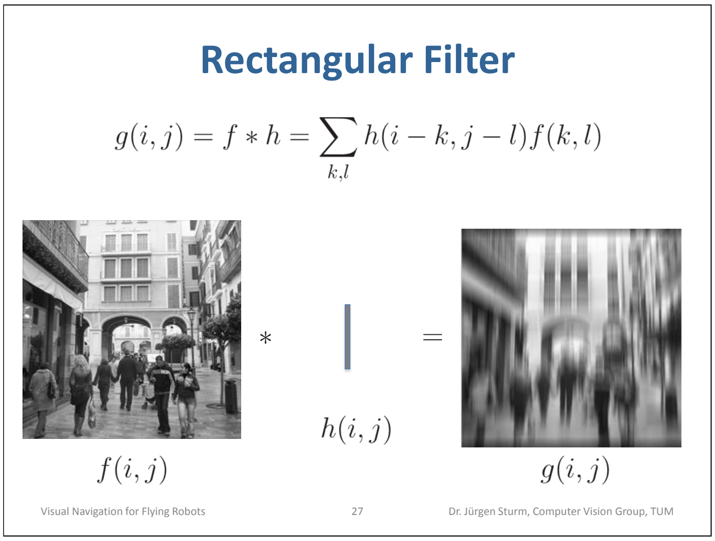
$f(i,j)$ $*$ $h(i,j)$ $=$ $g(i,j)$



Rectangular Filter

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

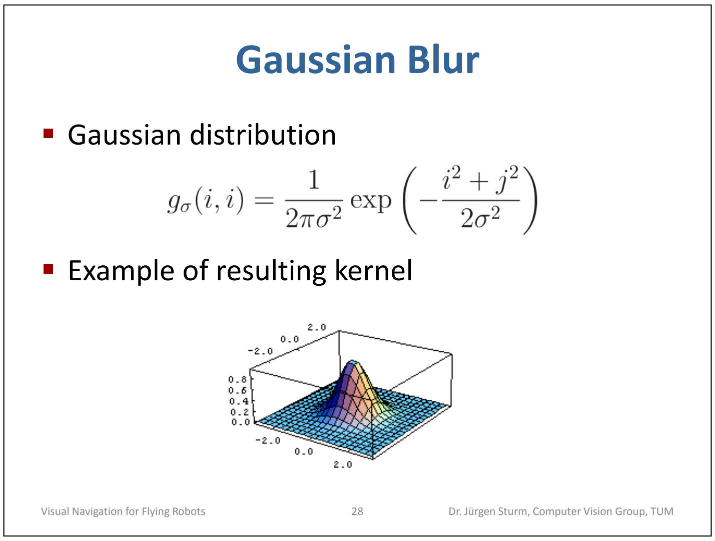
$f(i,j)$ $*$ $h(i,j)$ $=$ $g(i,j)$



Rectangular Filter

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

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

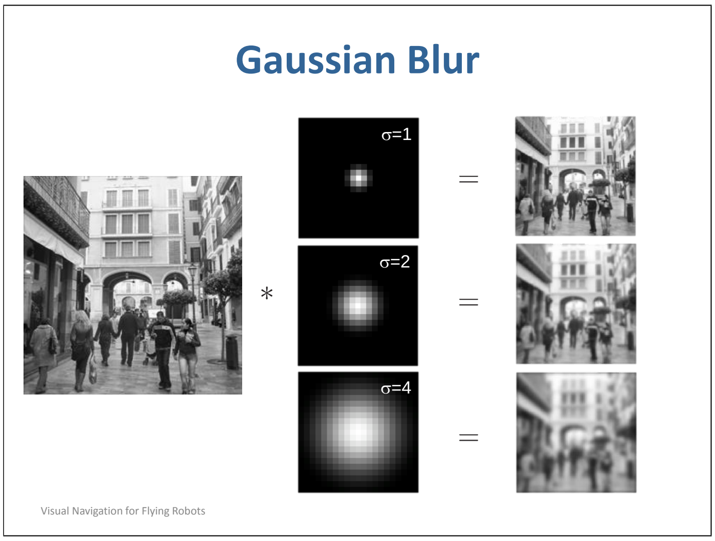


Gaussian Blur

- 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

$*$ $=$ $=$ $=$

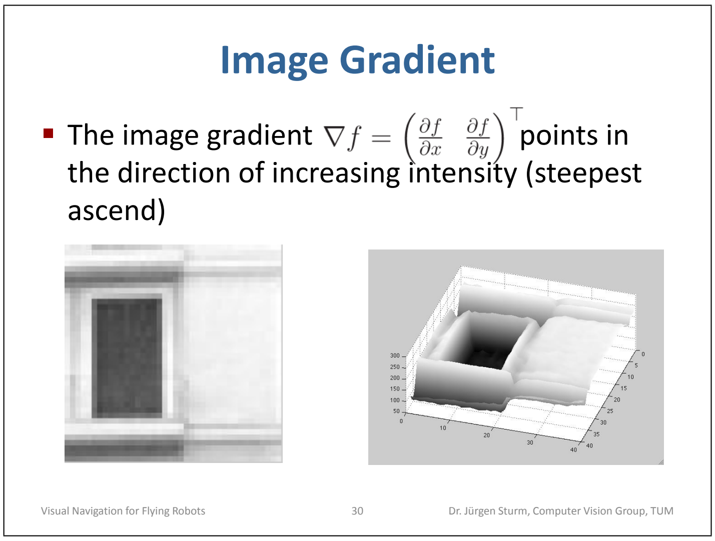
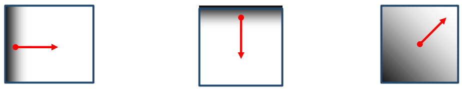


Image Gradient

- 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 $\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{pmatrix}^\top$ points in the direction of increasing intensity (steepest ascend)



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

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)

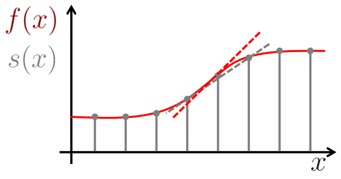
Finite difference

- 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



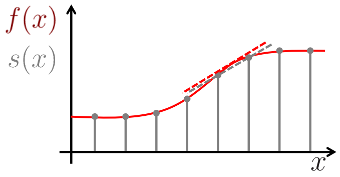
Finite difference

- 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



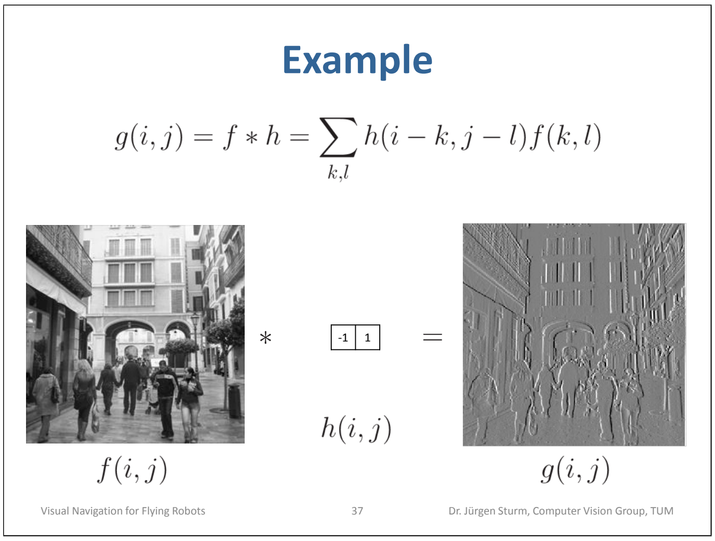
Second-order Derivative

- Differentiate again to get second-order central difference

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

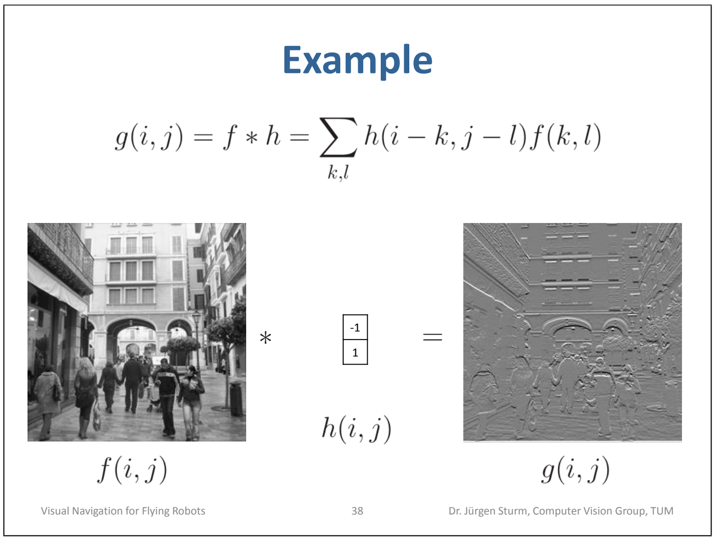
Corresponding convolution kernel:

1	-2	1



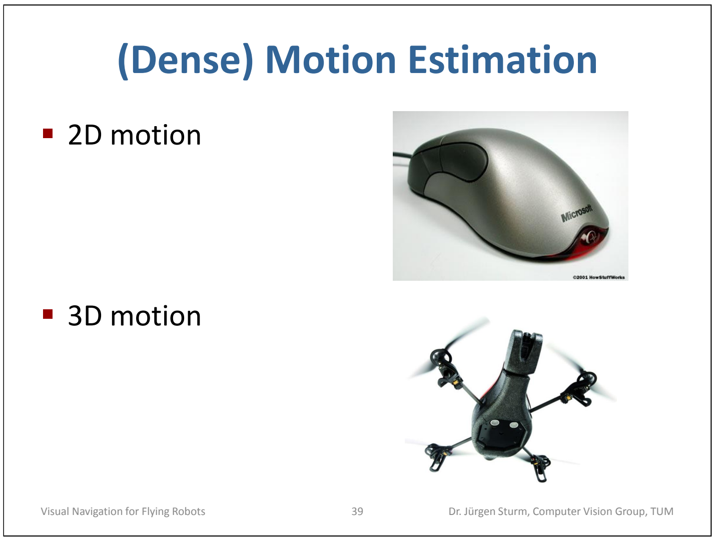
Example

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



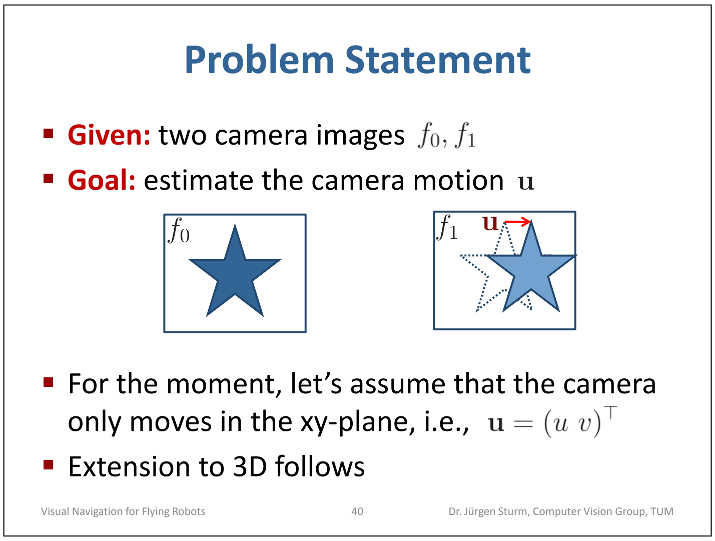
Example

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



(Dense) Motion Estimation

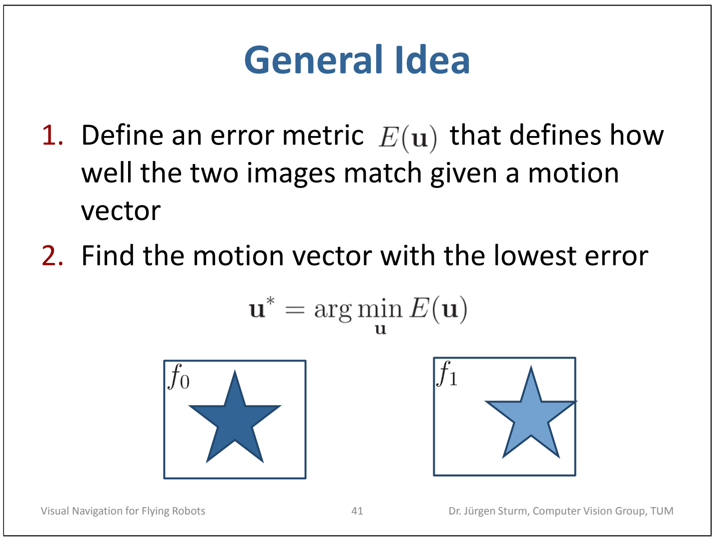
- 2D motion
- 3D motion



Problem Statement

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

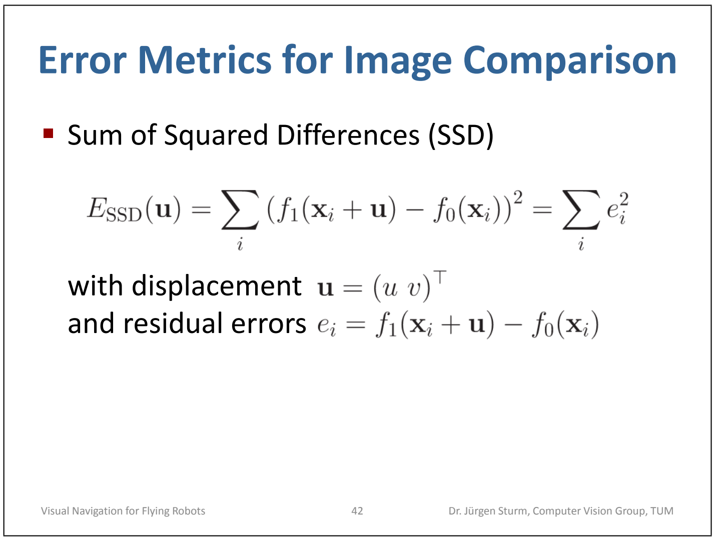
- For the moment, let's assume that the camera only moves in the xy-plane, i.e., $\mathbf{u} = (u\ v)^\top$
- Extension to 3D follows



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)

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

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 (SAD, L1 norm)

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

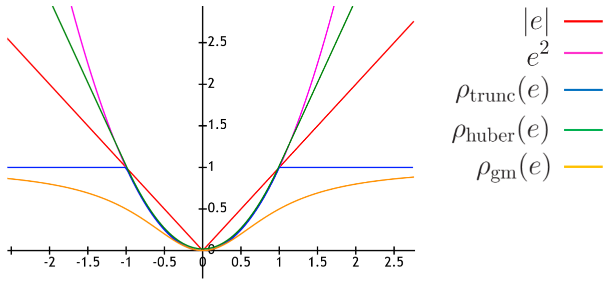
- Sum of truncated errors

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

- Geman-McClure

$$\rho_{\mathrm{gm}}(e) = \frac{x^2}{1 + x^2}$$

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

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

Cross-Correlation

- Maximize the product (instead of minimizing the differences)

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

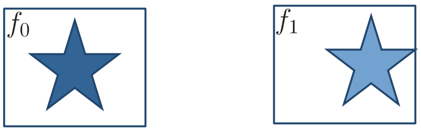
- Normalized cross-correlation (between -1..1)

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

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**

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



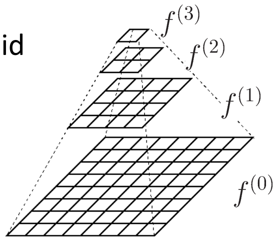
Finding the minimum

- 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

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

Motion Estimation

- Perform Gauss-Newton minimization on the SSD energy function (Lucas and Kanade, 1981)
- Gauss-Newton minimization
  - Linearize residuals w.r.t. to camera motion
  - Yields quadratic cost function
  - Build normal equations and solve linear system

Motion Estimation

- Taylor expansion of energy function

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

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

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

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 Minimization

- 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

$$\frac{\partial E(\mathbf{u}+\Delta\mathbf{u})}{\partial \Delta\mathbf{u}} = 2A\Delta\mathbf{u} + 2\mathbf{b} \overset{!}{=} 0$$

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 Minimization

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

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

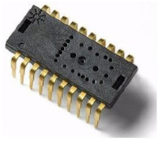
with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

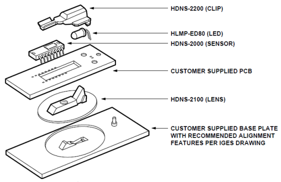
- … results in uncertainty in the motion estimate with covariance (e.g., useful for Kalman filter)

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

Optical Computer Mouse (since 1999)

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





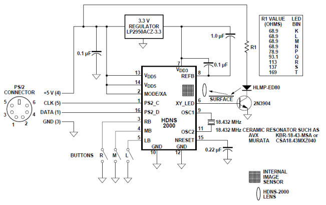
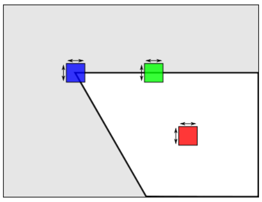


Image Patches

- 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?

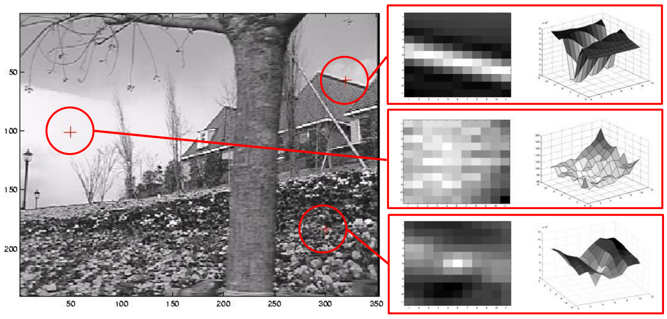
Image Patches

- 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



Corner Detection

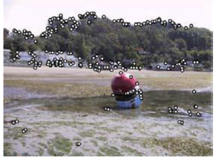
$$A = \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix}$$

- **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)

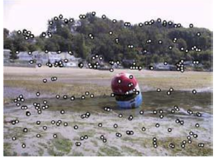
  $$\lambda_1 \lambda_2 > \kappa \left(\lambda_1 + \lambda_2\right)^2 \Leftrightarrow \det(A) > \kappa \operatorname{trace}^2(A)$$
- Shi-Tomasi (or Kanade-Lucas) $\min(\lambda_1, \lambda_2) > \kappa$

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

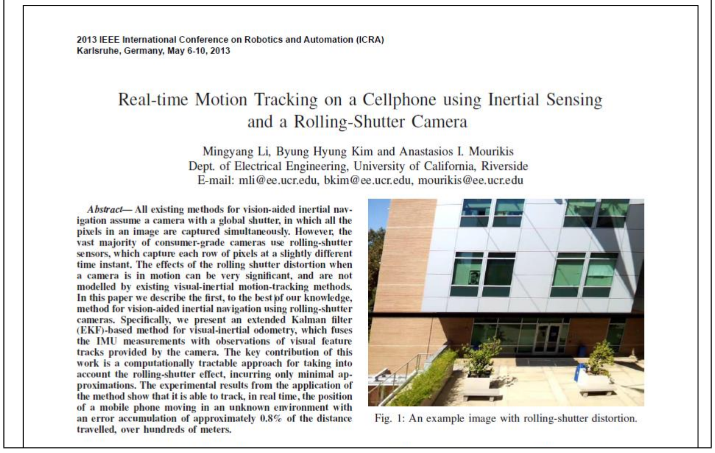
- Förstner detector (localize corner with sub-pixel accuracy)
- FAST corners (learn decision tree, minimize number of tests → super fast)
- Difference of Gaussians / DoG (scale-invariant detector)
- …

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

Interesting Papers from ICRA 2013



2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

Real-time Motion Tracking on a Cellphone using Inertial Sensing and a Rolling-Shutter Camera

Mingyang Li, Byung Hyung Kim and Anastasios I. Mourikis
Dept. of Electrical Engineering, University of California, Riverside
E-mail: mli@ee.ucr.edu, bkim@ee.ucr.edu, mourikis@ee.ucr.edu

Abstract— All existing methods for vision-aided inertial navigation assume a camera with a global shutter, in which all the pixels in an image are captured simultaneously. However, the vast majority of consumer-grade cameras use rolling-shutter sensors, which capture each row of pixels at a slightly different time instant. The effects of the rolling shutter distortion when a camera is in motion can be very significant, and are not modelled by existing visual-inertial motion-tracking methods. In this paper we describe the first, to the best of our knowledge, method for vision-aided inertial navigation using rolling-shutter cameras. Specifically, we present an extended Kalman filter (EKF)-based method for visual-inertial odometry, which fuses the IMU measurements with observations of visual feature tracks provided by the camera. The key contribution of this work is a computationally tractable approach for taking into account the rolling-shutter effect, incurring only minimal approximations. The experimental results from the application of the method show that it is able to track, in real time, the position of a mobile phone moving in an unknown environment with an error accumulation of approximately 0.8% of the distance travelled, over hundreds of meters.

Fig. 1: An example image with rolling-shutter distortion.

Smartphone as a Sensor Platform

[Li et al., ICRA '13]

- Quadcore
- Multiple cameras
- Accelerometer
- Gyroscopes
- GPS
- Wireless
- Relatively cheap

Challenges

[Li et al., ICRA '13]

- Rolling shutter camera
- Poor synchronization between camera and IMU

Kalman Filter

[Li et al., ICRA '13]

- Kalman state contains
  - IMU pose
  - last m camera poses

$$\mathbf{x}_k = [\underbrace{\mathbf{x}_{I_k}^T}_{\text{IMU state}} \quad \underbrace{\boldsymbol{\pi}_{C_{k-m}}^T \quad \cdots \quad \boldsymbol{\pi}_{C_{k-1}}^T}_{\text{camera poses of m last states}}]^T$$

- We need to define
  - Motion model
  - Observation model

Kalman Filter: Motion Model

[Li et al., ICRA '13]

- Motion model (IMU)
  - Angular velocity

$$\underset{\substack{\uparrow\\ \text{measure-}\\ \text{ment}}}{\boldsymbol{\omega}_m} = \underset{\substack{\uparrow\\ \text{true}\\ \text{velocity}}}{{}^{I}\boldsymbol{\omega}} + \underset{\substack{\uparrow\\ \text{bias}}}{\mathbf{b}_{\mathbf{g}}} + \underset{\substack{\uparrow\\ \text{noise}}}{\mathbf{n}_{\mathbf{r}}}$$

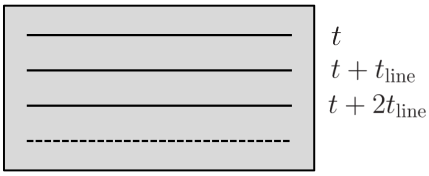
  - Linear acceleration

$$\underset{\substack{\uparrow\\ \text{measure-}\\ \text{ment}}}{\mathbf{a}_m} = \underset{\substack{\uparrow\\ \text{orient-}\\ \text{ation}}}{{}^{I}_{G}\mathbf{R}} \left(\underset{\substack{\uparrow\\ \text{true}\\ \text{acc.}}}{{}^{G}\mathbf{a}} - \underset{\substack{\uparrow\\ \text{gravity}}}{{}^{G}\mathbf{g}} \right) + \underset{\substack{\uparrow\\ \text{bias}}}{\mathbf{b}_{\mathbf{a}}} + \underset{\substack{\uparrow\\ \text{noise}}}{\mathbf{n}_{\mathbf{a}}}$$

Rolling Shutter

[Li et al., ICRA '13]

- Read-out time of the whole image $t_{\text{image}} = nt_{\text{line}}$
- Observation time of first row t
- Observation time of i-th row $t + it_{\text{line}}$



Kalman Filter: Observation Model

[Li et al., ICRA '13]

- Observations from KLT tracker
  - 2D observations of 3D points (3D positions estimated by triangulation, covered next week)
- Observation function

$$\mathbf{z}_j = \mathbf{h}({}^{C_j}\mathbf{p}_f) + \mathbf{n}_j$$

- Observation function with rolling shutter comp.

$$\mathbf{z}_j^{(n)} = \mathbf{h}({}^{C}\mathbf{p}_f(t_j + nt_d)) + \mathbf{n}_j^{(n)}$$

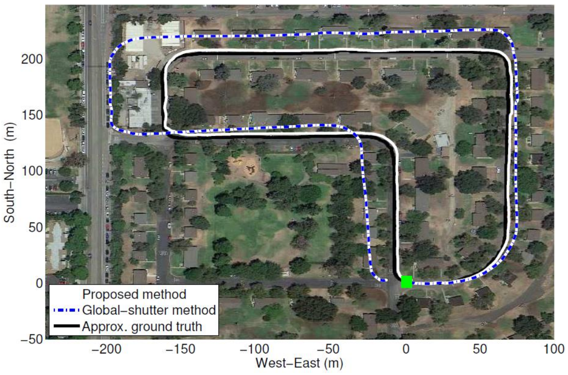
Results

[Li et al., ICRA '13]

- Galaxy S2 (live demo at ICRA with S3)
- IMU updates at 90 Hz (downsampled)
  - Gyroscope at 106 Hz
  - Accelerometer at 93 Hz
- Images at 15 Hz
  - Read-out time 32msec
- 610m and 900m trajectory
- Drift less than 0.8%, absolute scale from IMU

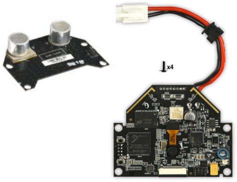
Results

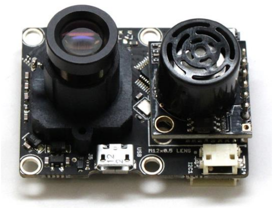
[Li et al., ICRA '13]



Commercial Solutions

- Pix4flow from ETH/3D robotics
  1 camera, IMU, ultrasound, 150 EUR
- Parrot Mainboard + Navigation board
  1 camera, IMU, ultrasound, 210 USD

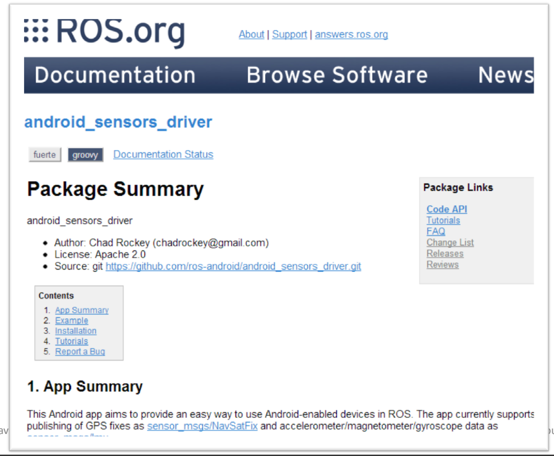




Talks at ROSCon 2013

- ROS Conference (in conjunction with ICRA)
- Talk by Chad Rockey (Willow Garage) on Android sensors driver
- Android app (Google Play store or github)
- Provides:
  - `/android/imu`
  - `/android/fix (GPS)`
  - `/camera/camera_info`
  - `/camera/image/compressed`

Android Sensors Driver



android_sensors_driver

Package Summary

android_sensors_driver

- Author: Chad Rockey (chadrockey@gmail.com)
- License: Apache 2.0
- Source: git https://github.com/ros-android/android_sensors_driver.git

Package Links

Code API
Tutorials
FAQ
Change List
Releases
Reviews

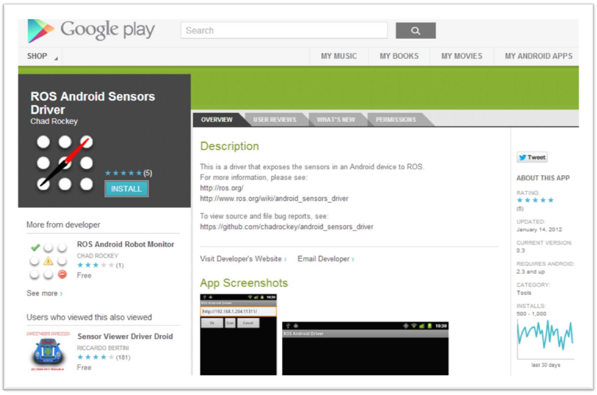
Contents

1. App Summary
2. Example
3. Installation
4. Tutorials
5. Report a Bug

1. App Summary

This Android app aims to provide an easy way to use Android-enabled devices in ROS. The app currently supports publishing of GPS fixes as sensor_msgs/NavSatFix and accelerometer/magnetometer/gyroscope data as

Android Sensors Driver



ROS Android Sensors Driver
Chad Rockey
★★★★★ (5)

Description

This is a driver that exposes the sensors in an Android device to ROS.
For more information, please see:
http://ros.org/
http://www.ros.org/wiki/android_sensors_driver

To view source and file bug reports, see:
https://github.com/chadrockey/android_sensors_driver

Visit Developer's Website › Email Developer ›

App Screenshots

More from developer

ROS Android Robot Monitor
CHAD ROCKEY
★★★★★ (1)
Free

See more ›

Users who viewed this also viewed

Sensor Viewer Driver Droid
RICCARDO BERTINI
★★★★★ (181)
Free

ABOUT THIS APP

RATING:
★★★★★
(5)

UPDATED:
January 14, 2012

CURRENT VERSION:
0.3

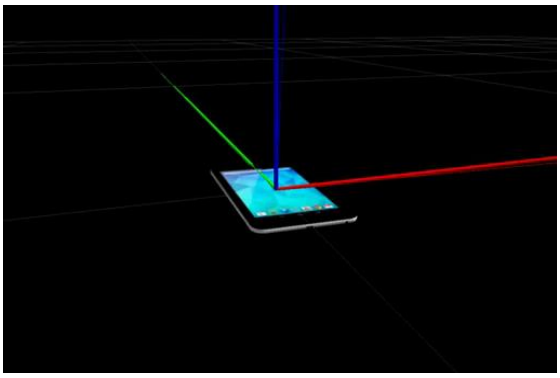
REQUIRES ANDROID:
2.3 and up

CATEGORY:
Tools

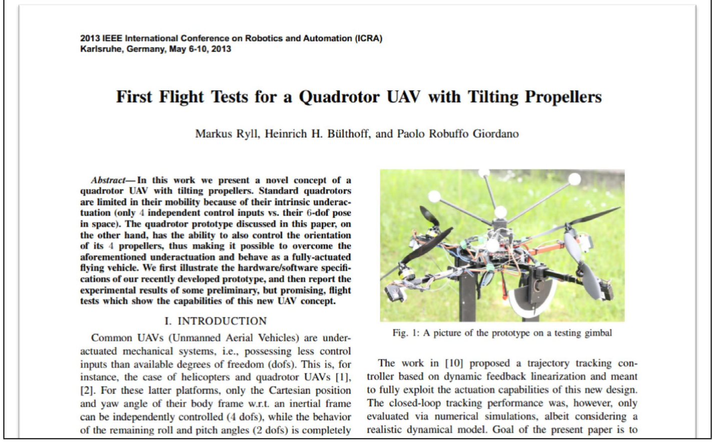
INSTALLS:
500 - 1,000

last 30 days

Android Sensors Driver



Cool ICRA Papers



2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

First Flight Tests for a Quadrotor UAV with Tilting Propellers

Markus Ryll, Heinrich H. Bülthoff, and Paolo Robuffo Giordano

Abstract—In this work we present a novel concept of a quadrotor UAV with tilting propellers. Standard quadrotors are limited in their mobility because of their intrinsic underactuation (only 4 independent control inputs vs. their 6-dof pose in space). The quadrotor prototype discussed in this paper, on the other hand, has the ability to also control the orientation of its 4 propellers, thus making it possible to overcome the aforementioned underactuation and behave as a fully-actuated flying vehicle. We first illustrate the hardware/software specifications of our recently developed prototype, and then report the experimental results of some preliminary, but promising, flight tests which show the capabilities of this new UAV concept.

I. INTRODUCTION

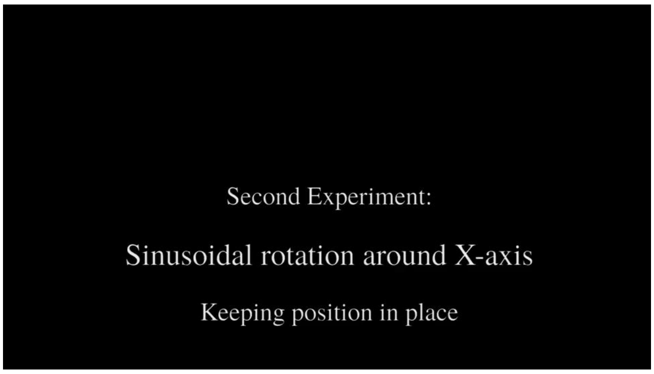
Common UAVs (Unmanned Aerial Vehicles) are underactuated mechanical systems, i.e., possessing less control inputs than available degrees of freedom (dofs). This is, for instance, the case of helicopters and quadrotor UAVs [1], [2]. For these latter platforms, only the Cartesian position and yaw angle of their body frame w.r.t. an inertial frame can be independently controlled (4 dofs), while the behavior of the remaining roll and pitch angles (2 dofs) is completely

Fig. 1: A picture of the prototype on a testing gimbal

The work in [10] proposed a trajectory tracking controller based on dynamic feedback linearization and meant to fully exploit the actuation capabilities of this new design. The closed-loop tracking performance was, however, only evaluated via numerical simulations, albeit considering a realistic dynamical model. Goal of the present paper is to

Tilting Propellors

[Ryll et al., ICRA '13]



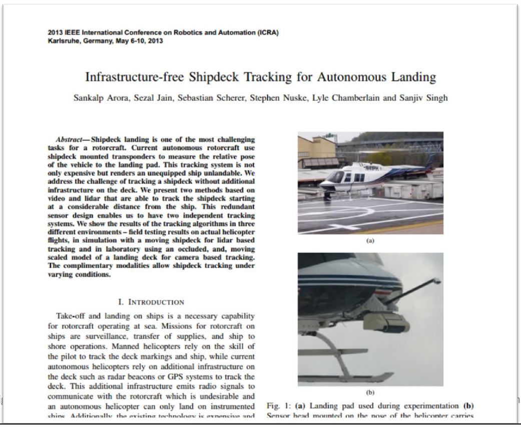
Second Experiment:

Sinusoidal rotation around X-axis

Keeping position in place

Shipdeck Tracking

[Arora et al., ICRA '13]



2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

Infrastructure-free Shipdeck Tracking for Autonomous Landing

Sankalp Arora, Sezal Jain, Sebastian Scherer, Stephen Nuske, Lyle Chamberlain and Sanjiv Singh

Abstract—Shipdeck landing is one of the most challenging tasks for a rotorcraft. Current autonomous rotorcraft use shipdeck mounted transponders to measure the relative pose of the vehicle to the landing pad. This tracking system is not only expensive but renders an unequipped ship unlandable. We address the challenge of tracking a shipdeck without additional infrastructure on the deck. We present two methods based on video and lidar that are able to track the shipdeck starting at a considerable distance from the ship. This redundant sensor design enables us to have two independent tracking systems. We show the results of the tracking algorithms in three different environments – field testing results on actual helicopter flights, in simulation with a moving shipdeck for lidar based tracking and in laboratory using an occluded, and, moving scaled model of a landing deck for camera based tracking. The complimentary modalities allow shipdeck tracking under varying conditions.

I. INTRODUCTION

Take-off and landing on ships is a necessary capability for rotorcraft operating at sea. Missions for rotorcraft on ships are surveillance, transfer of supplies, and ship to shore operations. Manned helicopters rely on the skill of the pilot to track the deck markings and ship, while current autonomous helicopters rely on additional infrastructure on the deck such as radar beacons or GPS systems to track the deck. This additional infrastructure emits radio signals to communicate with the rotorcraft which is undesirable and an autonomous helicopter can only land on instrumented ships. Additionally, the existing technology is expensive and

(a)

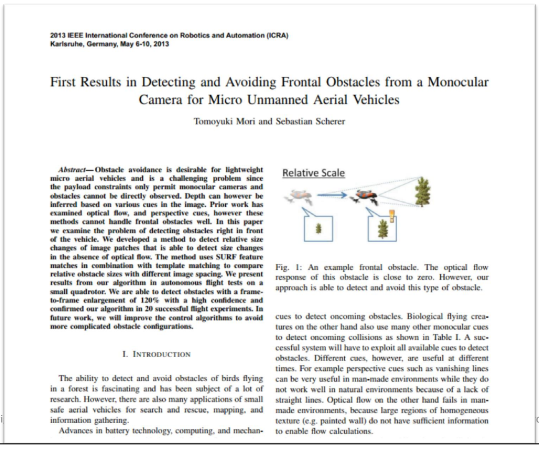
(b)

Fig. 1: **(a)** Landing pad used during experimentation **(b)** Sensor head mounted on the nose of the helicopter carries

Shipdeck Tracking

[Arora et al., ICRA '13]

Cool ICRA Papers



2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

First Results in Detecting and Avoiding Frontal Obstacles from a Monocular Camera for Micro Unmanned Aerial Vehicles

Tomoyuki Mori and Sebastian Scherer

Abstract—Obstacle avoidance is desirable for lightweight micro aerial vehicles and is a challenging problem since the payload constraints only permit monocular cameras and obstacles cannot be directly observed. Depth can however be inferred based on various cues in the image. Prior work has examined optical flow, and perspective cues, however these methods cannot handle frontal obstacles well. In this paper we examine the problem of detecting obstacles right in front of the vehicle. We developed a method to detect relative size changes of image patches that is able to detect size changes in the absence of optical flow. The method uses SURF feature matches in combination with template matching to compare relative obstacle sizes with different image spacing. We present results from our algorithm in autonomous flight tests on a small quadrotor. We are able to detect obstacles with a frame-to-frame enlargement of 120% with a high confidence and confirmed our algorithm in 20 successful flight experiments. In future work, we will improve the control algorithms to avoid more complicated obstacle configurations.

I. INTRODUCTION

The ability to detect and avoid obstacles of birds flying in a forest is fascinating and has been subject of a lot of research. However, there are also many applications of small safe aerial vehicles for search and rescue, mapping, and information gathering.

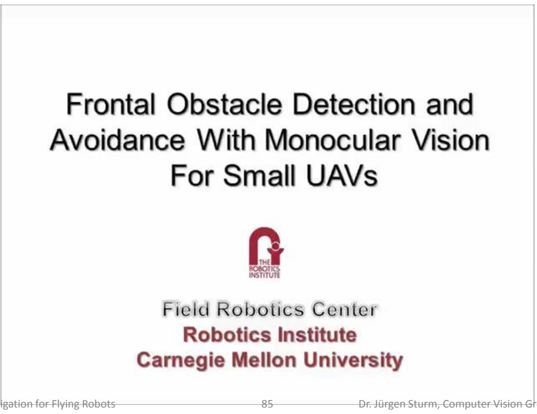
Advances in battery technology, computing, and mechan-

Fig. 1: An example frontal obstacle. The optical flow response of this obstacle is close to zero. However, our approach is able to detect and avoid this type of obstacle.

cues to detect oncoming obstacles. Biological flying creatures on the other hand also use many other monocular cues to detect oncoming collisions as shown in Table I. A successful system will have to exploit all available cues to detect obstacles. Different cues, however, are useful at different times. For example perspective cues such as vanishing lines can be very useful in man-made environments while they do not work well in natural environments because of a lack of straight lines. Optical flow on the other hand fails in man-made environments, because large regions of homogeneous texture (e.g. painted wall) do not have sufficient information to enable flow calculations.

Frontal Obstacle Avoidance

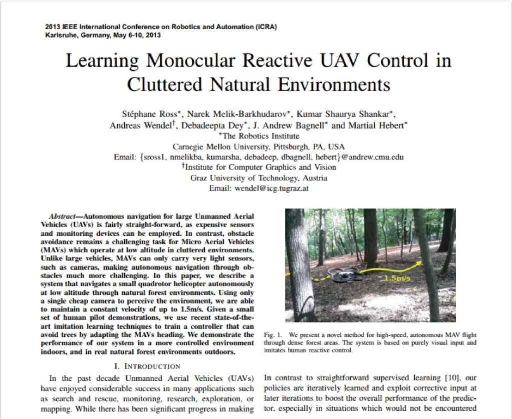
[Mori and Scherer, ICRA '13]



Frontal Obstacle Detection and Avoidance With Monocular Vision For Small UAVs

Field Robotics Center
Robotics Institute
Carnegie Mellon University

Cool ICRA Papers



2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

Learning Monocular Reactive UAV Control in Cluttered Natural Environments

Stéphane Ross*, Narek Melik-Barkhudarov*, Kumar Shaurya Shankar*,
Andreas Wendel†, Debadeepta Dey*, J. Andrew Bagnell* and Martial Hebert*
*The Robotics Institute
Carnegie Mellon University, Pittsburgh, PA, USA
Email: {sross1, nmelikba, kumarsha, debadeep, dbagnell, hebert}@andrew.cmu.edu
†Institute for Computer Graphics and Vision
Graz University of Technology, Austria
Email: wendel@icg.tugraz.at

***Abstract*—Autonomous navigation for large Unmanned Aerial Vehicles (UAVs) is fairly straight-forward, as expensive sensors and monitoring devices can be employed. In contrast, obstacle avoidance remains a challenging task for Micro Aerial Vehicles (MAVs) which operate at low altitude in cluttered environments. Unlike large vehicles, MAVs can only carry very light sensors, such as cameras, making autonomous navigation through obstacles much more challenging. In this paper, we describe a system that navigates a small quadrotor helicopter autonomously at low altitude through natural forest environments. Using only a single cheap camera to perceive the environment, we are able to maintain a constant velocity of up to 1.5m/s. Given a small set of human pilot demonstrations, we use recent state-of-the-art imitation learning techniques to train a controller that can avoid trees by adapting the MAVs heading. We demonstrate the performance of our system in a more controlled environment indoors, and in real natural forest environments outdoors.**

I. Introduction

In the past decade Unmanned Aerial Vehicles (UAVs) have enjoyed considerable success in many applications such as search and rescue, monitoring, research, exploration, or mapping. While there has been significant progress in making

Fig. 1. We present a novel method for high-speed, autonomous MAV flight through dense forest areas. The system is based on purely visual input and imitates human reactive control.

In contrast to straightforward supervised learning [10], our policies are iteratively learned and exploit corrective input at later iterations to boost the overall performance of the predictor, especially in situations which would not be encountered

Learning to Avoid Obstacles

[Ross et al., ICRA '13]

ICRA Proceedings

- All papers of a conference are (usually) collected into so-called proceedings
- Previously: One or more books
- Today: USB sticks or online

- Will put ICRA proceedings online (see website)
- Remember password (or ask by email)

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 marked spot/moving target
- **Your own idea here – be creative!**
- ...

Lessons Learned Today

- How to estimate the translational motion from camera images
- Which image patches are easier to track than others
- How to estimate 2D motion from camera images
- Summary of ICRA and ROSCon papers/talks

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

Structure From Motion

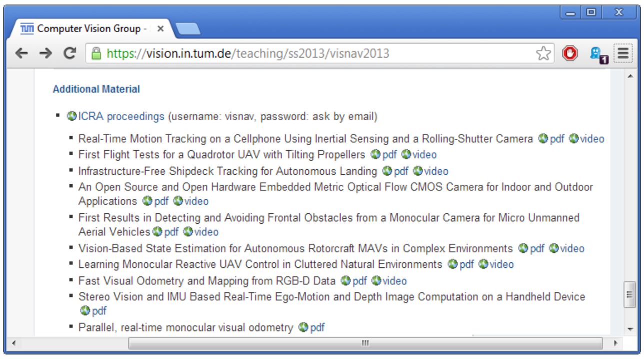
Dr. Jürgen Sturm

VISNAV Oral Team Exam

Date and Time	Student Name	Student Name	Student Name
Mon, July 29, 10am			
Mon, July 29, 11am			
Mon, July 29, 2pm			
Mon, July 29, 3pm			
Mon, July 29, 4pm			
Tue, July 30, 10am			
Tue, July 30, 11am			
Tue, July 30, 2pm			
Tue, July 30, 3pm			
Tue, July 30, 4pm			

Will put up this list in front of our secretary's office (02.09.052)

ICRA Papers+Videos are Online



Additional Material

- ICRA proceedings (username: visnav, password: ask by email)
  - Real-Time Motion Tracking on a Cellphone Using Inertial Sensing and a Rolling-Shutter Camera pdf video
  - First Flight Tests for a Quadrotor UAV with Tilting Propellers pdf video
  - Infrastructure-Free Shipdeck Tracking for Autonomous Landing pdf video
  - An Open Source and Open Hardware Embedded Metric Optical Flow CMOS Camera for Indoor and Outdoor Applications pdf video
  - First Results in Detecting and Avoiding Frontal Obstacles from a Monocular Camera for Micro Unmanned Aerial Vehicles pdf video
  - Vision-Based State Estimation for Autonomous Rotorcraft MAVs in Complex Environments pdf video
  - Learning Monocular Reactive UAV Control in Cluttered Natural Environments pdf video
  - Fast Visual Odometry and Mapping from RGB-D Data pdf video
  - Stereo Vision and IMU Based Real-Time Ego-Motion and Depth Image Computation on a Handheld Device pdf
  - Parallel, real-time monocular visual odometry pdf

Agenda for Today

- **This week:** basic ingredients of a visual SLAM system
  - Feature detection, descriptors and matching
  - Place recognition
  - 3D motion estimation
- **Next week:** bundle adjustment, graph SLAM, stereo cameras, Kinect
- **In two weeks:** map representations, mapping and (dense) 3D reconstruction

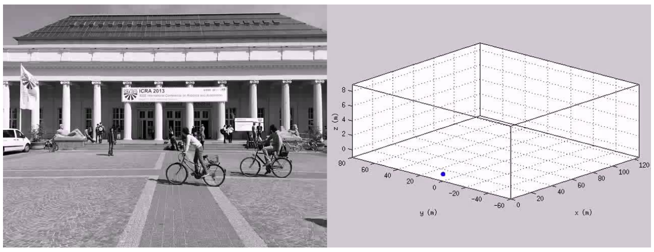
Last week: KLT Tracker

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
- Can use coarse-to-fine for larger motions

Visual Odometry

[Li et al., ICRA '13]



Limitations

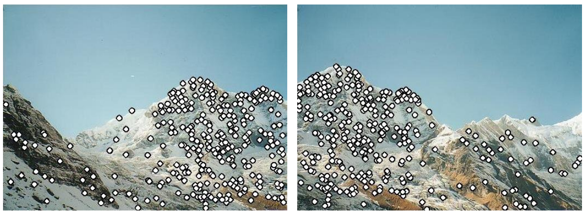
- Tracking is based on image gradients (dx/dy/dt)
  - Only works for small motions
  - Preferably high frame rate
- Cannot recover when tracks are lost
- How can we recognize previously seen patches?

Example: How to 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?

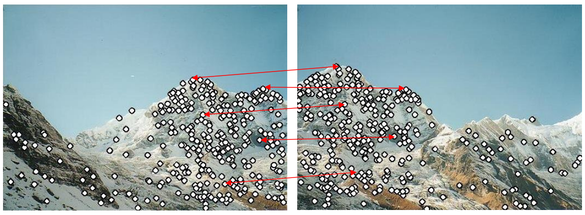
Matching with Features

- Detect features in both images



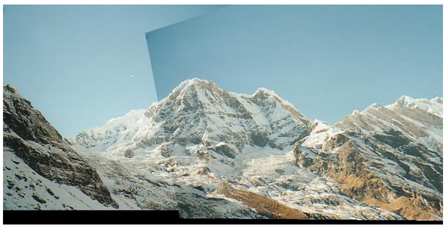
Matching with Features

- Detect features in both images
- Find corresponding pairs



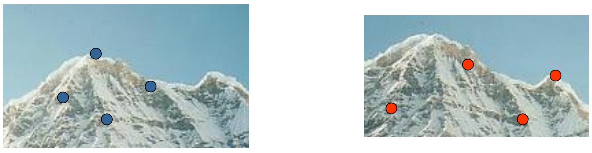
Matching with Features

- 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

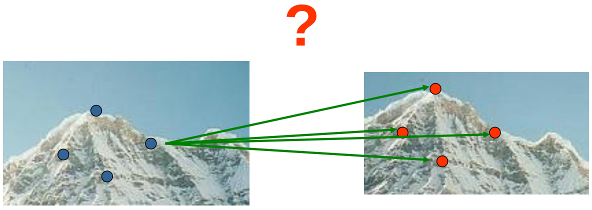


no chance to match!

→ We need a reliable detector

Matching with Features

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



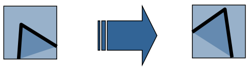
→ We need a reliable and distinctive descriptor

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

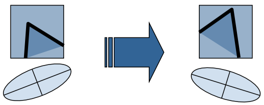
Harris Detector

- Rotation invariance?



Harris Detector

- Rotation invariance?

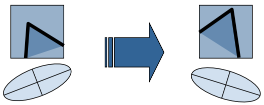


- Remember from last week

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

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

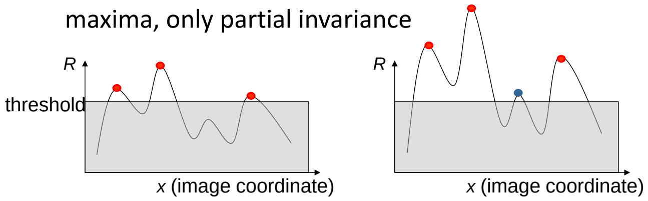
→ Corner response R is invariant to rotation

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

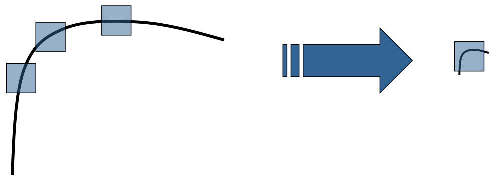


Harris Detector

- Invariant to scaling?

Harris Detector

- Not invariant to image scale

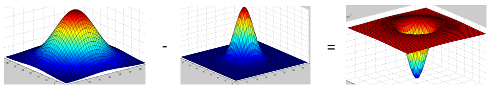


All points classified as edge

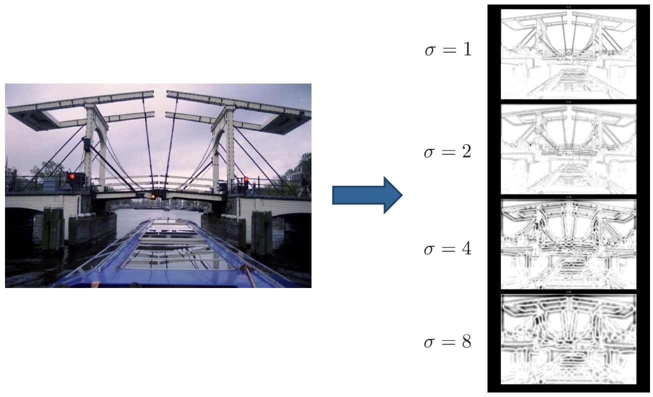
Point classified as corner

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

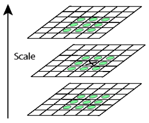


Example: Difference of Gaussians

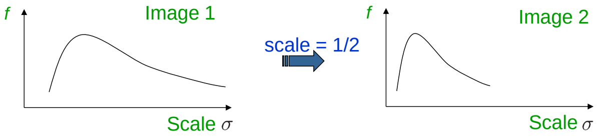


SIFT Detector

- Search for local maximum in space and scale



- Corner detections are invariant to scale change

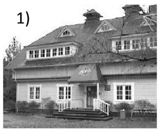


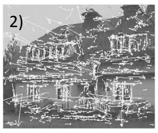
SIFT Detector

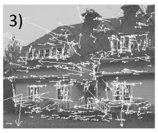
1. Detect maxima in scale-space
2. Non-maximum suppression
3. Eliminate edge points (check ratio of eigenvalues)
4. For each maximum, fit quadratic function and compute center at sub-pixel accuracy

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

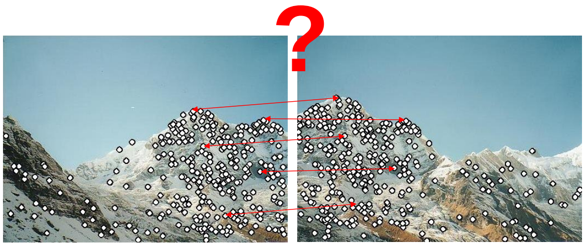






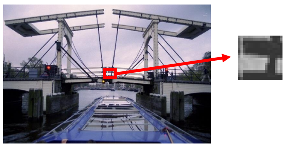
Feature Matching

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

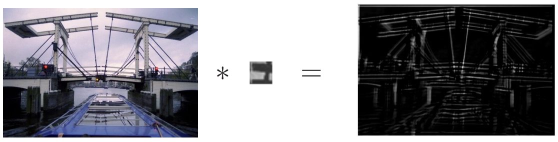


Template Convolution

- Extract a small as a template



- Convolve image with this template



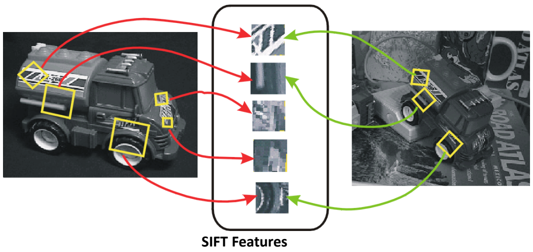
Template Convolution

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



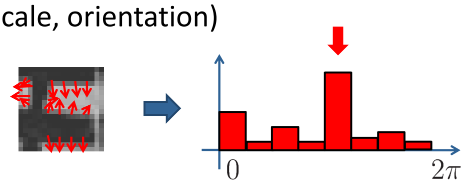
Scale Invariant Feature Transform (SIFT)

Approach

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

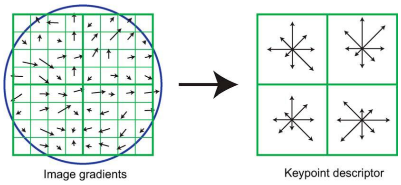
Select Dominant Orientation

- 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)

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

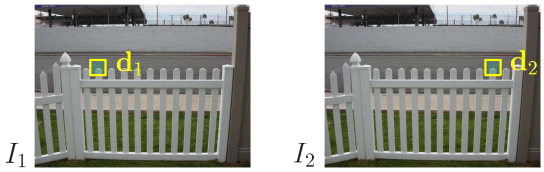
Feature Distance

How to define the difference between features?

- Simple approach is Euclidean distance (or SSD)

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

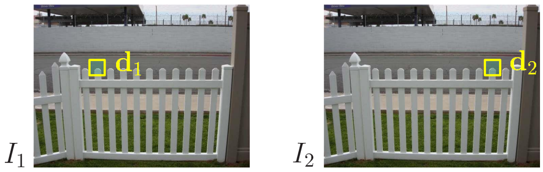
- Problem: can give good scores to ambiguous (bad) matches



Feature Distance

How to define the difference between features?

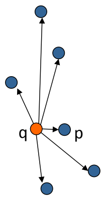
- Better approach $\mathrm{d}(\mathbf{d}_1, \mathbf{d}_2) = \|\mathbf{d}_1 - \mathbf{d}_2\| / \|\mathbf{d}_1 - \mathbf{d}_2'\|$
  with $\mathbf{d}_2$ best matching feature from I_2
  $\mathbf{d}_2'$ second best matching feature from I_2
- Gives small values for ambiguous matches



Efficient Matching

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

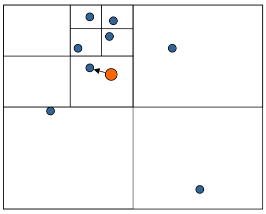
- Exhaustive search $\mathrm{O}(n^2)$



Efficient Matching

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

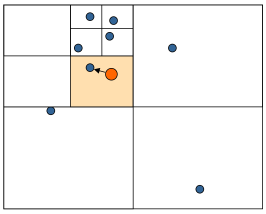
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)



Efficient Matching

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

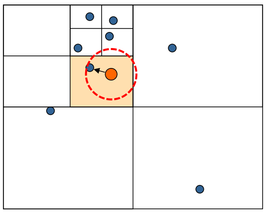
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found



Efficient Matching

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

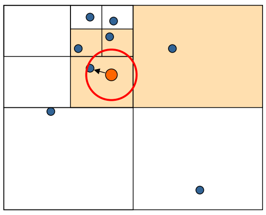
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found



Efficient Matching

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

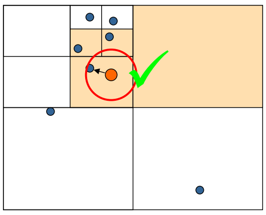
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found



Efficient Matching

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

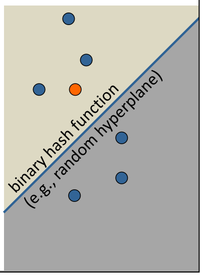
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found
  - Best-bin-first: use priority queue for unchecked leafs



Efficient Matching

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

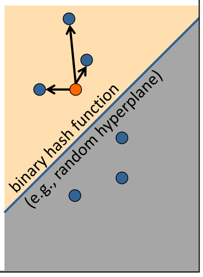
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
- Approximate search
  - Locality sensitive hashing
  - Approximate nearest neighbor



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
  - 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

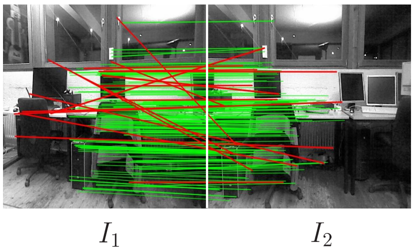
Other Descriptors (for intensity images)

- 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

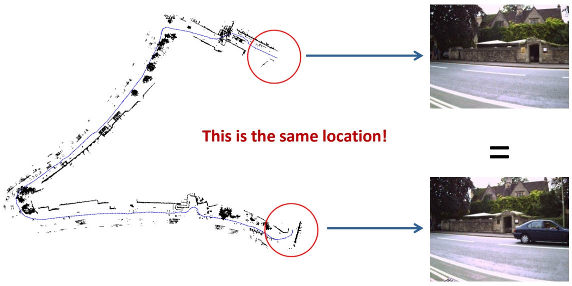
[Engelhard et al., 2011; Endres et al. 2012]

- Feature descriptor: SURF
- Feature matching: FLANN (approximate nearest neighbor)



Appearance-based Place Recognition

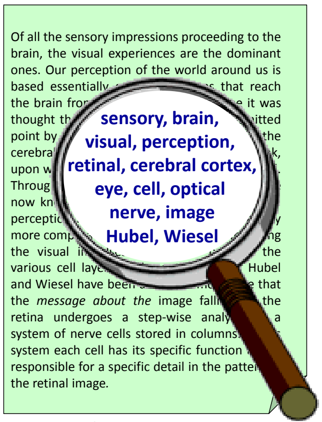
- How can we recognize that we have been visiting the same place before?



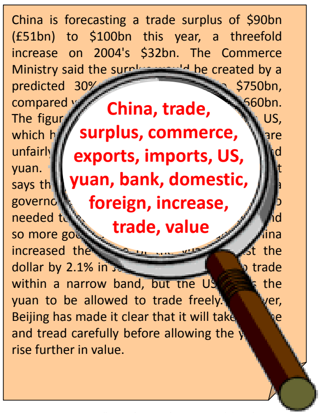
Appearance-based Place Recognition

- Brute-force matching with all previous images is slow (why?)
- How can we do this faster?

Analogy to Document Retrieval



Of all the sensory impressions proceeding to the
brain, the visual experiences are the dominant
ones. Our perception of the world around us is
based essentially … that reach
the brain fro… it was
thought th… itted
point by … the
cerebra… k,
upon w…
Throug…
now kn…
perceptic… y
more comp… ng
the visual i… the
various cell laye… Hubel
and Wiesel have been … e that
the *message about the* image falli… the
retina undergoes a step-wise analy… a
system of nerve cells stored in columns…
system each cell has its specific function …
responsible for a specific detail in the patte…
the retinal image.



China is forecasting a trade surplus of $90bn
(£51bn) to $100bn this year, a threefold
increase on 2004's $32bn. The Commerce
Ministry said the surp… be created by a
predicted 30% … $750bn,
compared … 660bn.
The figur… US,
which h… are
unfairly… d
yuan. … t
says th… a
governo… o
needed t… nd
so more go… ina
increased the … st the
dollar by 2.1% in … trade
within a narrow band, but the US … the
yuan to be allowed to trade freely. … ver,
Beijing has made it clear that it will take … e
and tread carefully before allowing the y…
rise further in value.

Object/Scene Recognition

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

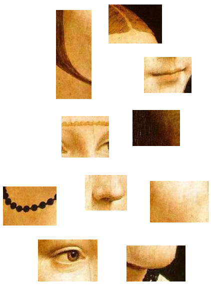
object

bag of visual words

Bag of Visual Words

- Visual words = (independent) features

face



features

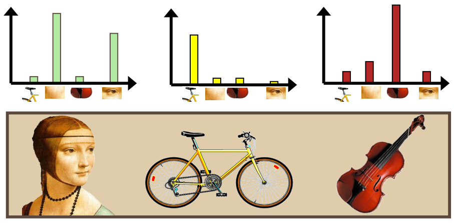
Bag of Visual Words

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

dictionary of visual words (codebook)

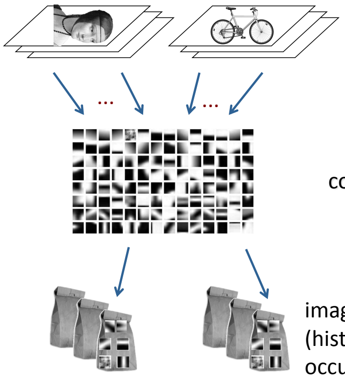
Bag of Visual Words

- 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

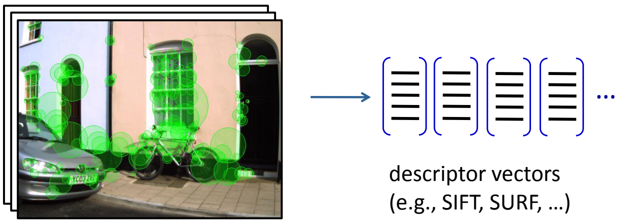


feature detection and extraction (e.g., SIFT, …)

codewords dictionary

image representation (histogram of word occurrences)

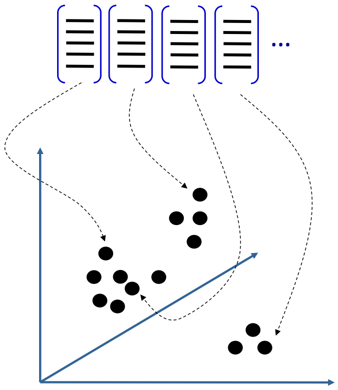
Learning the Dictionary



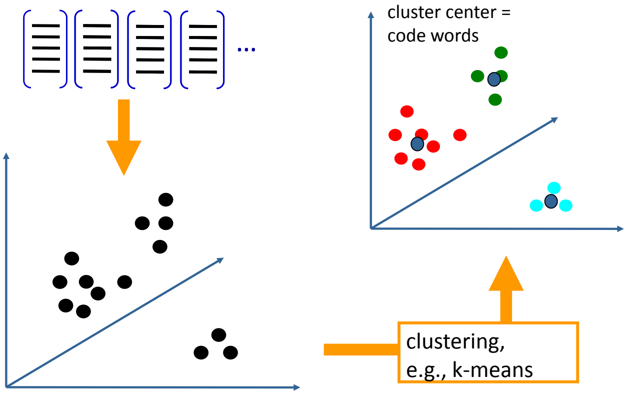
descriptor vectors (e.g., SIFT, SURF, …)

example patch

Learning the Dictionary

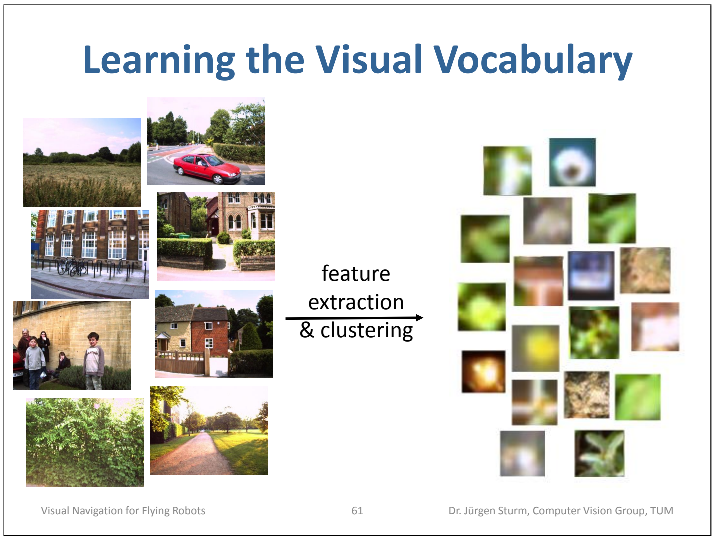


Learning the Dictionary

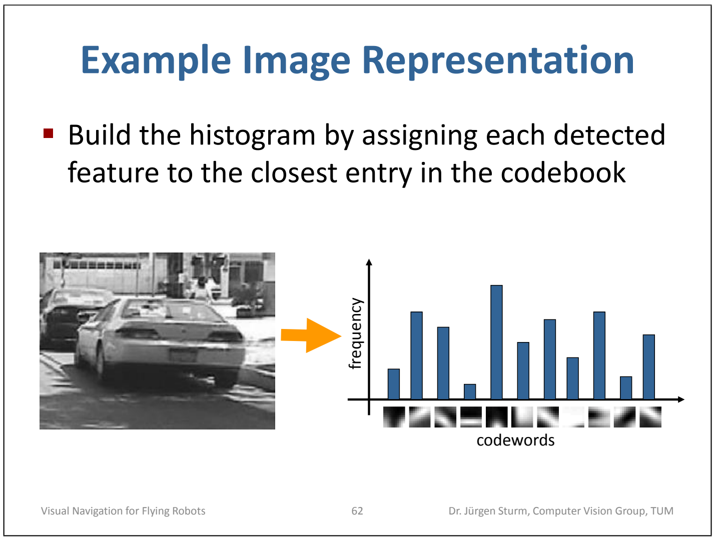


cluster center = code words

clustering, e.g., k-means

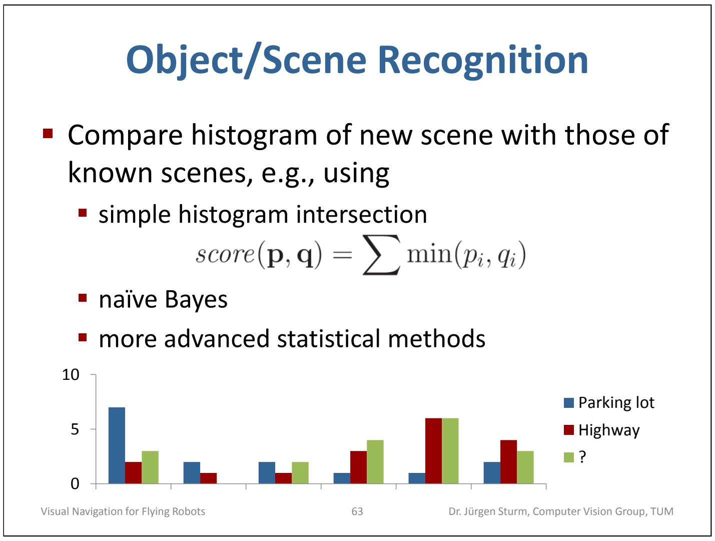


Learning the Visual Vocabulary



Example Image Representation

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



Object/Scene Recognition

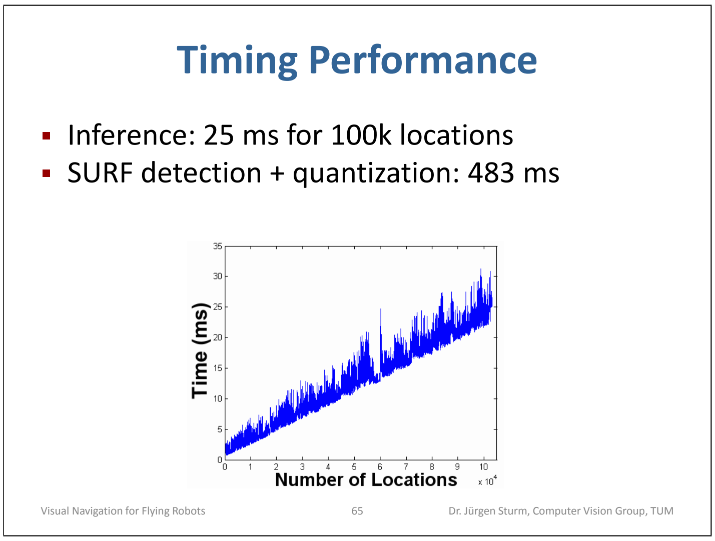
- Compare histogram of new scene with those of known scenes, e.g., using
  - simple histogram intersection

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

  - naïve Bayes
  - more advanced statistical methods

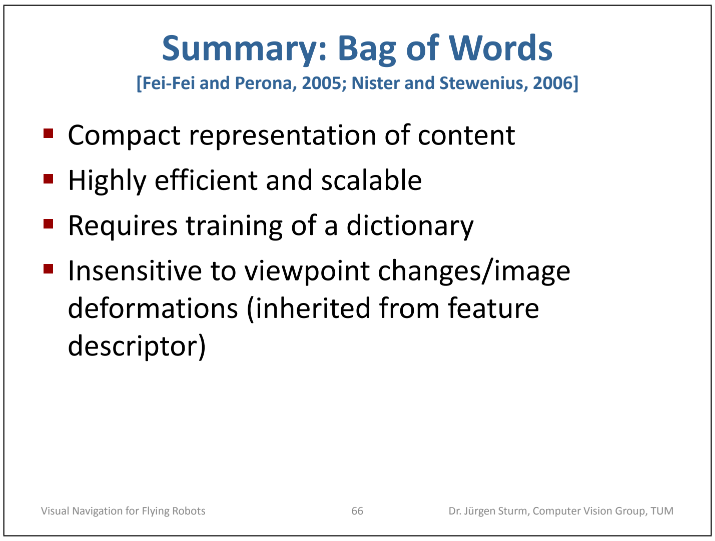
Example: FAB-MAP

[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)

Structure From Motion (SfM)

- Now we can retrieve relevant images and compute point correspondences between them
- What can we use them for?

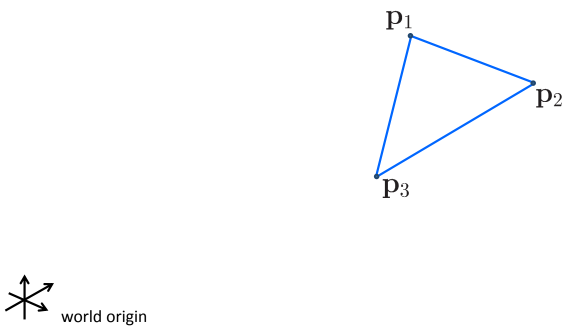
Four Important SfM Problems

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

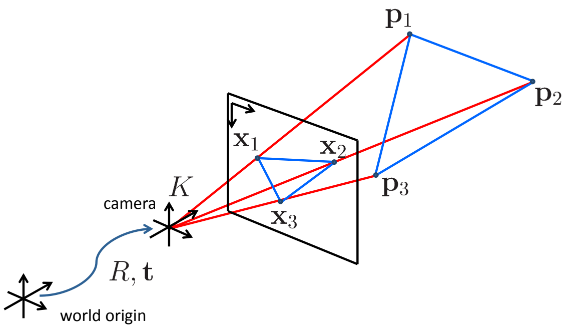
Four Important SfM Problems

- Each of these problems has many solution algorithms
- Approaches differ in:
  - Number of minimum points and assumptions of their configuration
  - Effect of noise (bias)
  - Conditioning
  - Simplicity vs. accuracy (linear vs. non-linear)

Camera Calibration (Perspective n-Point Problem)



Camera Calibration (Perspective n-Point Problem)



Camera Calibration

- **Given:** n 2D/3D correspondences $\mathbf{x}_i \leftrightarrow \mathbf{p}_i$
- **Wanted:** $M = K(R \;\; \mathbf{t})$
  such that $\tilde{\mathbf{x}}_i = M\mathbf{p}_i$
- Question: How many DOFs does M have?
- 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 $\mathbf{m} = (m_{11} \quad m_{12} \quad \dots \quad m_{34}) \in \mathbb{R}^{12}$

- 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

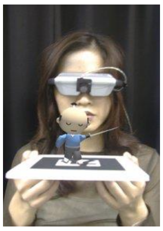
$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

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)

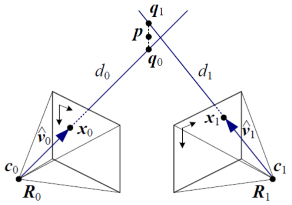
1. Threshold image
2. Detect edges and fit lines
3. Intersect lines to obtain corners
4. Estimate projection matrix M
5. Extract camera pose R,t (assume K is known)



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

Triangulation

- **Given:** n 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

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

Triangulation

- Multiply with denominator gives

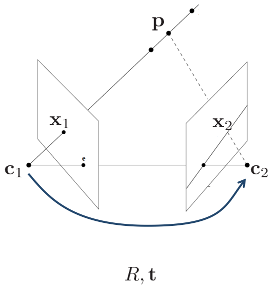
$$0 = (x_j m_{31} - m_{11})X + (x_j m_{32} - m_{12})Y + (x_j m_{33} - m_{13})Z + (x_j m_{34} - m_{14})W$$
$$0 = (y_j m_{31} - m_{21})X + (y_j m_{32} - m_{22})Y + (y_j m_{33} - m_{23})Z + (y_j m_{34} - m_{24})W$$

Solve for $\mathbf{p} = (X\ Y\ Z\ W)^\top$ using:

- Linear least squares with W=1
- Linear least squares using SVD
- Non-linear least squares of the residuals (most accurate)

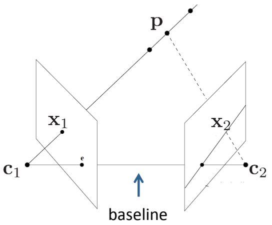
Epipolar Geometry

- Let's consider two cameras that observe a 3D world point



Epipolar Geometry

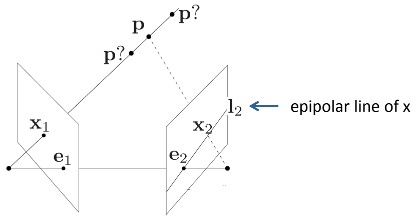
- The line connecting both camera centers is called the **baseline**



baseline
(line joining both camera centers)

Epipolar Geometry

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

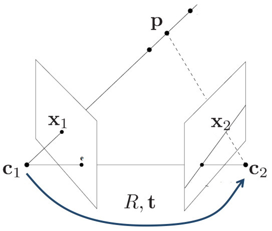


- A point in one image "generates" a line in another image (called the **epipolar line**)

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



Epipolar Geometry

- Left line in **right** camera frame $\mathbf{p}_1' = R d_1 \hat{\mathbf{x}}_1 + t$
- Right line in right camera frame $\mathbf{p}_2 = d_2 \hat{\mathbf{x}}_2$

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

- Intersection of both lines

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

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

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

3D Motion Estimation

- **Given:** 2 camera images
  n point correspondences
- **Wanted:** Camera motion R,t (up to scale)

- Solutions:
  - 8-point algorithm
  - normalized 8-point algorithm
  - 6-point algorithm
  - 5-point algorithm

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

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

Step 1: Estimate E

- Each correspondence gives us one constraint

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

- 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]

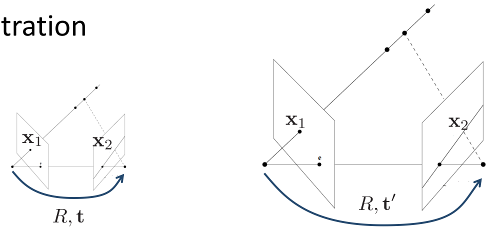
- Noise in the point observations is unequally distributed in the constraints, e.g.,

$$\begin{array}{lccccccc} \text{double noise} & \boxed{x_1x_2e}_{11} & + & y_1x_2e_{12} & + & x_2e_{13} & + & \\ & x_1y_2e_{21} & + & y_1y_2e_{22} & + & y_2e_{23} & + & \\ \text{normal noise} & \boxed{x_1e}_{31} & + & y_1e_{32} & + & \boxed{1e}_{33} & = & 0 \\ & & & & & \text{noise free} & & \end{array}$$

- 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°

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

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

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

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

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

Step 2b: Recover R

- Matrices U,V are not guaranteed to be rotations (sign flip still yields a valid SVD)

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

- Identify the correct solution using
  - Select those two solutions with $\det R = 1$
  - Triangulate points in 3D
  - Select the solution with the largest number of points **in front** of the camera

Summary: 8-Point Algorithm

Given: Image pair

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

- Compute correspondences
- Compute essential matrix
- Extract camera motion

Lessons Learned Today

- … how to detect and match feature points
- … how to efficiently recognize places
- … how to estimate the camera pose and to triangulate points

Visual Navigation for Flying Robots

Simultaneous Localization and Mapping

Dr. Jürgen Sturm

Agenda for Today

- Outlier rejection using RANSAC
- Laser-based motion estimation
- The SLAM problem
- Pose graph SLAM
- Map optimization

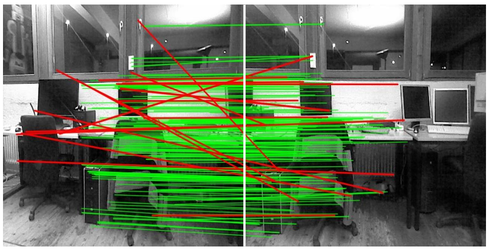
Remember: 8-Point Algorithm

Given: Image pair

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

- Find at least 8 correspondences
- Compute essential matrix
- Extract camera motion

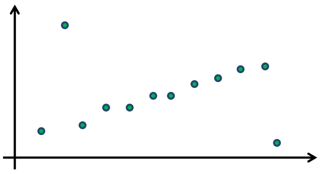
How To Deal With Outliers?



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

Robust Estimation

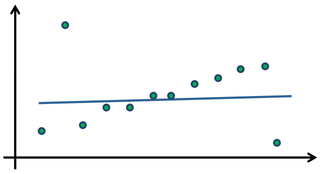
Example: Fit a line to 2D data containing outliers



- Input data is a mixture of
  - Inliers (perturbed by Gaussian noise)
  - Outliers (unknown distribution)
- Let’s fit a line using least squares…

Robust Estimation

Example: Fit a line to 2D data containing outliers



- Input data is a mixture of
  - Inliers (perturbed by Gaussian noise)
  - Outliers (unknown distribution)
- Least squares fit gives poor results!

RANdom SAmple Consensus (RANSAC)

[Fischler and Bolles, 1981]

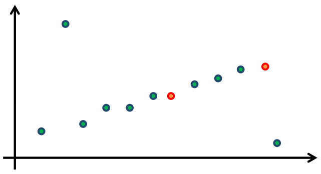
Goal: Robustly fit a model to a data set S which contains outliers

Algorithm:

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

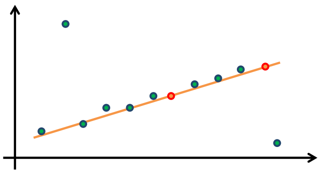
Example

- Step 1: Sample a random subset



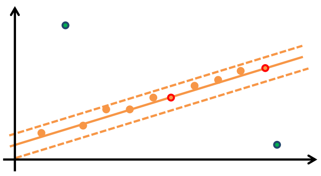
Example

- Step 2: Fit a model to this subset



Example

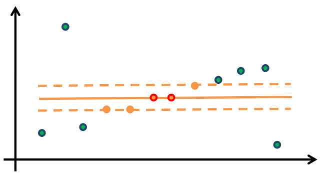
- Step 3: Classify points as inliers and outliers (e.g., using a threshold distance)



→ 10 inliers, 2 outliers

Example

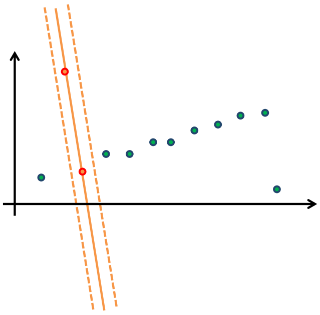
- Step 4: Repeat steps 1-3 for N iterations



Iteration 2:
→ 5 inliers, 7 outliers

Example

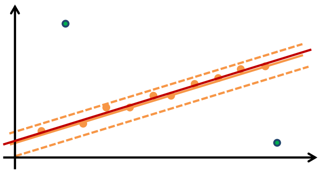
- Step 4: Repeat steps 1-3 for N iterations



Iteration 3:
→ 2 inliers, 10 outliers

Example

- Step 5: Select the best model (most inliers), the re-fit model using all inliers



Best model:
Iteration 1
(10 inliers, 2 outliers)

How Many Iterations Do We Need?

- For a probability of success p, we need

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)} \text{ iterations}$$

for subset size s and outlier ratio ϵ

- E.g., for p=0.99:

	Required points s	Outlier ratio ε						
		10 %	20 %	30 %	40 %	50 %	60 %	70 %
Line	2	3	5	7	11	17	27	49
Plane	3	4	7	11	19	35	70	169
Essential matrix	8	9	26	78	272	1177	7025	70188

Summary on RANSAC

- Efficient algorithm to estimate a model from noisy and outlier-contaminated data
- RANSAC is used today very widely
- Often used in feature matching / visual motion estimation
- Many improvements/variants (e.g., PROSAC, MLESAC, …)

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 …)

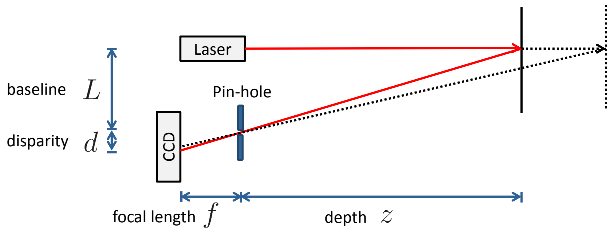
Laser Triangulation

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

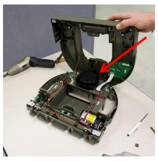
- Function principle

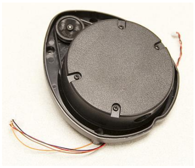


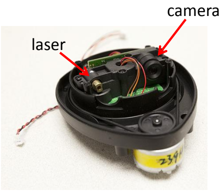
- Depth triangulation $z = f\frac{L}{d}$
(note: same for stereo disparities)

Example: Neato XV-11

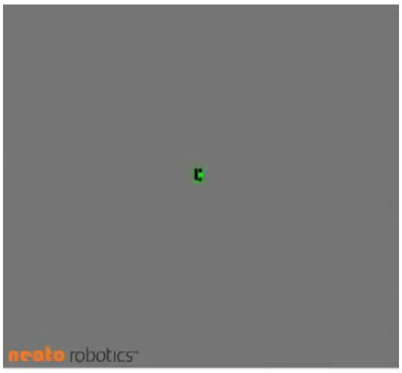
- K. Konolige, "A low-cost laser distance sensor", ICRA 2008
- Specs: 360deg, 10Hz, 30 USD







How Does the Data Look Like?



Laser Scanner

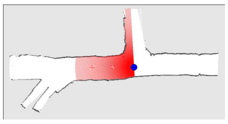
- Measures angles and distances to closest obstacles

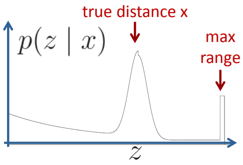
$$\mathbf{z} = (\theta_1, z_1, \ldots, \theta_n, z_n) \in \mathbb{R}^{2n}$$

- 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?

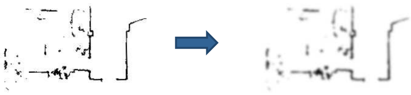
Laser-based Motion Estimation

How can we best align two laser scans?

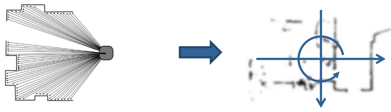
- Exhaustive search
- Iterative minimization (ICP)

Exhaustive Search

- Estimate a map using first scan and sensor model

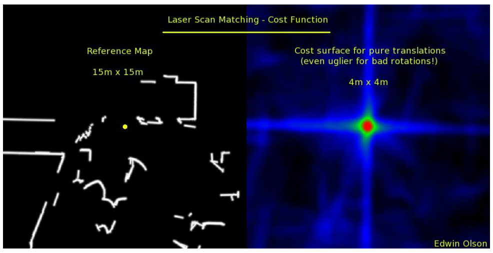


- Sweep second scan over map, compute correlation and select best pose

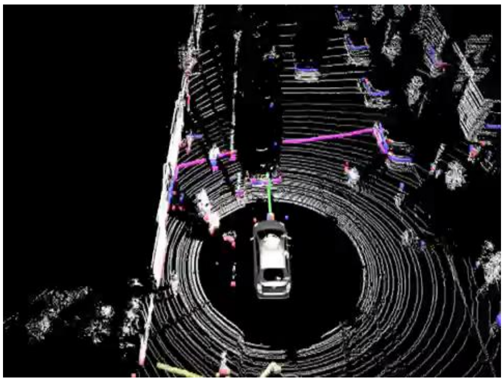


Example: Exhaustive Search [Olson, ICRA '09]

- Multi-resolution correlative scan matching
- Real-time by using GPU
- Remember: SE(2) has 3 DOFs


Laser Scan Matching - Cost Function
Reference Map
15m x 15m
Cost surface for pure translations
(even uglier for bad rotations!)
4m x 4m
Edwin Olson

Does Exhaustive Search Generalize To 3D As Well?



Iterative Closest Point (ICP)

- **Given:** Two corresponding point sets (clouds)

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

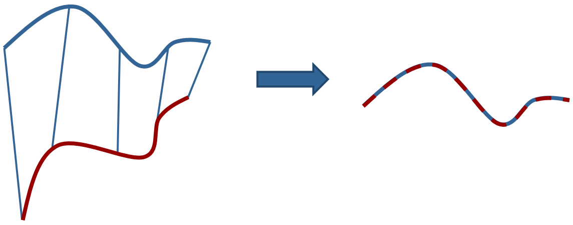
- **Wanted:** Translation $\mathbf{t}$ and rotation R that minimize the sum of the squared error

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

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

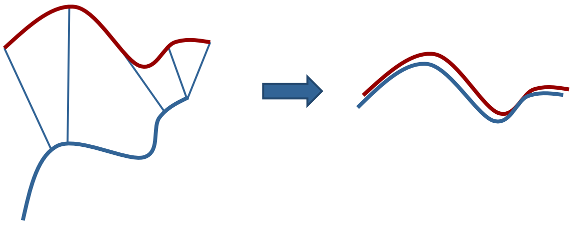
$$R = UV^\top$$

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

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

Unknown Correspondences

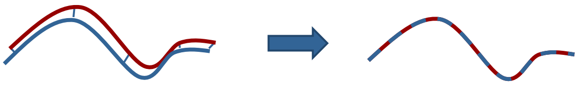
- If the correct correspondences are not known, it is generally impossible to determine the optimal transformation in one step



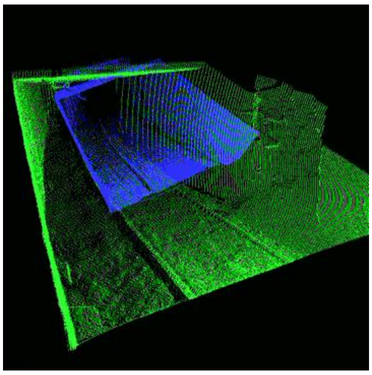
ICP Algorithm

[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

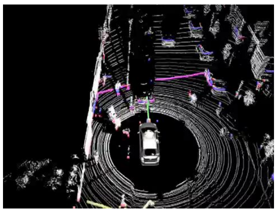
- Various aspects of performance
  - Speed
  - Stability (local minima)
  - Tolerance w.r.t. noise and/or outliers
  - Basin of convergence (maximum initial misalignment)
- Choice depends on data and application

Selecting Source Points

- Use all points
- Random sampling
- Spatially uniform sub-sampling
- Feature-based sampling

Spatially Uniform Sampling

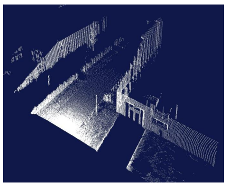
- Density of points usually depends on the distance to the sensor → no uniform distribution
- Can lead to a bias in ICP



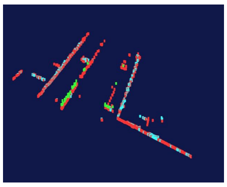
Feature-based Sampling

Detect interest points (same as with images)

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



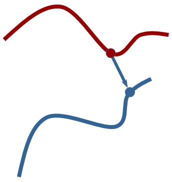
3D Scan (~200.000 Points)



Extracted Features (~5.000 Points)

Closest Point Matching

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



- Closest-point matching generally stable, but slow

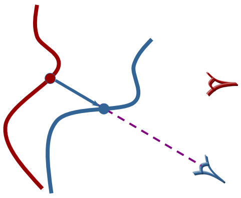
Speeding Up Correspondence Search

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)

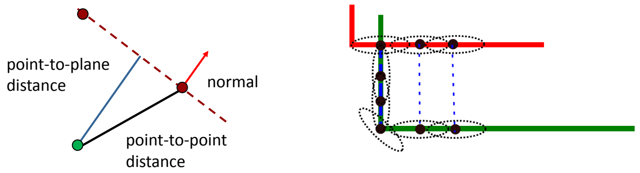
Projection-based Matching

- 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, RSS 2009]

Minimization

- Only point-to-point metric has closed form solution(s)
- Other error metrics require non-linear minimization methods

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

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 acronym SLAM stands for “simultaneous localization and mapping”.

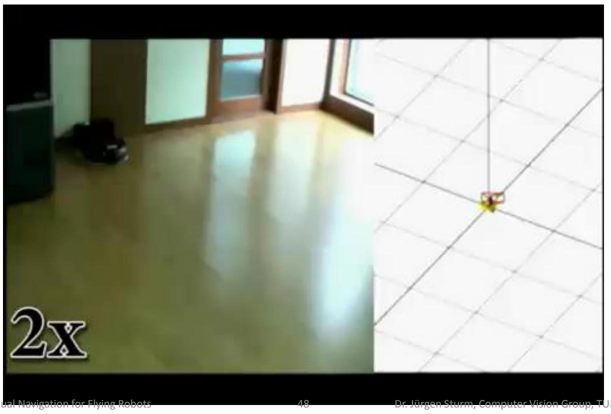
SLAM Applications

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

Examples

- At home: vacuum cleaner, lawn mower
- Air: inspection, transportation, surveillance
- Underwater: reef/environmental monitoring
- Underground: search and rescue
- Space: terrain mapping, navigation

SLAM with Ceiling Camera (Samsung Hauzen RE70V, 2008)



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

SfM vs. SLAM

- Structure from Motion (SfM)
  - Monocular/stereo camera
  - Sometimes uncalibrated sensors (e.g., Flickr images)
- Simultaneous Localization and Mapping (SLAM)
  - Multiple sensors: Laser scanner, ultrasound, monocular/stereo camera, GPS, …
  - Typically in combination with an odometry sensor
  - Typically pre-calibrated sensors

Remember: 3D Transformations

- Representation as a homogeneous matrix

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

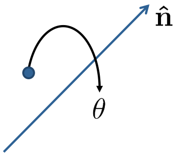
Pro: easy to concatenate and invert
Con: not minimal

- Representation as a twist coordinates

$$\boldsymbol{\xi} = (\underbrace{\omega_x \quad \omega_y \quad \omega_z}_{\text{angular velocity}} \quad \underbrace{v_x \quad v_y \quad v_z}_{\text{linear velocity}})^\top \in \mathbf{R}^6$$

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

Remember: 3D Rotation as 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 representation

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)

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

(or compute using Rodriguez' formula)

Notation

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

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

- … as transformation matrices

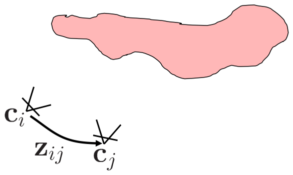
$$M_1, M_2, \ldots, M_n$$

- … 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

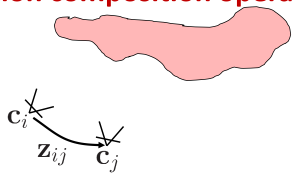


Incremental Motion Estimation

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

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

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

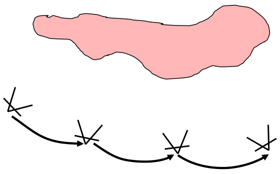


$$\mathbf{c}_j = \mathbf{c}_i \oplus \mathbf{z}_{ij}$$

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

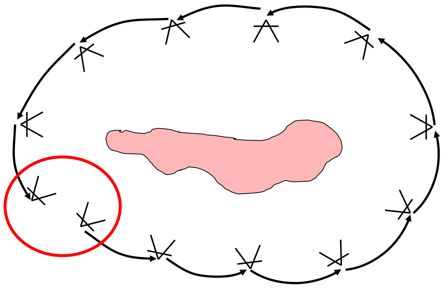
Incremental Motion Estimation

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



Incremental Motion Estimation

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

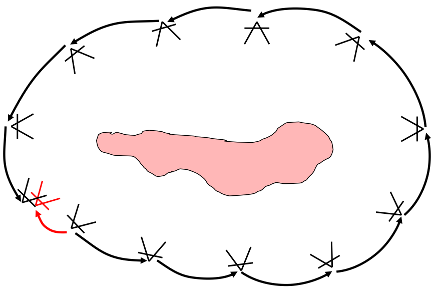


Loop Closures

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

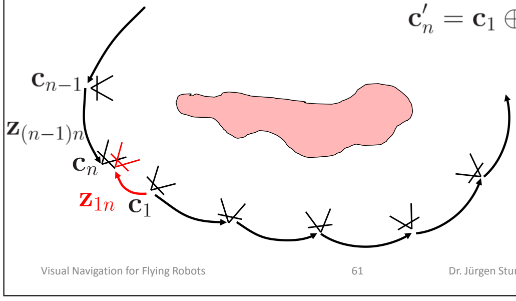
Incremental Motion Estimation

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



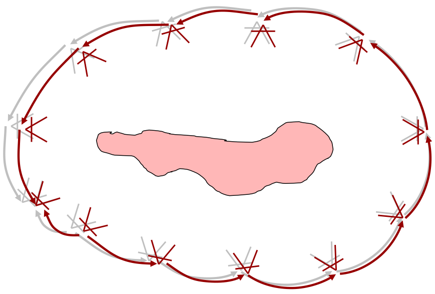
Incremental Motion Estimation

- **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

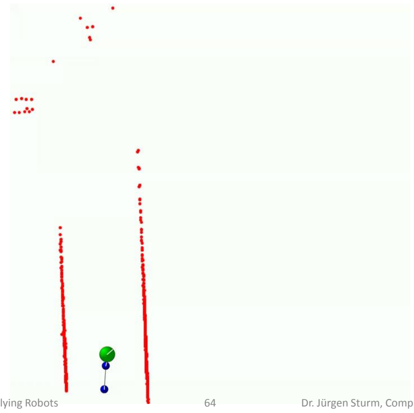


Graph SLAM

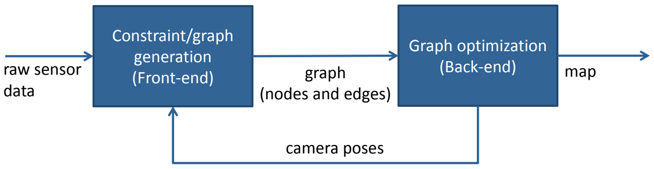
[Thrun and Montemerlo, 2006; Olson et al., 2006]

- 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



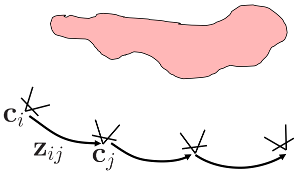
Graph SLAM Architecture


raw sensor data
Constraint/graph generation (Front-end)
graph (nodes and edges)
Graph optimization (Back-end)
map
camera poses

- 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 relative pose 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

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

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

- Given the correct map $\mathbf{x}$, this difference is the result of observation/sensor noise…

Error Function

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

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

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

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

- 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

Linearization and Derivation

- Error function

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

- Linearize the error function around the initial guess

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

Linearizing the Error Function

- Approximate the error function around an initial guess $\mathbf{x} \in \mathbb{R}^{6n}$ using Taylor expansion

$$\mathbf{e}_{ij}(\mathbf{x}+\Delta\mathbf{x}) \simeq \mathbf{e}_{ij}(\mathbf{x}) + J_{ij}\Delta\mathbf{x} \qquad (\in \mathbb{R}^6)$$

with increment

$$\Delta\mathbf{x} \in \mathbb{R}^{6n}$$

and Jacobian

$$J_{ij}(\mathbf{x}) = \begin{pmatrix} \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_1} & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_2} & \cdots & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_n} \end{pmatrix} \in \mathbb{R}^{6\times 6n}$$

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

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

(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
$$\mathbf{b}^\top = \sum_{ij} \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij} \qquad H = \sum_{ij} J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$
2. Solve the linear system to get new increment
$$H\Delta\mathbf{x} = -\mathbf{b}$$
3. Update previous estimate
$$\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$$

Structure of the Minimization Problem

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 $\mathbf{b}^\top = \sum_{ij} \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij} \in \mathbb{R}^{6n}$

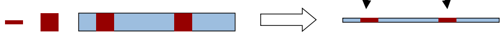
$$H = \sum_{ij} J_{ij}^\top \Sigma_{ij}^{-1} J_{ij} \in \mathbb{R}^{6n \times 6n}$$

this quickly gets huge!

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

Illustration of the Structure

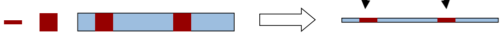
$$\mathbf{b}_{ij}^\top = \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$



Non-zero only at $\mathbf{c}_i$ and $\mathbf{c}_j$

Illustration of the Structure

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



Non-zero only at $\mathbf{c}_i$ and $\mathbf{c}_j$

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

Non-zero on the main diagonal at $\mathbf{c}_i$ and $\mathbf{c}_j$

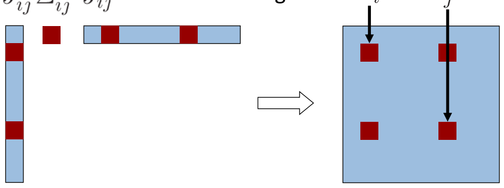
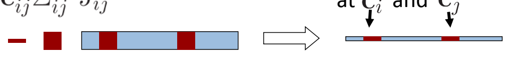


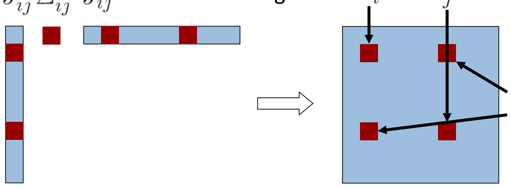
Illustration of the Structure

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



Non-zero only at $\mathbf{c}_i$ and $\mathbf{c}_j$

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

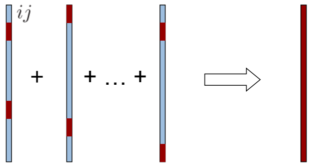


Non-zero on the main diagonal at $\mathbf{c}_i$ and $\mathbf{c}_j$

... and at the blocks *ij,ji*

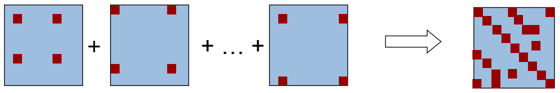
Illustration of the Structure

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



b: dense vector

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



H: sparse block structure with main diagonal

Sparsity of the Hessian

- Remember: We have to solve $H\Delta\mathbf{x} = -\mathbf{b}$
- 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 $$e_{12} = z_{12} - \bar{z}_{12}$$ $$= z_{12} - (c_2 - c_1) = 1 - (0 - 0) = 1$$
- 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

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

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 pose fits
- **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 pose fits
- **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

- **Observations:**
  - Gauss-Newton method typically converges very quickly
  - Sometimes diverges when initial solution is far off
  - Gradient descent (with line search) never diverges
- **How can we combine the advantages of both minimization methods?**

Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor

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

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

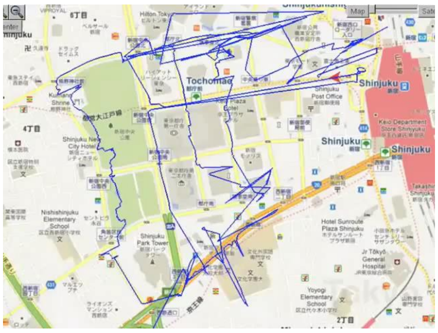
- What is the effect of this damping factor?
  - Small λ → same as least squares
  - Large λ → steepest descent (with small step size)
- **Algorithm**
  - If error decreases, accept $\Delta\mathbf{x}$ and reduce λ
  - If error increases, reject $\Delta\mathbf{x}$ and increase λ

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]
  - Ceres [Google, 2012]
- Other extensions:
  - Robust error functions
  - Alternative parameterizations

Example: Google Street View Map Optimization with Ceres Solver

[Google, 2012]



Example: RGB-D SLAM

[Engelhard et al., 2011; Endres et al., 2012]

- **Given:** Kinect data
  - Color image
  - Registered depth image (=point cloud)
- **Wanted:**
  - Camera poses
  - Aligned point cloud

Example: RGB-D SLAM

[Engelhard et al., 2011; Endres et al., 2012]

- Step 1: Extract 2D features (SIFT)

Example: RGB-D SLAM

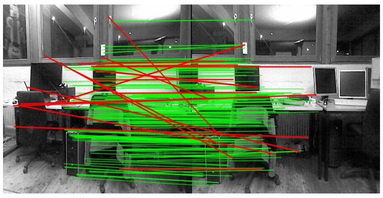
[Engelhard et al., 2011; Endres et al., 2012]

- Step 1: Extract 2D features (SIFT)
- Step 2: Associate features with 3D points

Example: RGB-D SLAM

[Engelhard et al., 2011; Endres et al., 2012]

- Step 1: Extract 2D features (SIFT)
- Step 2: Associate features with 3D points
- Step 3: Find corresponding points (RANSAC)
- Step 4: Estimate camera motion (ICP)



Example: RGB-D SLAM

[Engelhard et al., 2011; Endres et al., 2012]

RGBD SLAM
with
ROS + Kinect

Lessons Learned Today

- How to separate inliers from outliers using RANSAC
- How to align point clouds using ICP
- How to model the SLAM problem in a graph
- How to optimize the map using non-linear least squares

Computer Vision Group
Prof. Daniel Cremers
Technische Universität München

Visual Navigation for Flying Robots

Bundle Adjustment and Dense 3D Reconstruction

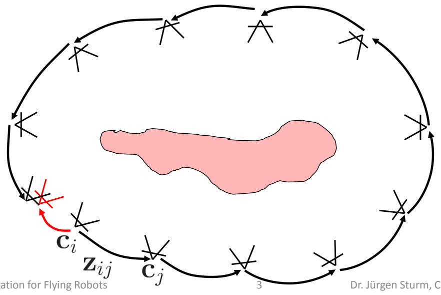
Dr. Jürgen Sturm

Agenda for Today

- Bundle adjustment
- Depth cameras
- Occupancy grid maps
- Signed distance functions

Reminder: Pose Graph SLAM

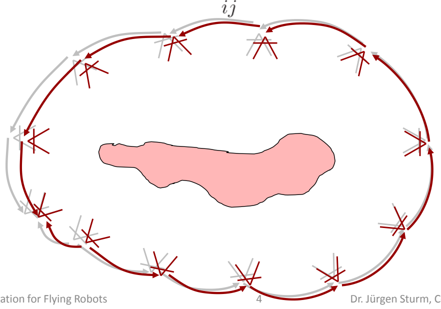
- **Given:** Set of relative pose observations $\mathbf{z}_{ij} \in \mathbb{R}^6$
- **Wanted:** Set of camera poses $\mathbf{c}_1, \ldots, \mathbf{c}_n \in \mathbb{R}^6$



Reminder: Pose Graph SLAM

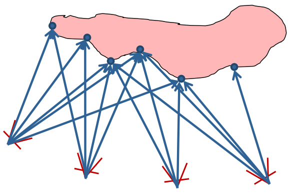
- **Goal:** Minimize the error over all constraints

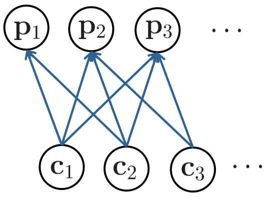
$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$



Bundle Adjustment

- 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





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

Primary Structure

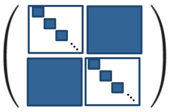
- Characteristic structure

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

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

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

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

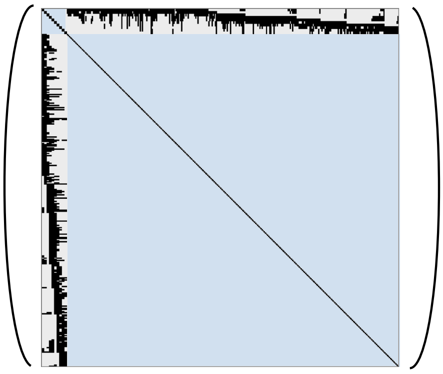
- If D is invertible, then (using Gauss elimination)

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

- **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)

$H =$

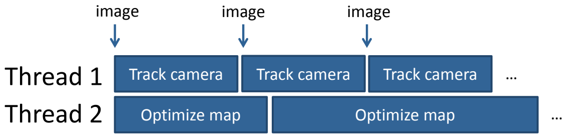


Two Examples

- 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



image

image

image

Thread 1

Thread 2

Track camera

Track camera

Track camera

...

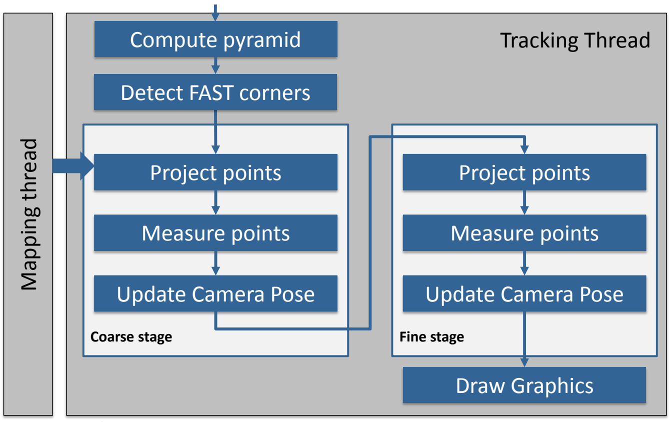
Optimize map

Optimize map

...

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

PTAM – Tracking Thread



Tracking Thread

Mapping thread

Compute pyramid

Detect FAST corners

Project points

Measure points

Update Camera Pose

Coarse stage

Project points

Measure points

Update Camera Pose

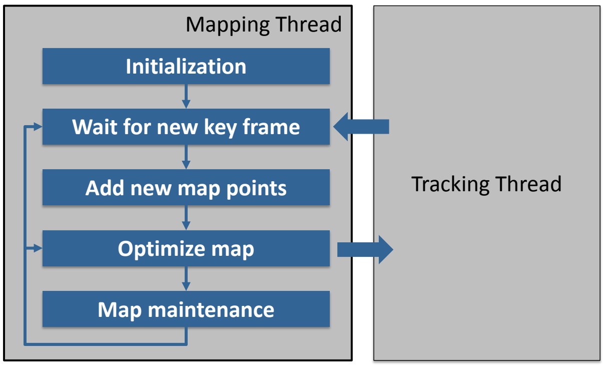
Fine stage

Draw Graphics

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



Mapping Thread

Initialization

Wait for new key frame

Add new map points

Optimize map

Map maintenance

Tracking 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

Parallel Tracking and Mapping
for Small AR Workspaces

Extra video results made for
ISMAR 2007 conference

Georg Klein and David Murray
Active Vision Laboratory
University of Oxford

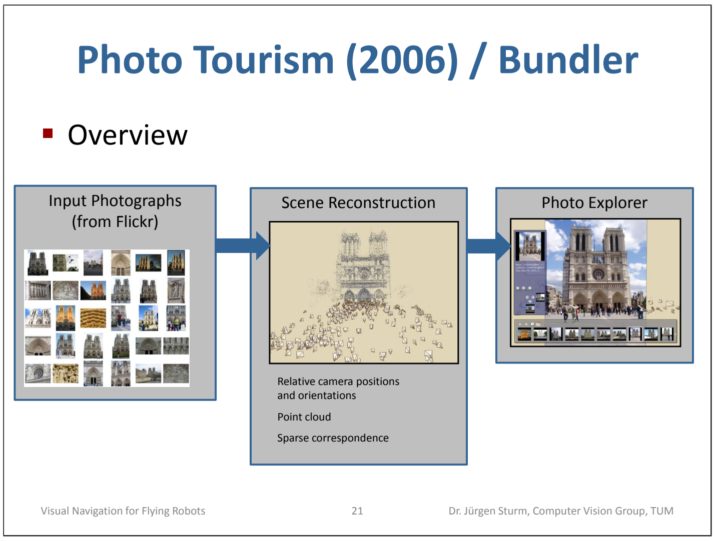


Photo Tourism (2006) / Bundler

- Overview

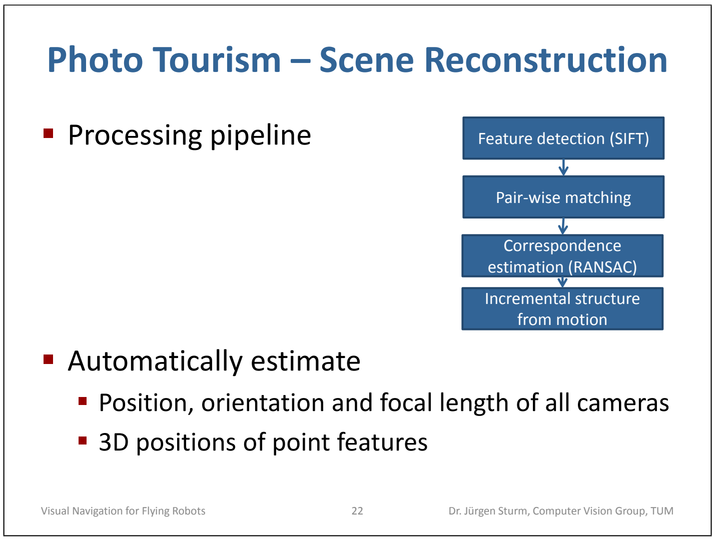


Photo Tourism – Scene Reconstruction

- Processing pipeline

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

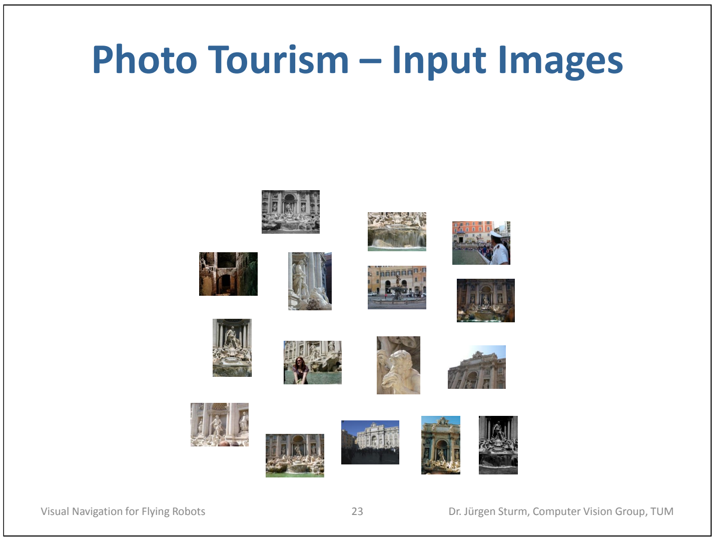


Photo Tourism – Input Images

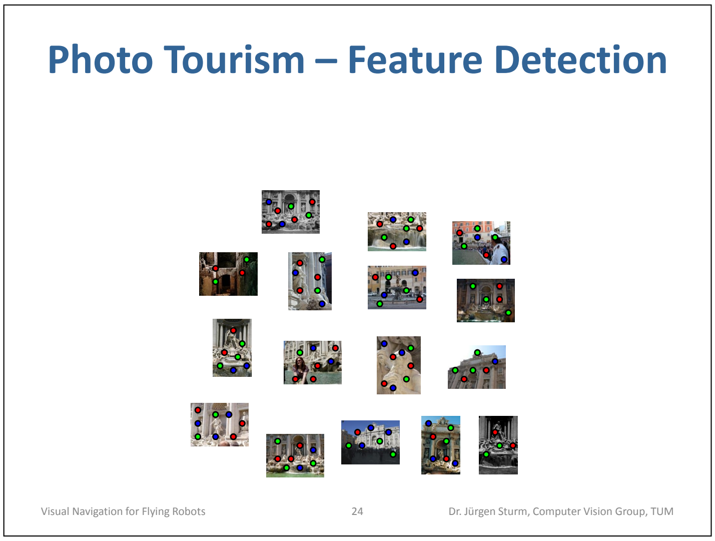


Photo Tourism – Feature Detection

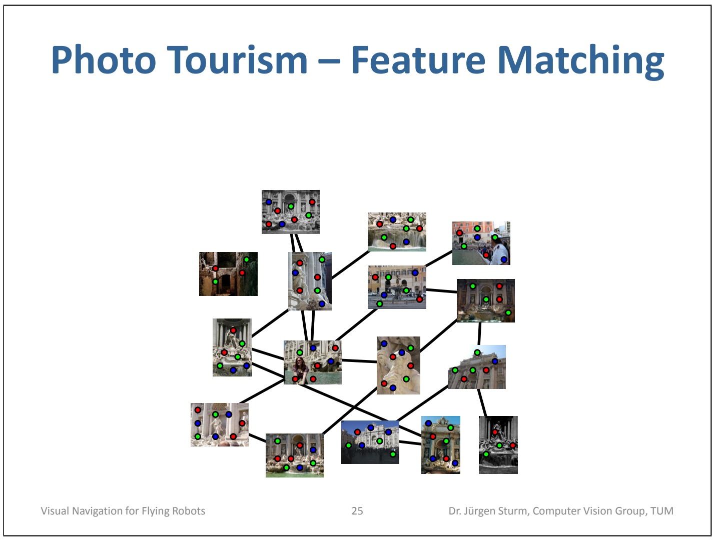
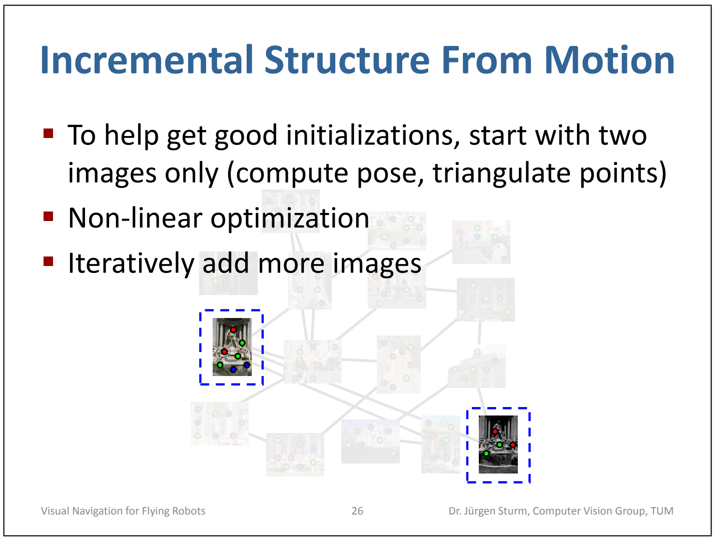


Photo Tourism – Feature Matching



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

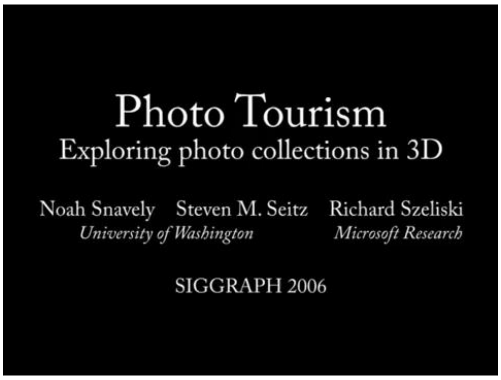


Photo Tourism

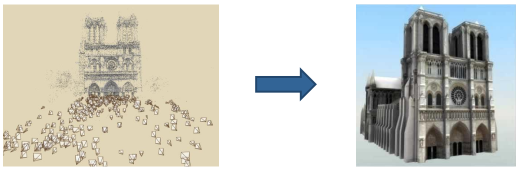
Exploring photo collections in 3D

Noah Snavely, University of Washington · Steven M. Seitz, University of Washington · Richard Szeliski, Microsoft Research

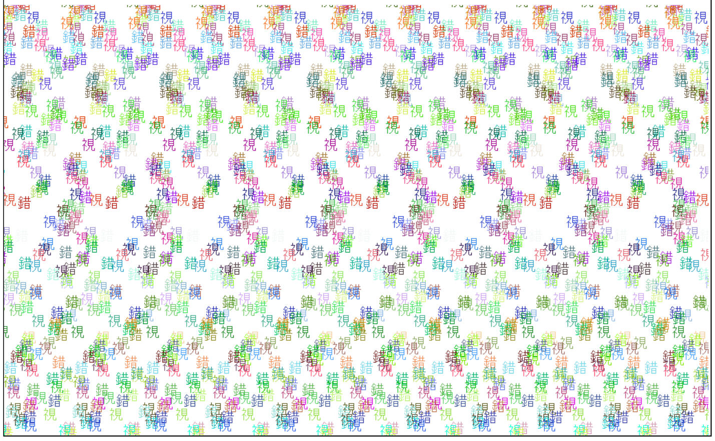
SIGGRAPH 2006

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?

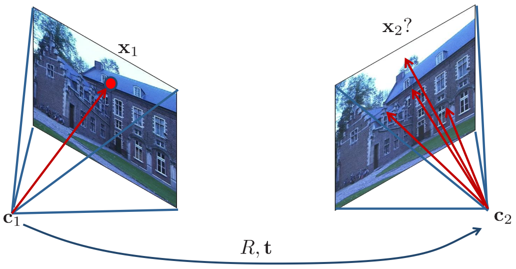


Human Stereo Vision



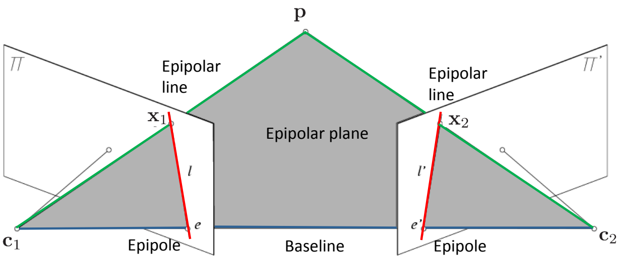
Stereo Correspondence Constraints

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

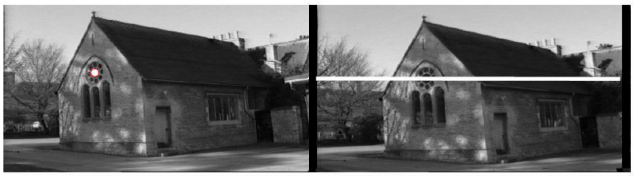


Reminder: Epipolar Geometry

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

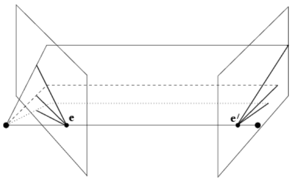


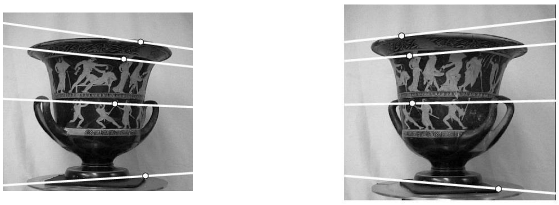
Epipolar Constraint



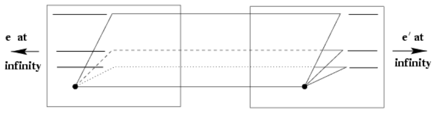
- This is useful because it reduces the correspondence problem to a 1D search along an epipolar line

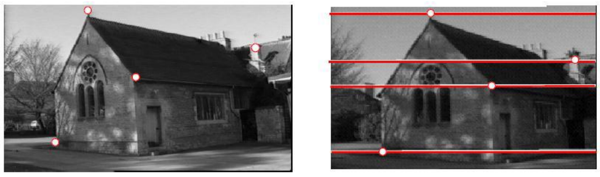
Example: Converging Cameras





Example: Parallel Cameras



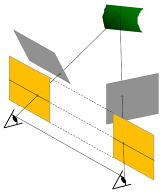


Rectification

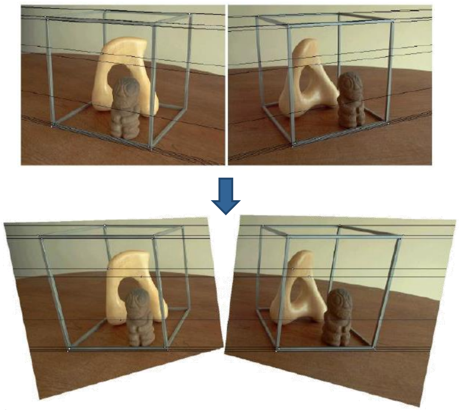
- In practice, it is convenient if the image scanlines (rows) are the epipolar lines

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

- Afterwards pixel motion is horizontal

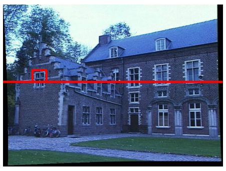


Example: Rectification



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)

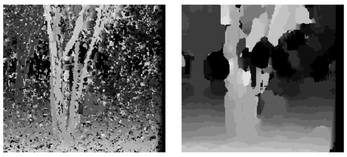
Example

- Input

- Output

What is the Influence of the Block Size?

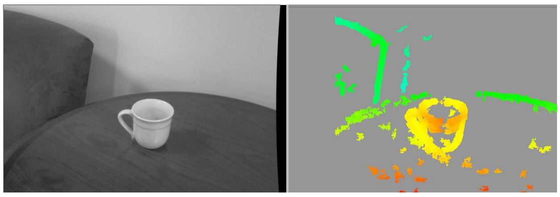
- Common choices are 5x5 .. 11x11
- Smaller neighborhood: more details
- Larger neighborhood: less noise
- Suppress pixels with low confidence (e.g., check ratio best match vs. 2nd best match)



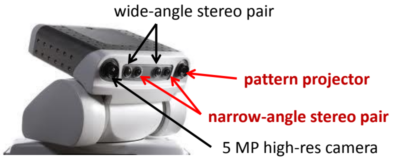
3x3 20x20

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

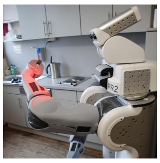


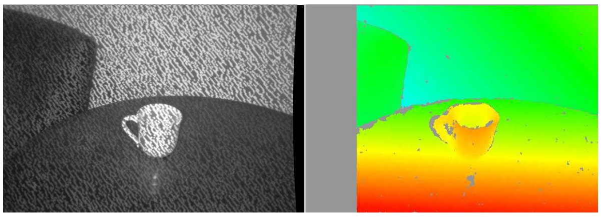
wide-angle stereo pair

pattern projector

narrow-angle stereo pair

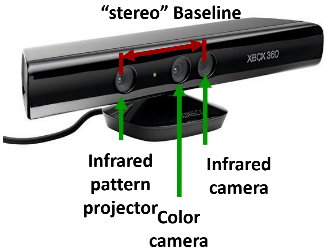
5 MP high-res camera





Sensor Principle of Kinect

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



"stereo" Baseline

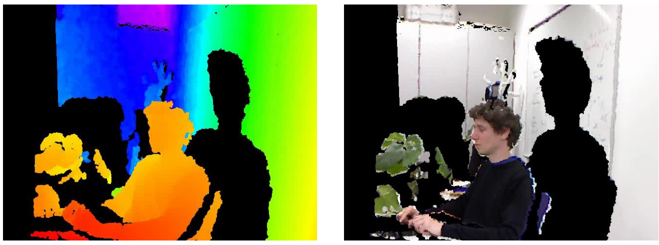
Infrared pattern projector

Color camera

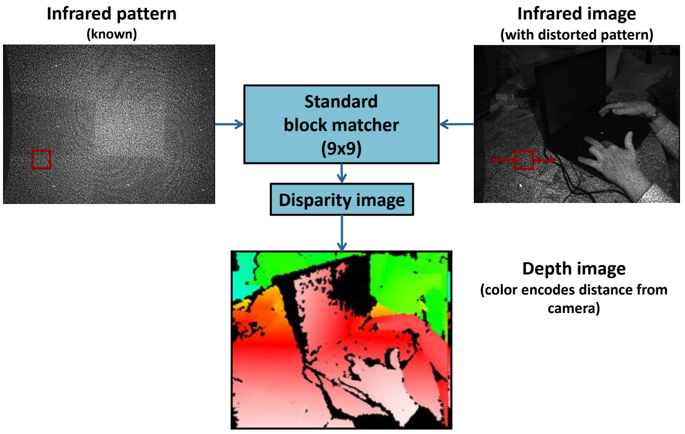
Infrared camera

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

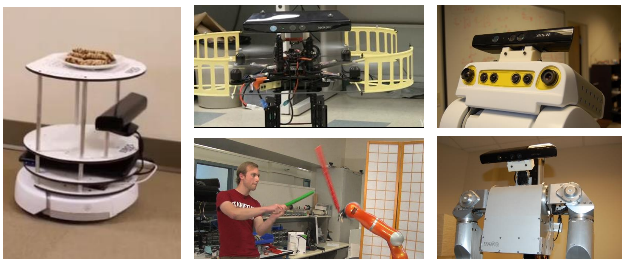
- Pattern is memorized at a known depth
- For each pixel in the IR image
  - Extract 9x9 template from memorized pattern
  - Correlate with current IR image over 64 pixels and search for the maximum
  - Interpolate maximum to obtain sub-pixel accuracy (1/8 pixel)
  - Calculate depth by triangulation

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

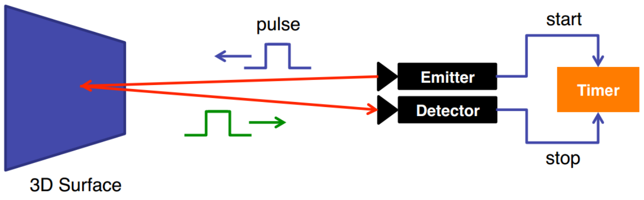
Impact of the Kinect Sensor

- Sold >18M units
- Has become a “standard” sensor in robotics
- Several variants (Asus Xtion Pro, PrimeSense)



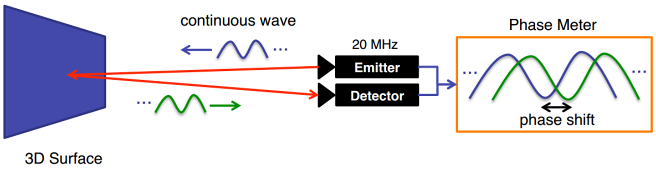
Time-of-Flight Cameras

- Direct time-of-flight measurement
  - Emit short light pulse (flash)
  - Every pixel counts time until signal is detected



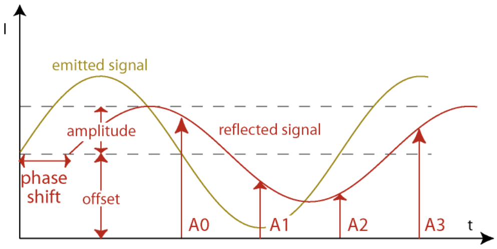
Time-of-Flight Cameras

- Indirect measurement (phase shift)
  - Emit modulated light (e.g., at 30 MHz → 10m wave length)
  - Every pixel measures the phase shift



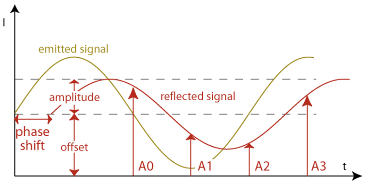
Decoding the Phase

- Take four intensity measurements at 90° angle
- Integrate over several waves to reduce noise
- Decode amplitude, offset, phase shift



Decoding the Phase

- Amplitude (=quality) $A = \frac{1}{2}\sqrt{(A_3 - A_1)^2 + (A_2 - A_0)^2}$
- Offset (=intensity) $B = A_0 + A_1 + A_2 + A_3$
- Phase shift (=distance) $\phi = \arctan \frac{A_3 - A_1}{A_2 - A_0}$



Commercial Time-Of-Flight Sensors

- Mesa SwissRanger ($4300), 176x144
- PMDTec: 200x200
- Intel Creative Camera ($150), 320x240
- Xbox One Kinect, 512x424, 13-bit depth

Xbox Kinect One

Intel Perceptual Computing Challenge

- https://perceptualchallenge.intel.com/
- API for raw data + hand gesture recognition
- This Saturday (22.6.2013): Hacknight
  - Every participating team gets a sensor for free
  - WERK1, Kultfabrik, Grafinger Str. 6, 81671 München
  - Sign up on Facebook page: http://www.facebook.com/groups/154028434769715/
- Cash Prizes for the Hacknight
  - €2.500 for the Best app
  - €1.000 for the Most Innovative app
  - €1.000 for the Best User Experience app

Project with Ardrones?

Perceptual Programming SDK controlled Parrot AR.Drone

an idea by Thomas Endres and Martin Förtsch

- control your Parrot AR.Drone using
  - … gestures to take-off and land
  - … your face and your bare hands to steer the drone in all directions
  - … speech for special flight animations and commands
- fly races with friends without touching any controller
- fly through obstacle courses and test your skilfulness
- AR.Drone video transmission and time-races included
- no 3D graphics card needed – it‘s real!

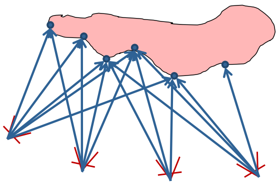
Pictures taken from:
http://software.intel.com/sites/default/files/styles/banner_carousel_983x400_forced/public/landing_page_banners/IDZ_UB_perceptualComputing.jpg?itok=ZWUVL0n0
http://wallpaperbus.net/wp-content/uploads/2012/07/Hyperspace_3D_Tunnel_Blue.jpg
http://www.assodigitale.it/wp-content/uploads/2010/01/Parrot_AR.Drone_09.jpg

Agenda for Today

- Bundle adjustment ✓
- Depth cameras ✓
- Occupancy grid maps
- Signed distance functions

Mapping and 3D Reconstruction

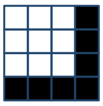
- So far: We have camera poses and 3D points



- Robot needs a map for:
  - Path planning and collision-free navigation
  - Exploration of unmapped areas
- How can we estimate such a map?

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

$$\text{Bel}(m) = P(m \mid z_1, \dots, z_n)$$

- How can this be computed?

Binary Bayes Filter

- **Goal:** Estimate

$$\text{Bel}(m) = P(m \mid z_1, \dots, z_n)$$

- How can this be computed?
- Using the (binary) 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

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

- 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

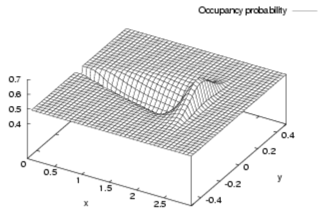
- For the Bayes filter, we need the inverse sensor model

$$p(m \mid z)$$

- Let’s consider an ultrasound sensor
  - Located at (0,0)
  - Measures distance of 2.5m
  - How does the inverse sensor model look like?

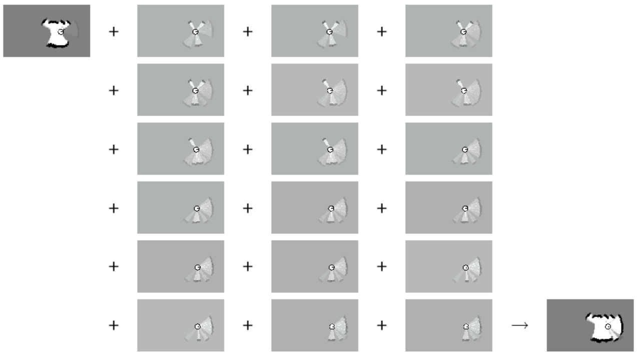
Typical Sensor Model for Ultrasound

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

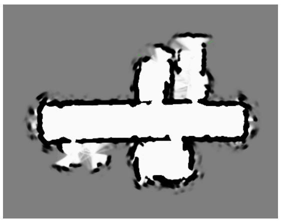


- Question: What about a laser scanner?

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?

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

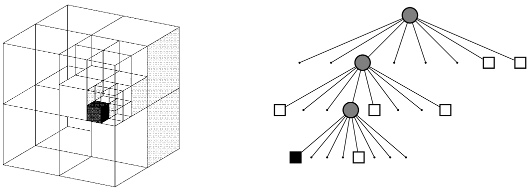
- And for 3D?

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

- And what about a whole city?

Map Representation by Octtrees

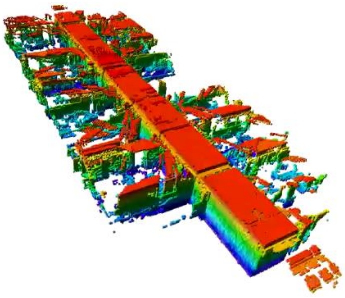
- Tree-based data structure
- Recursive subdivision of space into octants
- Volumes can be allocated as needed
- Multi-resolution



Example: OctoMap

[Wurm et al., 2011]

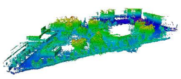
- Freiburg, building 79
  44 x 18 x 3 m^3, 0.05m resolution, 0.7mb on disk



Example: OctoMap

[Wurm et al., 2011]

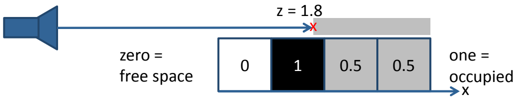
- Freiburg computer science campus
  292 x 167 x 28 m^3, 0.2m resolution, 2mb on disk



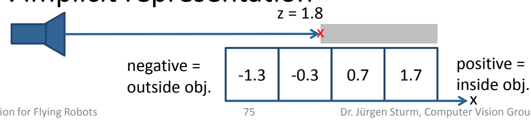
Signed Distance Field (SDF)

[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

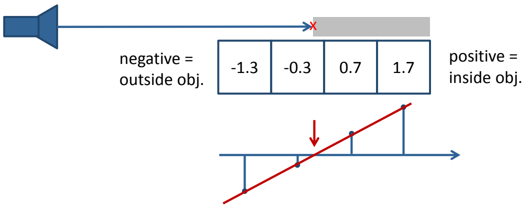


Signed Distance Field (SDF)

[Curless and Levoy, 1996]

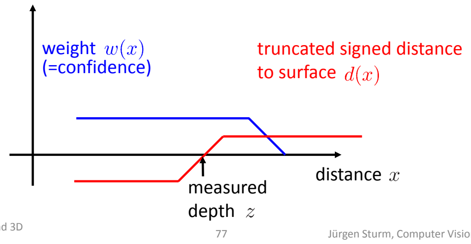
Algorithm:

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



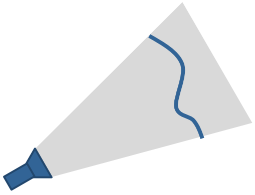
Distance and Weighting Functions

- Weight each observation according to its confidence
- Weight can additionally be influenced by other modalities (reflectance values, …)



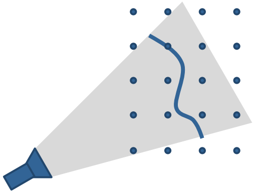
Dense Mapping: 2D Example

- Camera with known pose



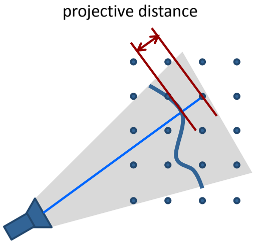
Dense Mapping: 2D Example

- Camera with known pose
- Grid with signed distance function



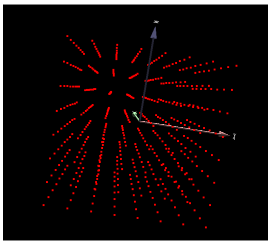
Dense Mapping: 2D Example

- For each grid cell, compute its projective distance to the surface



Dense Mapping: 3D Example

- Generalizes directly to 3D
- But: memory usage is cubic in side length



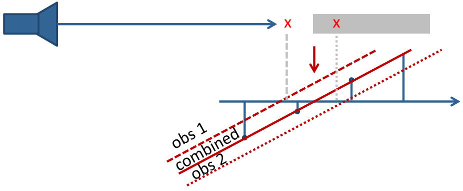
Data Fusion

- **Idea:** Compute weighted average
- 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: $x^* = \dfrac{\sum D_t(x)x}{\sum W_t(x)x}$

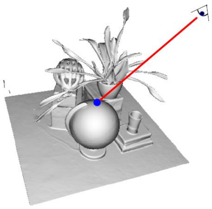
Visualizing Signed Distance Fields

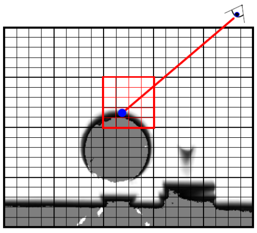
Common approaches to iso surface extraction:

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

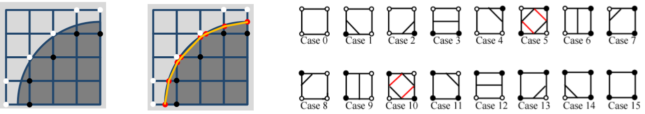




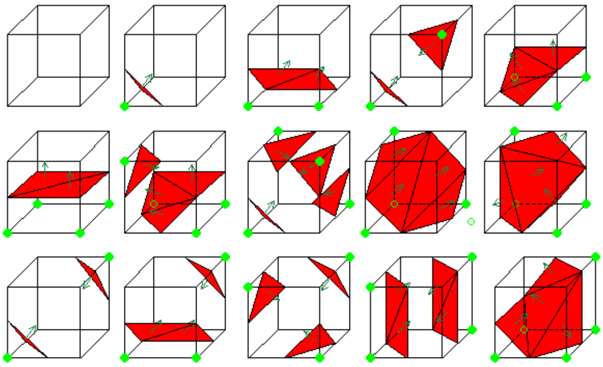
Marching Cubes

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

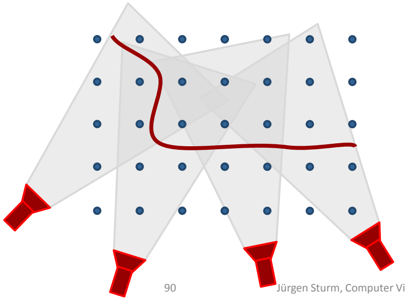
[Newcombe et al., 2011]

- Projective ICP with point-to-plane metric
- Truncated signed distance function (TSDF)
- Ray Casting

Dense Tracking: 2D Example

[Bylow et al., RSS 2013]

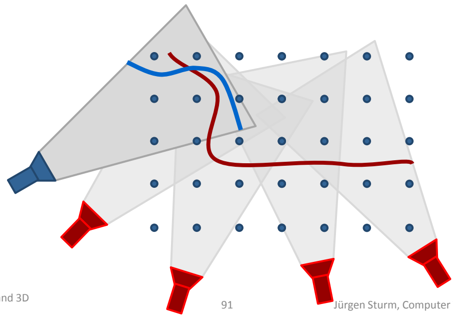
- 3D model built from the first k frames



Dense Tracking: 2D Example

[Bylow et al., RSS 2013]

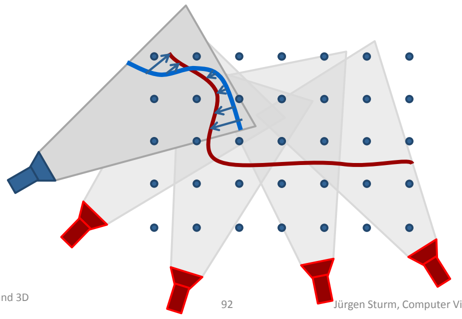
- Minimize distance between depth image and SDF



Dense Tracking: 2D Example

[Bylow et al., RSS 2013]

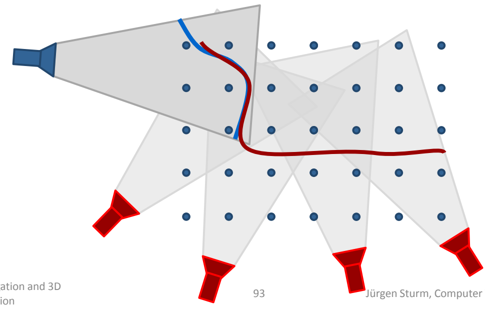
- Minimize distance between depth image and SDF



Dense Tracking: 2D Example

[Bylow et al., RSS 2013]

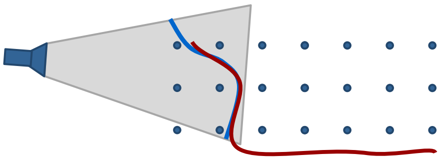
- Minimize distance between depth image and SDF



Dense Tracking: 2D Example

[Bylow et al., RSS 2013]

- Minimize distance between depth image and SDF



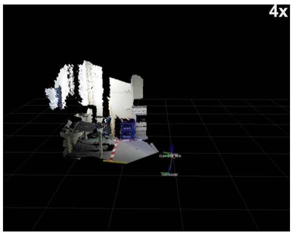
$$\arg\min_{\xi} E(\xi) = \arg\min_{\xi} \frac{1}{M} \sum_{ij} V(X(\xi,(i,j),I_d))^2$$

3D Reconstruction from a Quadrocopter

[Bylow et al., RSS 2013]

- AscTec Pelican quadrocopter
- Real-time 3D reconstruction, position tracking and control

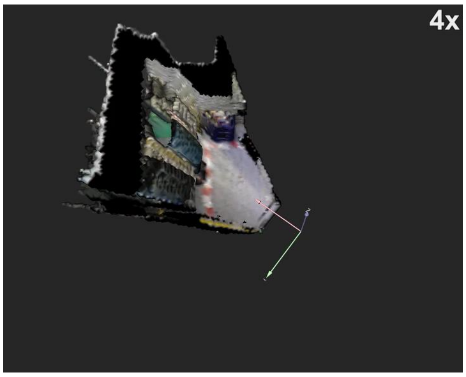
external view



estimated pose

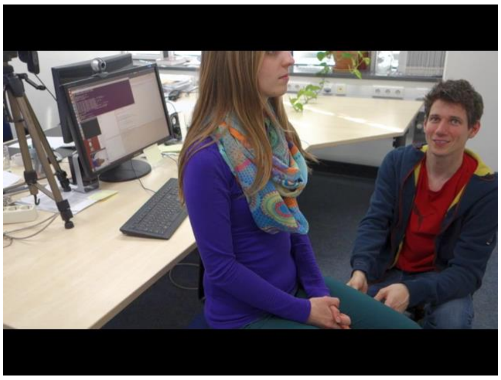
Resulting 3D Model

[Bylow et al., RSS 2013]

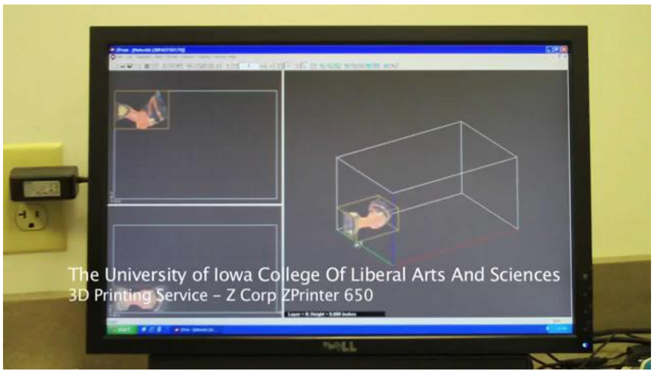


Let’s Scan a Person!

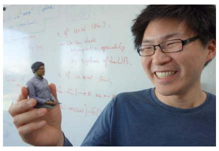
[Sturm et al., GCPR 2013]



3D Color Printing


The University of Iowa College Of Liberal Arts And Sciences
3D Printing Service – Z Corp ZPrinter 650

Can We Print These Models in 3D?



- Who wants to get a 3D scan of him/herself?

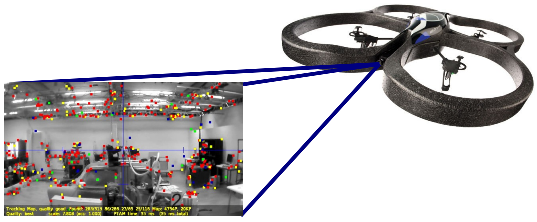
Lessons Learned Today

- How to estimate the camera poses and 3D points from monocular images using bundle adjustment
- How depth cameras work
- How to estimate occupancy maps
- What signed distance functions are
- How to reconstruct triangle meshes from SDFs

Computer Vision Group
Department of Computer Science
Technical University of Munich

Autonomous Navigation of Small-Scale Quadrocopters

Jakob Engel
VisNav invited talk, 25.06.2013



AR.Drone: Visual Navigation

How to make the AR.Drone fly more stable?

1. time delays
2. marker → PTAM

1. Time Delays

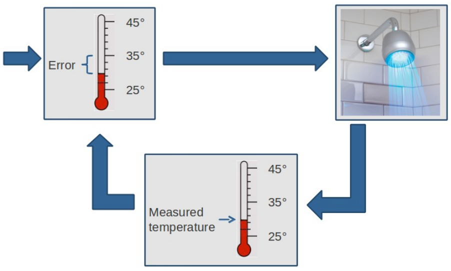
capture frame → send to PC → compute → send control

takes 150ms - 250ms

1. Time Delays

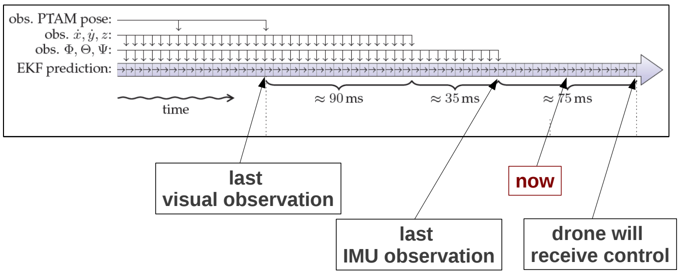
capture frame → send to PC → compute → send control

takes 150ms - 250ms



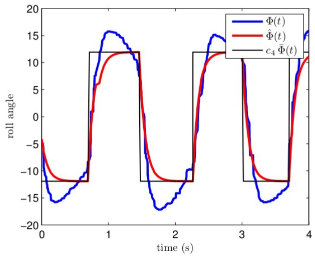
1. Time Delays

Solution: Explicitly model delays

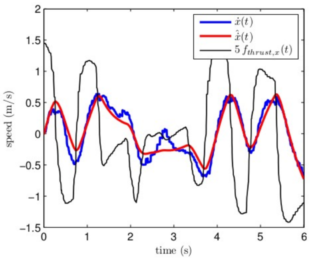


1. Time Delays

Solution: Model control & dynamics



control → attitude



attitude → velocity

→ **enables fully “blind” prediction**

Full EKF

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ z_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \\ \dot{z}_{t+1} \\ \Phi_{t+1} \\ \Theta_{t+1} \\ \Psi_{t+1} \\ \dot{\Psi}_{t+1} \end{pmatrix} \leftarrow \begin{pmatrix} x_t \\ y_t \\ z_t \\ \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \\ \Phi_t \\ \Theta_t \\ \Psi_t \\ \dot{\Psi}_t \end{pmatrix} + \delta_t \begin{pmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \\ \ddot{x}(\mathbf{x}_t) \\ \ddot{y}(\mathbf{x}_t) \\ \ddot{z}(\mathbf{x}_t,\mathbf{u}_t) \\ \dot{\Phi}(\mathbf{x}_t,\mathbf{u}_t) \\ \dot{\Theta}(\mathbf{x}_t,\mathbf{u}_t) \\ \dot{\Psi}_t \\ \ddot{\Psi}(\mathbf{x}_t,\mathbf{u}_t) \end{pmatrix}$$

$$\dot{\Phi}(\mathbf{x}_t,\mathbf{u}_t) = c_3\,\bar{\Phi}_t - c_4\,\Phi_t$$
$$\dot{\Theta}(\mathbf{x}_t,\mathbf{u}_t) = c_3\,\bar{\Theta}_t - c_4\,\Theta_t$$
$$\ddot{\Psi}(\mathbf{x}_t,\mathbf{u}_t) = c_5\,\bar{\dot{\Psi}}_t - c_6\,\dot{\Psi}_t$$
$$\ddot{z}(\mathbf{x}_t,\mathbf{u}_t) = c_7\,\bar{\dot{z}}_t - c_8\,\dot{z}_t$$
$$\ddot{x}(\mathbf{x}_t) = c_1\,(\cos\Psi_t\sin\Phi_t\cos\Theta_t - \sin\Psi_t\sin\Theta_t) - c_2\,\dot{x}_t$$
$$\ddot{y}(\mathbf{x}_t) = c_1\,(-\sin\Psi_t\sin\Phi_t\cos\Theta_t - \cos\Psi_t\sin\Theta_t) - c_2\,\dot{y}_t$$

EKF Prediction

$$h_{\text{PTAM}}(\mathbf{x}) := (x, y, z, \Phi, \Theta, \Psi)^T \in \mathbb{R}^6$$
$$\mathbf{z}_{\text{PTAM}} := \log(\mathbf{E}_{\mathcal{DC}}\,\mathbf{E}_{\mathcal{C}}) \in \mathbb{R}^6$$

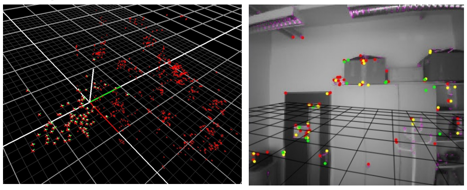
PTAM Observation

$$h_{\text{IMU}}(\mathbf{x}) := \begin{pmatrix} \cos(\Psi)\dot{x} - \sin(\Psi)\dot{y} \\ \sin(\Psi)\dot{x} + \cos(\Psi)\dot{y} \\ z \\ \Phi \\ \Theta \\ \Psi \end{pmatrix} \in \mathbb{R}^6$$
$$\mathbf{z}_{\text{IMU}} := \begin{pmatrix} \dot{x}_d \\ \dot{y}_d \\ z(t-\delta_t) + h(t) - h(t-\delta_t) \\ \hat{\Phi} \\ \hat{\Theta} \\ \Psi(t-\delta_t) + \hat{\Psi}(t) - \hat{\Psi}(t-\delta_t) \end{pmatrix} \in \mathbb{R}^6$$

IMU, altimeter, velocity Observation

2. PTAM

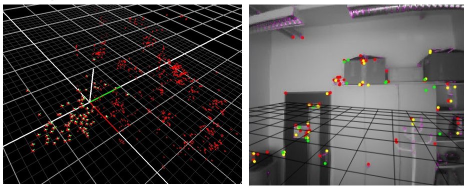
Parallel Tracking and Mapping [Murray '07]:
→ keyframe based, monocular SLAM system
→ open-source



Problems: Unreliable, no scale

2. PTAM

Parallel Tracking and Mapping [Murray '07]:
→ keyframe based, monocular SLAM system
→ open-source

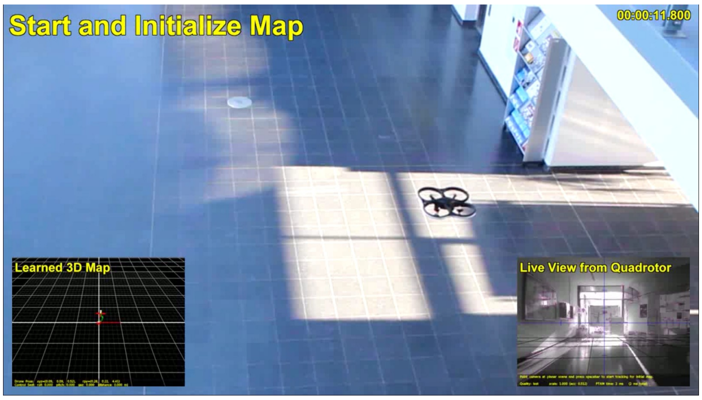


Problems: Unreliable, no scale

→ **enhance reliability** by incorporating IMU data

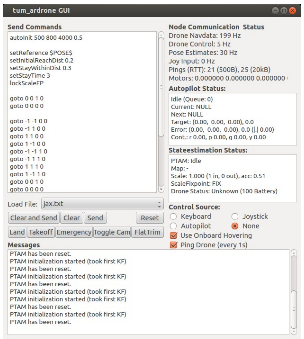
→ **add scale-estimation** from altimeter

2. PTAM



2. PTAM

tum_ardrone ROS package



Send Commands

```
autoInit 500 800 4000 0.5

setReference $POSE$
setInitialReachDist 0.2
setStayWithinDist 0.3
setStayTime 3
lockScaleFP

goto 0 0 1 0
goto 0 0 0 0

goto -1 -1 0 0
goto -1 1 0 0
goto 1 1 0 0
goto 1 -1 0 0
goto -1 -1 1 0
goto -1 1 1 0
goto 1 1 1 0
goto 1 -1 1 0
goto 0 0 1 0
goto 0 0 0 0
```

Load File: jax.txt

Node Communication Status
Drone Navdata: 199 Hz
Drone Control: 5 Hz
Pose Estimates: 30 Hz
Joy Input: 0 Hz
Pings (RTT): 21 (500B), 25 (20kB)
Motors: 0.000000 0.000000 0.000000

Autopilot Status:
Idle (Queue: 0)
Current: NULL
Next: NULL
Target: (0.00, 0.00, 0.00), 0.0
Error: (0.00, 0.00, 0.00), 0.0 (|.| 0.00)
Cont.: r 0.00, p 0.00, g 0.00, y 0.00

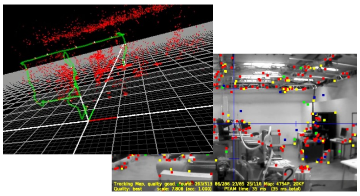
Stateestimation Status:
PTAM: Idle
Map: -
Scale: 1.000 (1 in, 0 out), acc: 0.51
ScaleFixpoint: FIX
Drone Status: Unknown (100 Battery)

Control Source:
○ Keyboard ○ Joystick
○ Autopilot ● None
☑ Use Onboard Hovering
☑ Ping Drone (every 1s)

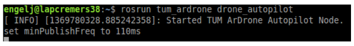
Messages

```
PTAM has been reset.
PTAM initialization started (took first KF)
PTAM has been reset.
PTAM initialization started (took first KF)
PTAM has been reset.
PTAM initialization started (took first KF)
PTAM initialization started (took first KF)
PTAM has been reset.
PTAM initialization started (took first KF)
PTAM has been reset.
```

Node 1: GUI + backup control


Tracking Map, quality good. Found: 263/513 86/286 23/85 25/116 Map: 4754P, 20KF
Quality: best scale: 7.808 (acc: 1.000) PTAM time: 35 ms (35 ms total)

Node 2: State-estimation (SLAM, EKF)



```
engelj@lapcremers38:~$ rosrun tum_ardrone drone_autopilot
[ INFO] [1369780328.885242358]: Started TUM ArDrone Autopilot Node.
set minPublishFreq to 110ms
```

Node 3: Autopilot (PID Controller)

Open Source @ www.ros.org/wiki/tum_ardrone

Future Work

- **Increase Range:**
  Augment PTAM with eg. FABMAP and/or g2o
- **Initialization:**
  Faster & more robust using gyro readings;
  Automatic re-initialization
- **Performance (→onboard):**
  e.g. only 3DoF coarse tracking + gyro.

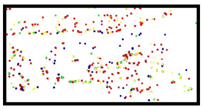
Future Work

- **Add feature**
  e.g. person following, gesture recognition, …
- **Obstacle Avoidance**
  e.g. using optical flow.
- …

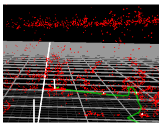
Preview

Monocular SLAM without keypoints

Image (pixels)



Features (e.g. point-positions)

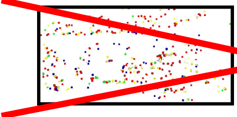


Mapping, Tracking

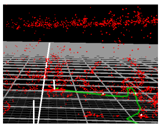
Preview

Monocular SLAM without keypoints

Image (pixels)

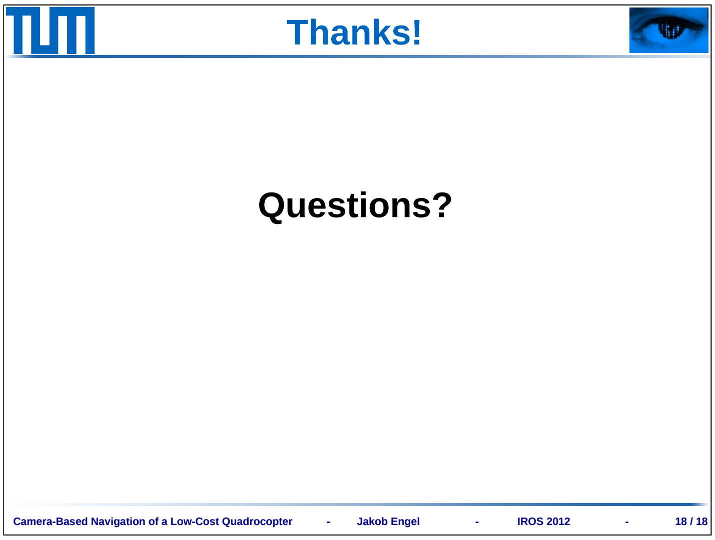


~~**Features (e.g. point-positions)**~~



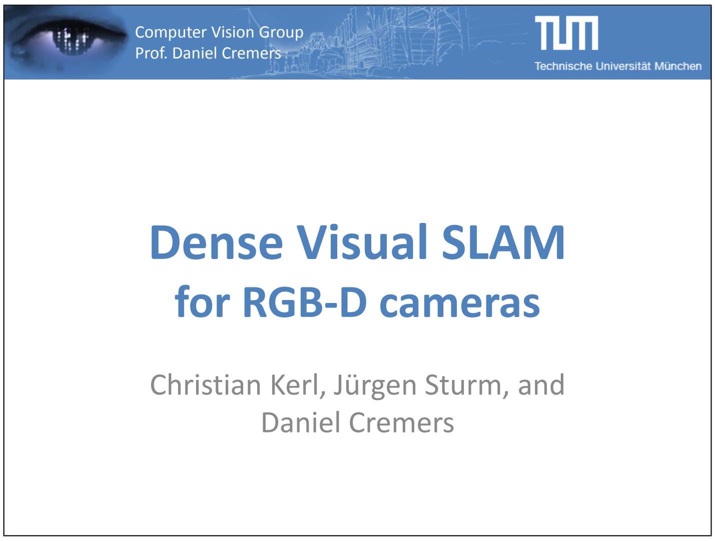
Mapping, Tracking

Preview



Thanks!

Questions?



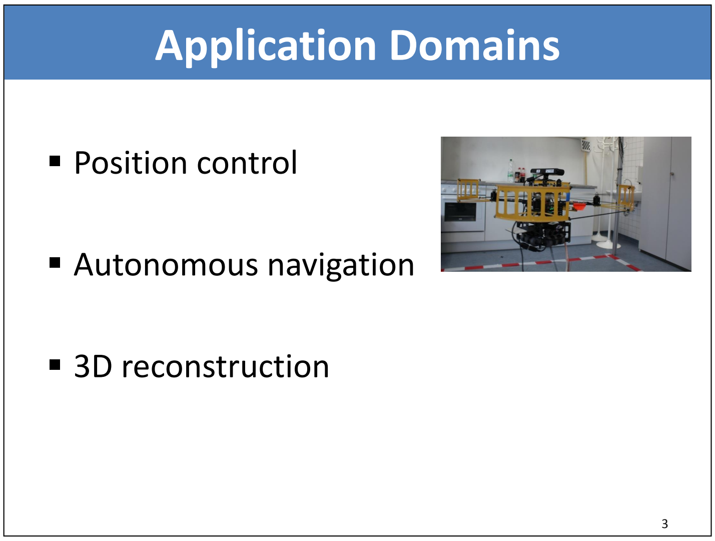
Computer Vision Group
Prof. Daniel Cremers
Technische Universität München

Dense Visual SLAM for RGB-D cameras

Christian Kerl, Jürgen Sturm, and Daniel Cremers

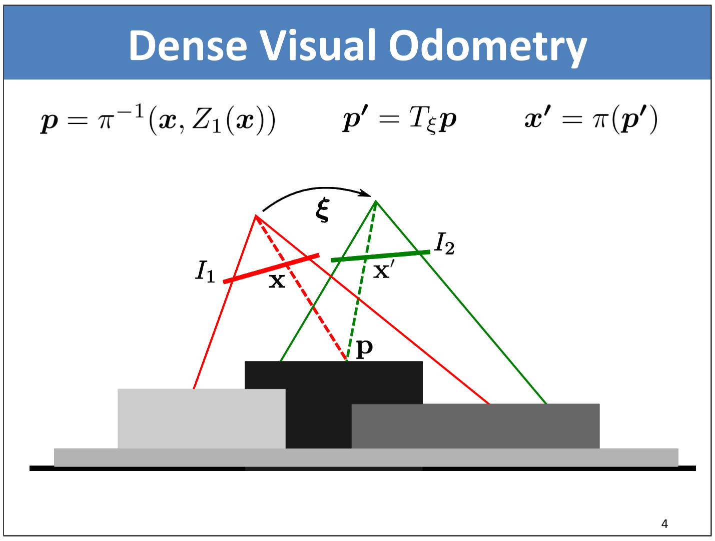
Goal

Estimate camera motion from RGB-D data



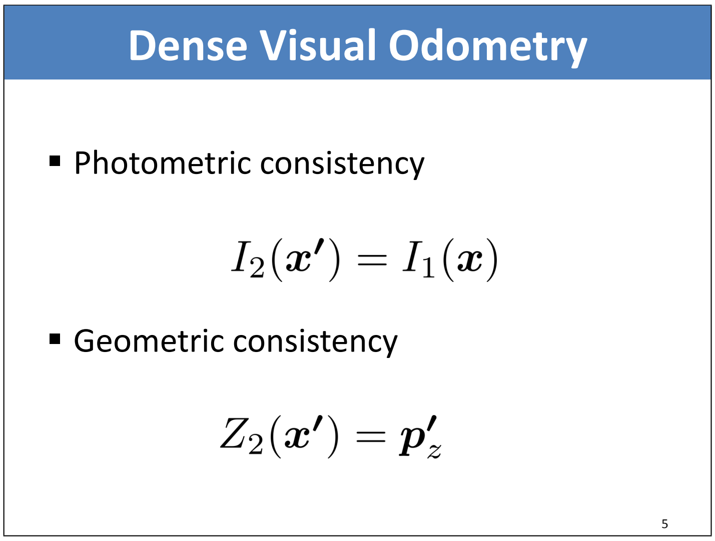
Application Domains

- Position control
- Autonomous navigation
- 3D reconstruction



Dense Visual Odometry

$$\boldsymbol{p} = \pi^{-1}(\boldsymbol{x}, Z_1(\boldsymbol{x})) \qquad \boldsymbol{p}' = T_\xi \boldsymbol{p} \qquad \boldsymbol{x}' = \pi(\boldsymbol{p}')$$



Dense Visual Odometry

- Photometric consistency

$$I_2(\boldsymbol{x}') = I_1(\boldsymbol{x})$$

- Geometric consistency

$$Z_2(\boldsymbol{x}') = \boldsymbol{p}'_z$$

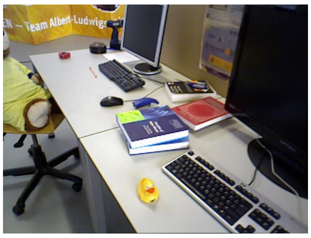
Dense Visual Odometry

- Least squares formulation

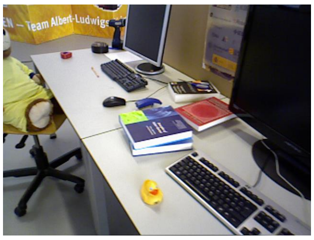
$$\mathbf{e} = \begin{pmatrix} e_I \\ e_Z \end{pmatrix} = \begin{pmatrix} I_2(\boldsymbol{x}') - I_1(\boldsymbol{x}) \\ Z_2(\boldsymbol{x}') - \boldsymbol{p}'_z \end{pmatrix}$$

$$\xi^* = \arg\min_{\xi} \sum_i^n \mathbf{e}_i^\top \Sigma^{-1} \mathbf{e}_i$$

Dense Visual Odometry



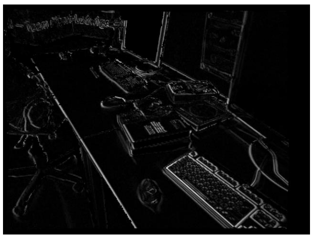
I_1



I_2

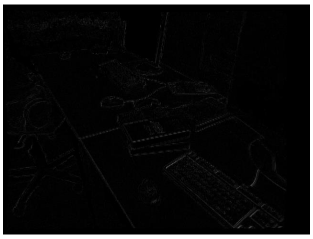
Dense Visual Odometry

Residuals before registration



$\left(I_2(\boldsymbol{x}') - I_1(\boldsymbol{x})\right)^2 \quad \xi = 0$

Residuals after registration



$\left(I_2(\boldsymbol{x}') - I_1(\boldsymbol{x})\right)^2 \quad \xi = \xi^*$

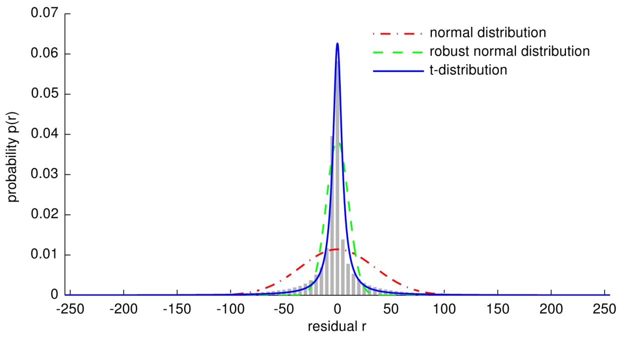
Robust Dense Visual Odometry

- Outliers violate consistency assumption
  - » Moving objects
  - » Non-lambertian surfaces
  - » Noise

- Problem: Quadratic term gives high influence

Robust Dense Visual Odometry

$I_2(\boldsymbol{x}') - I_1(\boldsymbol{x})$



Robust Dense Visual Odometry

- Weighted least squares formulation

$$\xi^* = \arg\min_{\xi} \sum_i^n \boxed{w_i} \mathbf{e}_i^\top \Sigma^{-1} \mathbf{e}_i$$

$$w_i(\mathbf{e}_i) = \frac{\nu + 1}{\nu + \mathbf{e}_i^\top \Sigma^{-1} \mathbf{e}_i}$$

Visual Odometry Results

Visual Odometry Results

- Frame-to-frame motion estimation
  - » Fast
  - » Highly accurate
  - » Drift 0.03 m/s
- Problem: drift accumulation (1.8 m/min)

Dense Visual SLAM

- Local drift

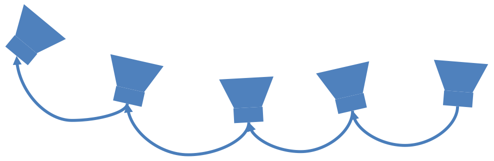
Keyframes

- Global drift

Pose graph optimization

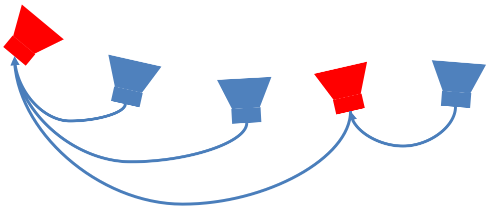
Keyframes

- Frame-to-frame



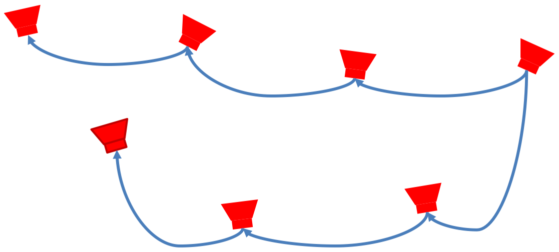
Keyframes

- Frame-to-keyframe



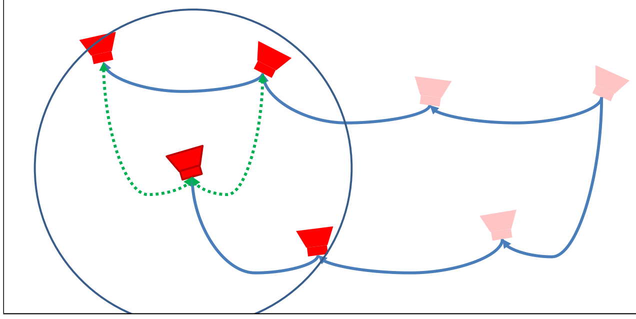
Pose Graph Optimization

- Correct global drift with loop closures



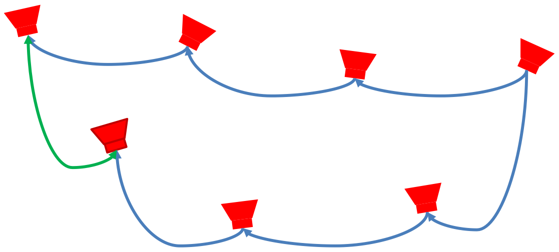
Pose Graph Optimization

- Search for loop closure candidates



Pose Graph Optimization

- Validate loop closure and update graph



Dense Visual SLAM

- How to select keyframes?

- How to validate loop closures?

Dense Visual SLAM

- Least squares yields estimate of covariance of ξ^*
- Compute entropy of parameter distribution as $H(\xi) = \ln(|\Sigma_\xi|)$
- $H(\xi)$ is a measure of uncertainty in estimate, i.e., quality

Visual SLAM Results

Master Thesis Topics

- Dense Visual SLAM for Quadrocopters
  - » Implement on AscTec Pelican
- Multi-Session Dense Visual SLAM
  - » Relocalization / place recognition
  - » Reduced pose graph
  - » Efficient map representation

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

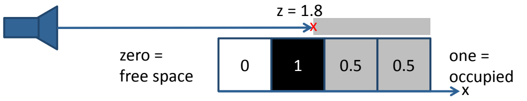
Dense Reconstruction in Sparse Data Structures

Frank Steinbrücker

Recap: 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

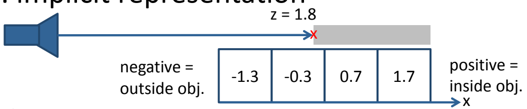


zero = free space

0	1	0.5	0.5

one = occupied

- SDF: implicit representation



negative = outside obj.

-1.3	-0.3	0.7	1.7

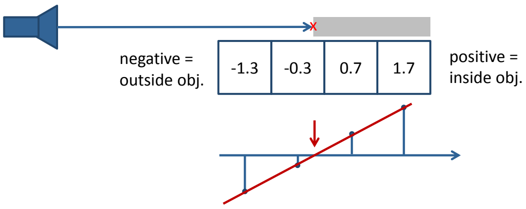
positive = inside obj.

Recap: 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)



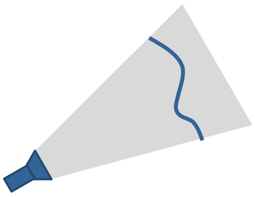
negative = outside obj.

-1.3	-0.3	0.7	1.7

positive = inside obj.

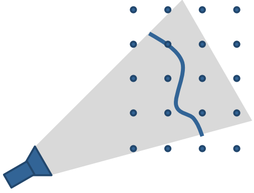
Recap: Dense Mapping: 2D Example

- Camera with known pose



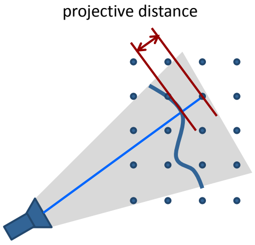
Recap: Dense Mapping: 2D Example

- Camera with known pose
- Grid with signed distance function



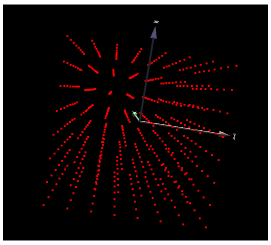
Recap: Dense Mapping: 2D Example

- For each grid cell, compute its projective distance to the surface



Recap: Dense Mapping: 3D Example

- Generalizes directly to 3D
- But: memory usage is cubic in side length



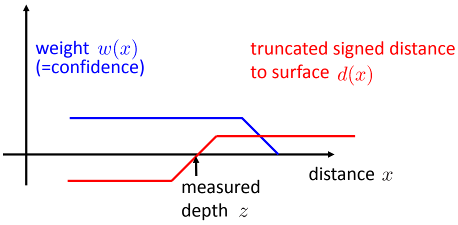
Recap: Data Fusion

- **Idea:** Compute weighted average
- 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

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

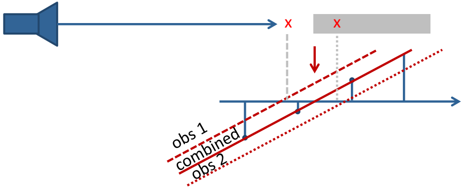
Recap: Distance and Weighting Functions

- Weight each observation according to its confidence
- Weight can additionally be influenced by other modalities (reflectance values, …)



Recap: Two Nice Properties

- Noise cancels out over multiple measurements



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

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

Recap: Memory Consumption

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

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

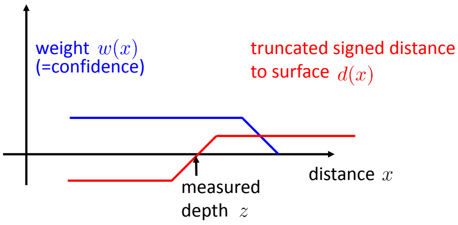
- And for 3D?

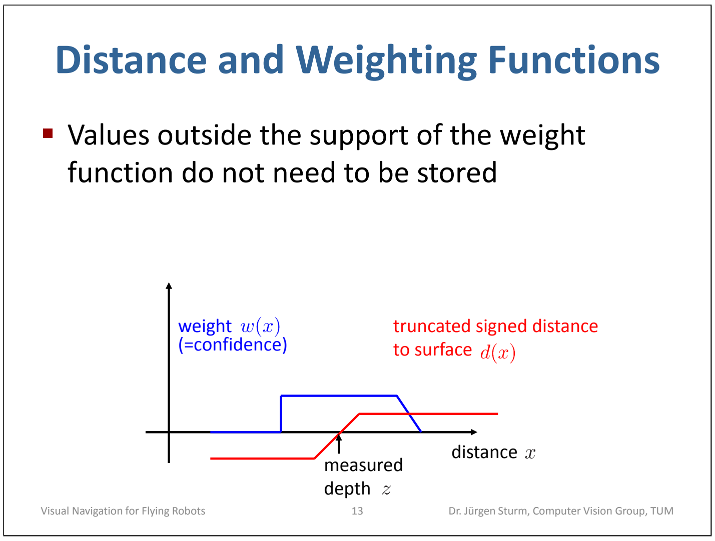
$$\left(\frac{40}{0.05}\right)^3 = 512.000.000 \text{ cells} = 3.8\text{gb}$$

- And what about a whole city?

Distance and Weighting Functions

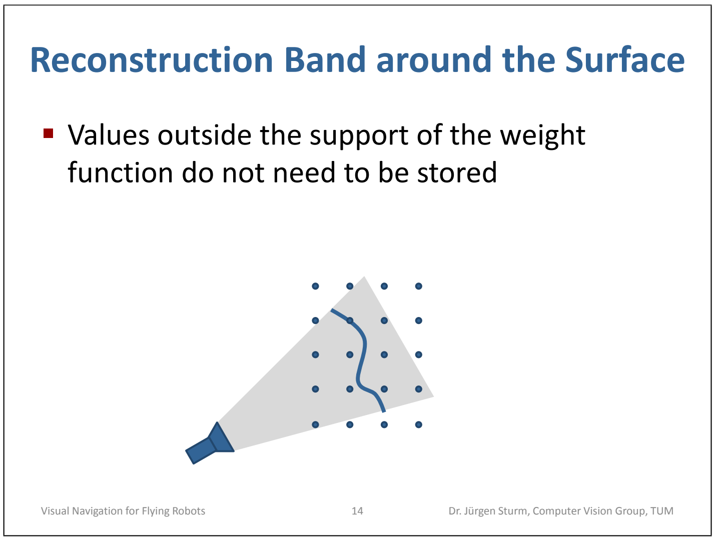
- Weight each observation according to its confidence
- Weight can additionally be influenced by other modalities (reflectance values, …)





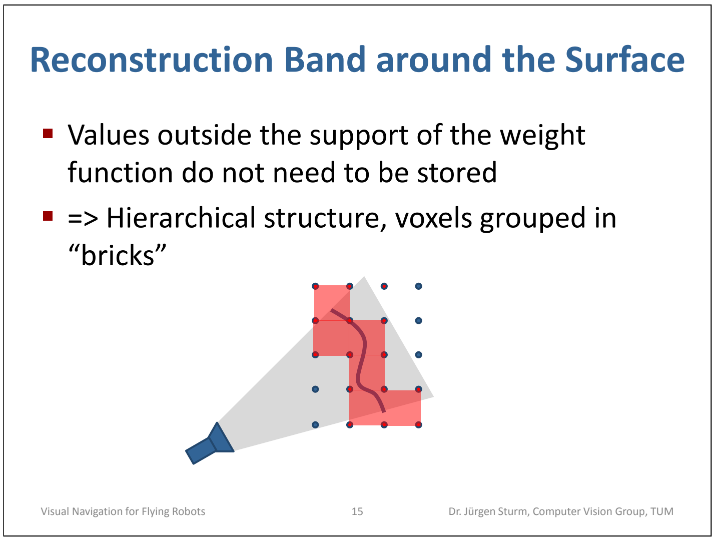
Distance and Weighting Functions

- Values outside the support of the weight function do not need to be stored



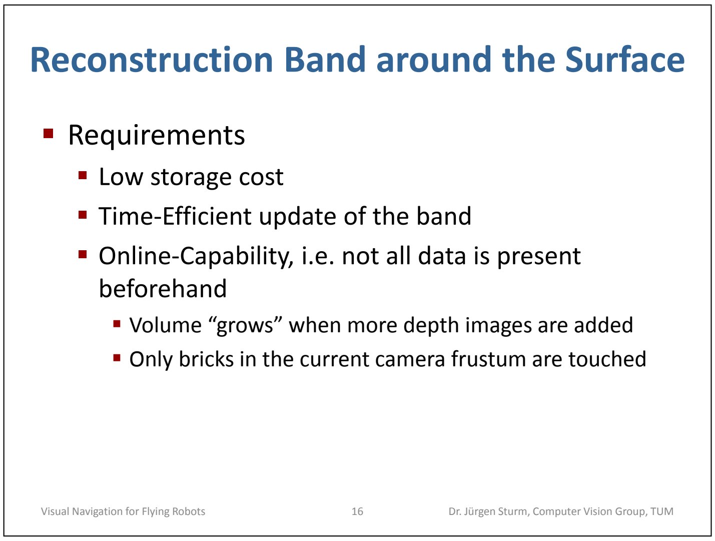
Reconstruction Band around the Surface

- Values outside the support of the weight function do not need to be stored



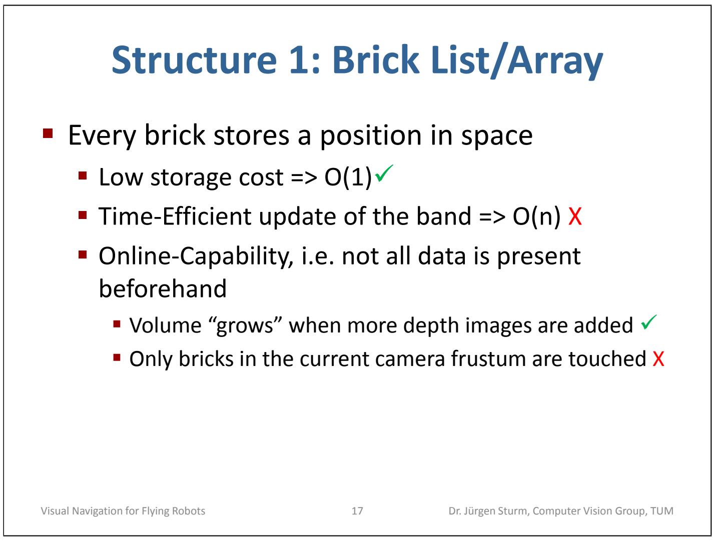
Reconstruction Band around the Surface

- Values outside the support of the weight function do not need to be stored
- => Hierarchical structure, voxels grouped in "bricks"



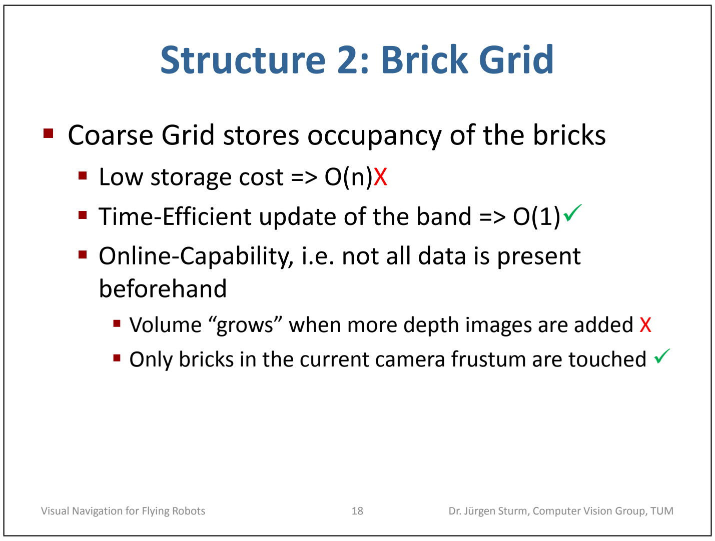
Reconstruction Band around the Surface

- Requirements
  - Low storage cost
  - Time-Efficient update of the band
  - Online-Capability, i.e. not all data is present beforehand
    - Volume "grows" when more depth images are added
    - Only bricks in the current camera frustum are touched



Structure 1: Brick List/Array

- Every brick stores a position in space
  - Low storage cost => O(1)✓
  - Time-Efficient update of the band => O(n) X
  - Online-Capability, i.e. not all data is present beforehand
    - Volume "grows" when more depth images are added ✓
    - Only bricks in the current camera frustum are touched X



Structure 2: Brick Grid

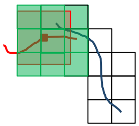
- Coarse Grid stores occupancy of the bricks
  - Low storage cost => O(n)X
  - Time-Efficient update of the band => O(1)✓
  - Online-Capability, i.e. not all data is present beforehand
    - Volume "grows" when more depth images are added X
    - Only bricks in the current camera frustum are touched ✓

Structure 3: Tree and List

- Brick List/Array with an Octree ontop
  - Low storage cost => O(log n) ✓
  - Time-Efficient update of the band => O(log n)✓
  - Online-Capability, i.e. not all data is present beforehand
    - Volume “grows” when more depth images are added ✓
    - Only bricks in the current camera frustum are touched ✓

Fusion step in the Octree

- (Parallel) iteration over all valid depth pixels
  - Bounding box around the 3D point according to distance/weight-threshold
    - Intersection of the bounding box with tree leaves
      - Allocation of new leaves, where necessary
    - Addition of the the intersected leaves to a queue
  - (Parallel) processing of the queue (projection und distance update)

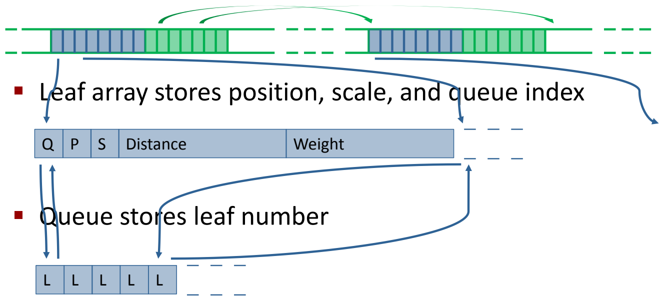


Implementation of the Tree

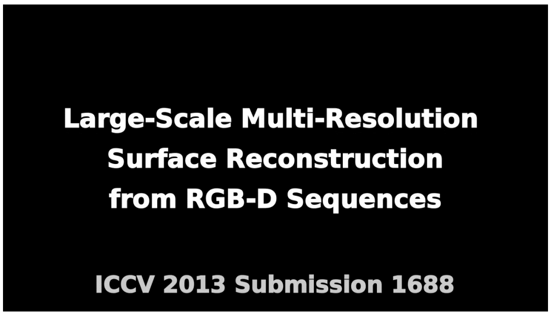
- CPU/GPU interoperability requirements
  - No explicit C-Pointers => Indices instead
  - All memory is allocated beforehand
    - Push-back operations are performed with global indices
  - Minimization of diverging code
    - => Reason for queue

Implementation of the Tree

- Branch array stores indices of leaves and subbranches
- Leaf array stores position, scale, and queue index
- Queue stores leaf number



Demo Video



Large-Scale Multi-Resolution Surface Reconstruction from RGB-D Sequences

ICCV 2013 Submission 1688

Computer Vision Group
Prof. Daniel Cremers

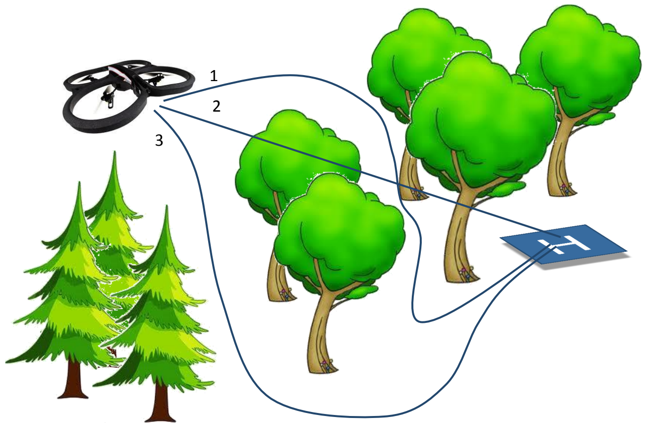
Technische Universität München

Visual Navigation for Flying Robots

Motion Planning

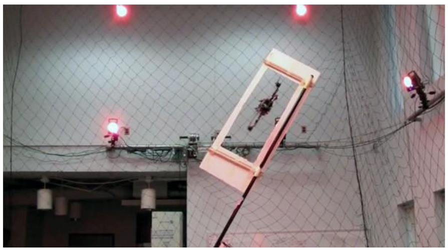
Dr. Jürgen Sturm

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?

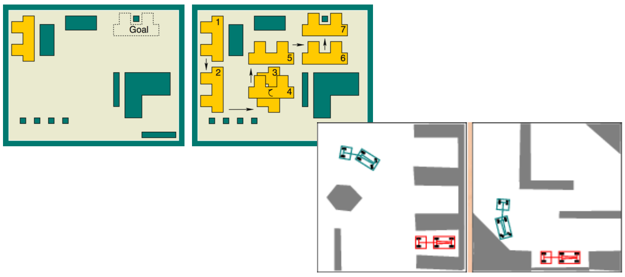
Motion Planning Problem

What are good performance metrics?

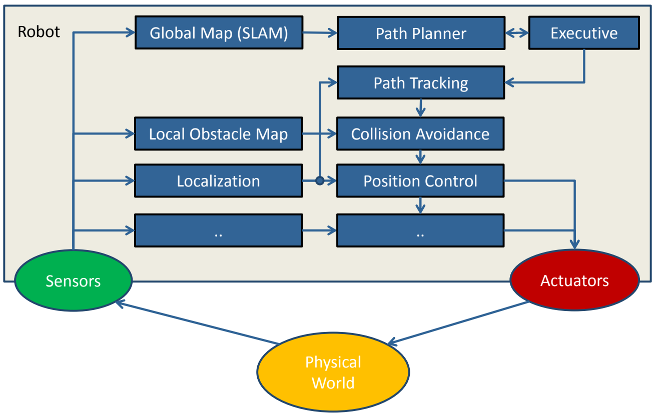
- Execution speed / path length
- Energy consumption
- Planning speed
- Safety (minimum distance to obstacles)
- Robustness against disturbances
- Probability of success
- …

Motion Planning Examples

Motion planning is sometimes also called the **piano mover's problem**



Robot Architecture



Robot

Global Map (SLAM)

Path Planner

Executive

Path Tracking

Local Obstacle Map

Collision Avoidance

Localization

Position Control

..

..

Sensors

Actuators

Physical World

Agenda for Today

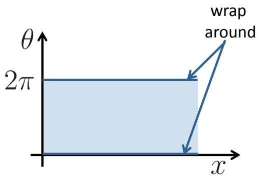
- Configuration spaces
- Roadmap construction
- Search algorithms
- Path optimization and re-planning
- Path execution

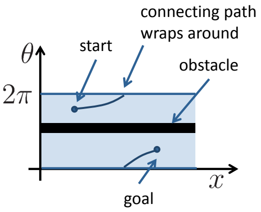
Configuration Space

- Work space
  - Position in 3D → 3 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





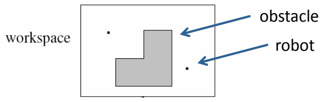
Notation

- Configuration space $C \subset \mathbb{R}^d$
- Configuration $\mathbf{q} \in C$
- Free space C_{free}
- Obstacle space C_{obs}
- Properties

$$C_{\text{free}} \cup C_{\text{obs}} = C$$
$$C_{\text{free}} \cap C_{\text{obs}} = \emptyset$$

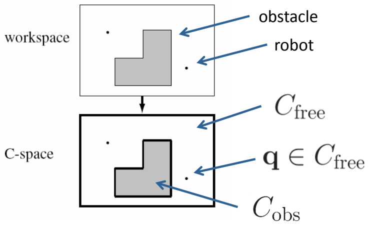
Free Space Example

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



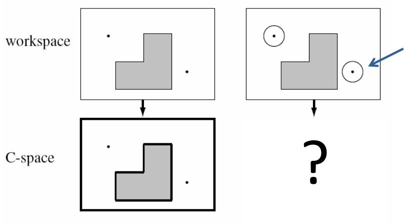
Example

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



Example

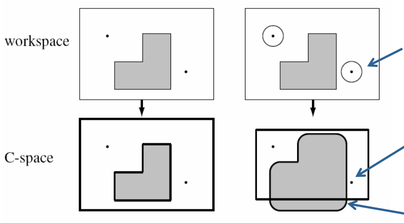
- What are admissible configurations for the robot? Equiv.: What is the free space?
- Circular robot



robot footprint

Example

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



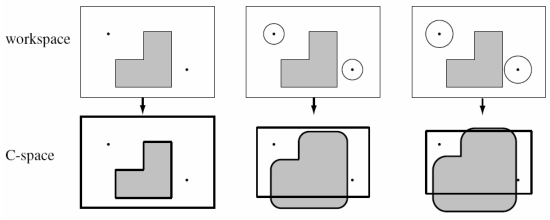
robot footprint in work space (disk)

robot footprint in configuration space (point)

obstacle in configuration space

Example

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



Computing the Free Space

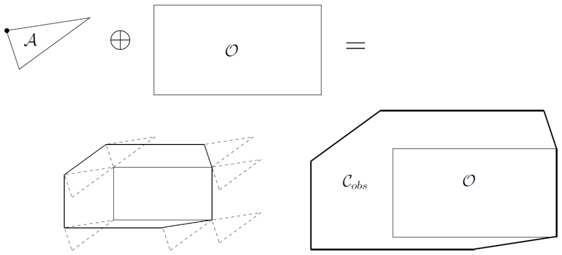
- Free configuration space is obtained by sliding the robot along the edge of the obstacle regions "blowing them up" by the robot radius
- This operation is called the **Minowski sum**

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

where $A, B \subset \mathbb{R}^d$

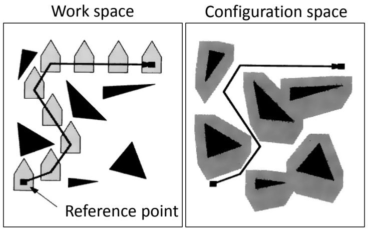
Example: Minowski Sum

- Triangular robot and rectangular obstacle



Example

- Polygonal robot, translation only


Work space

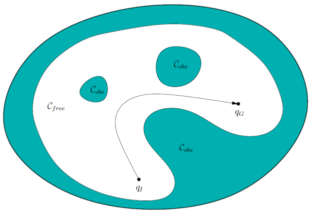
Configuration space

Reference point

- 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

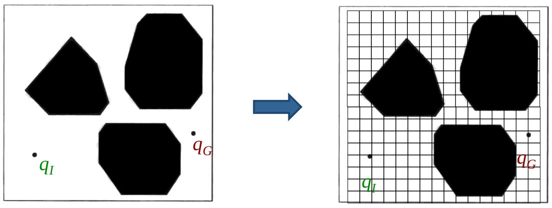
1. **C-Space discretization**
   (generating a graph / roadmap)
2. Search algorithm
   (Dijkstra’s algorithm, A*, …)
3. Re-planning
   (D*, …)
4. Path tracking
   (PID control, potential fields, funnels, …)

C-Space Discretizations

- **Combinatorial planning**
  - Find a solution when one exists (complete)
  - Require polygonal decomposition
  - Become quickly intractable for higher dimensions
- **Sampling-based planning**
  - Weaker guarantees but more efficient
  - Need only point-wise evaluations of C_{free}
  - We will have a look at:
    grid decomposition, road maps, random trees

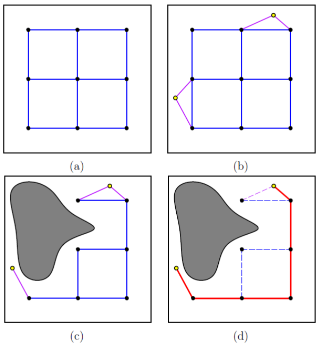
Grid Decomposition

- Construct a regular grid
- Determine status of every cell (free/occ)
- Simple, but not efficient (why?)
- Not exact (why?)



Grid Decomposition

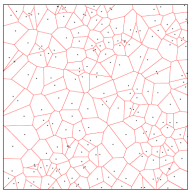
- Regular grid
- Construct graph
  - Grid cells as vertices
  - Edges encode traversability
- Query
  - Add start and goal to graph, connect to nearest neighbors
  - Perform graph search


(a) (b) (c) (d)

Probabilistic Roadmaps (PRMs)

[Kavraki et al., 1992]

- Grids do not scale well to high dimensions
- Sampling-based approach
- **Vertex:** Take random sample from C, check whether sample is in C_{free}



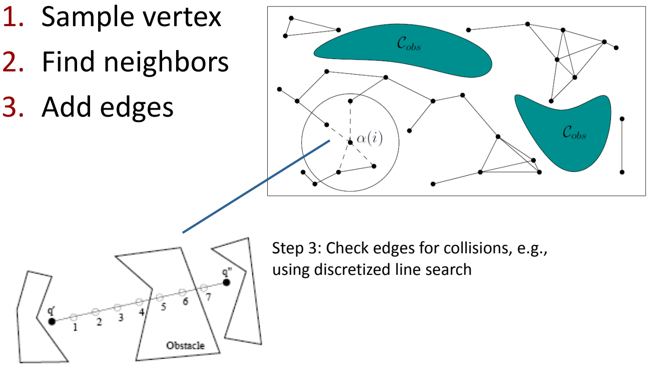
Probabilistic Roadmaps (PRMs)

[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

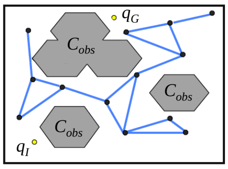


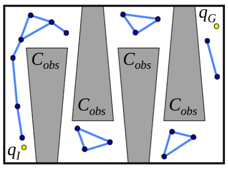
Step 3: Check edges for collisions, e.g., using discretized line search

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

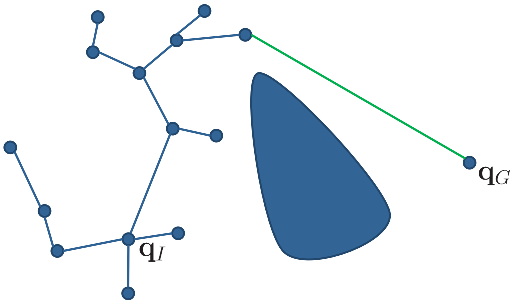




Rapidly Exploring Random Trees

[Lavalle and Kuffner, 1999]

- **Idea:** Grow a 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?

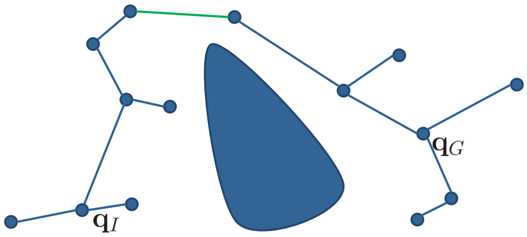
Rapidly Exploring Random Trees

- **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

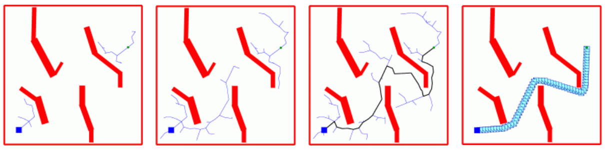
[Lavalle and Kuffner, 1999]

- **RRT:** Grow trees from start and goal location towards each other, stop when they connect

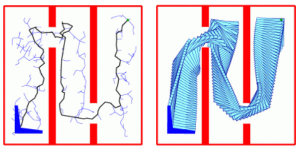


RRT Examples

- 2-DOF example

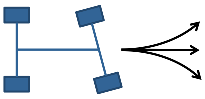


- 3-DOF example (2D translation + rotation)



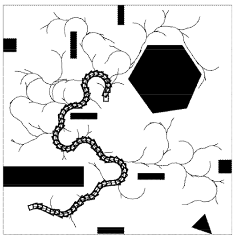
Non-Holonomic Robots

- 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

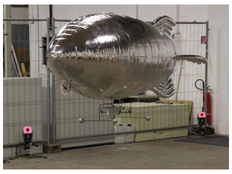


Example: Blimp Motion Planning

[Müller et al., IROS 2011]

Advantages

- Low power consumption
- Safe navigation capabilities



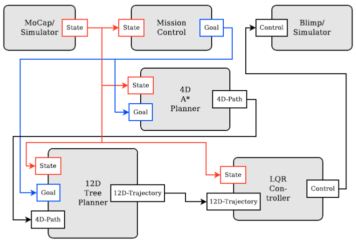
Challenges

- Seriously underactuated (only 3-DOF control)
- Heavily subject to drift
- Requires kinodynamic motion planning

Example: Blimp Motion Planning

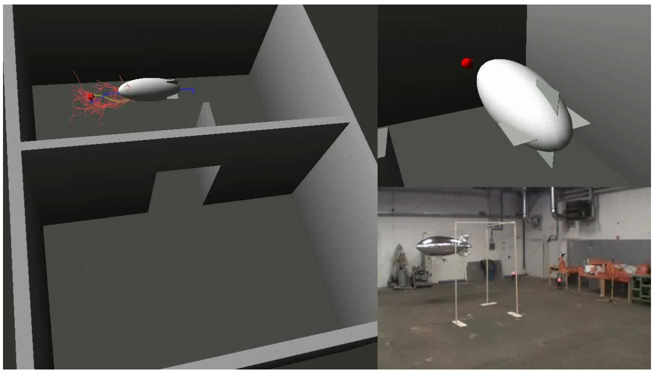
[Müller et al., IROS 2011]

- High-level planner: A* in 4D
- Low-level planner: RRT in 12D considering kinodynamic constraints



Example: Blimp Motion Planning

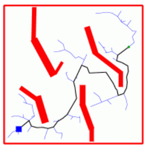
[Müller et al., IROS 2011]

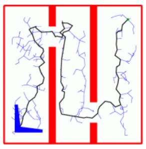


Rapidly Exploring Random Trees

- + 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, …)

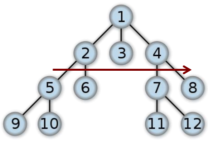
Search Algorithms

- **Given:** Graph G consisting of vertices and edges (with associated costs)
- **Wanted:** Find the best (shortest) path between two vertices

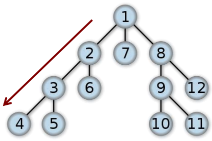
- What search algorithms do you know?

Uninformed Search

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

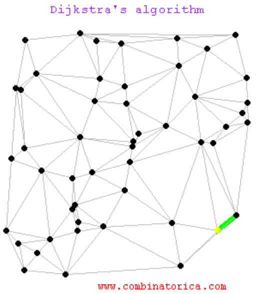


- **Depth-first**
  - Not complete in infinite spaces
  - Not optimal
  - Time $O(b^d)$
  - Space $O(bd)$ (can forget explored subtrees)



Example: Dijkstra's Algorithm

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



Informed Search

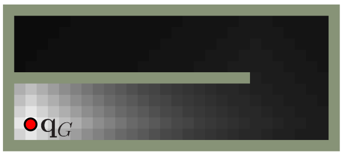
- **Idea**
  - Select nodes for further expansion based on an evaluation function $f(s)$
  - First explore the node with lowest value
- What is a good evaluation function?

Informed Search

- **Idea**
  - Select nodes for further expansion based on an evaluation function $f(s)$
  - First explore the node with lowest value
- What is a good evaluation function?
- Often a combination of
  - Path cost so far $g(s)$
  - Heuristic function $h(s)$ (e.g., estimated distance to goal, but can also encode additional domain knowledge)

What is a Good Heuristic Function?

- Choice is problem/application-specific
- Popular choices
  - Manhattan distance (neglecting obstacles)
  - Euclidean distance (neglecting obstacles)
  - Value iteration / Dijkstra (from the goal backwards)



Informed Search

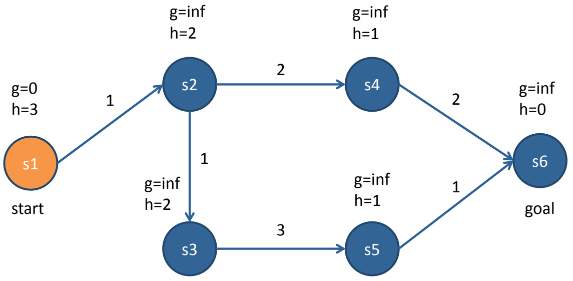
- **A* search**
  - Combines path cost with estimated goal distance
    $$f(s) = g(s) + h(s)$$
  - Heuristic function $h(s)$ has to be
    - Admissible (never over-estimate the true cost)
      $$h(s) < c^*(s, s_{\text{goal}})$$
    - Consistent (satisfies triangle inequality)
- **A* is optimal** (in the number of expanded nodes) and **complete** (finds a solution if there is one and fails otherwise)

A* Algorithm

- Initialize
  - OPEN = {start}, CLOSED = {}
  - f(s) = inf
- While goal not in CLOSED
  - Remove vertex s from OPEN with smallest estimated cost f(s)
  - Insert s into CLOSED
  - For every successor s' of s not yet in CLOSED,
    - Update g(s') = min(g(s'), g(s) + c(s,s'))
    - Insert s' into OPEN

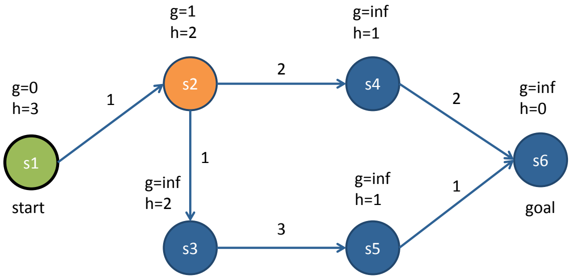
A* Example

- OPEN = {s1}
- CLOSED = {}



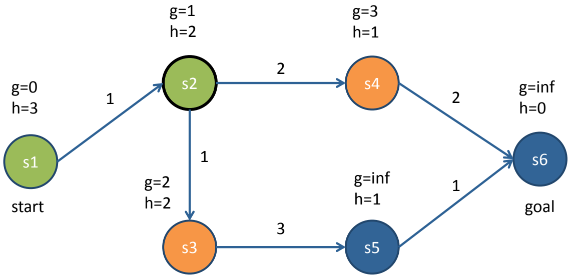
A* Example

- OPEN = {s2}
- CLOSED = {s1}



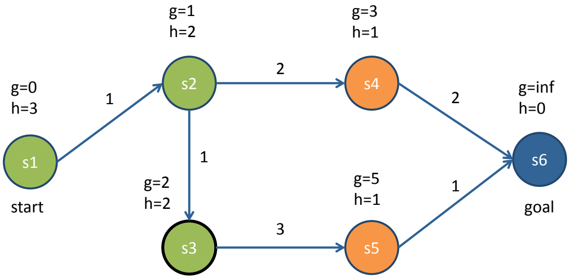
A* Example

- OPEN = {s3,s4}
- CLOSED = {s1,s2}



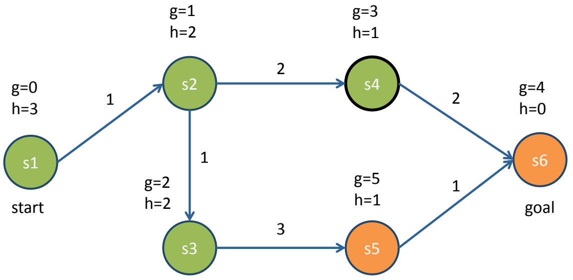
A* Example

- OPEN = {s4}
- CLOSED = {s1,s2,s3}



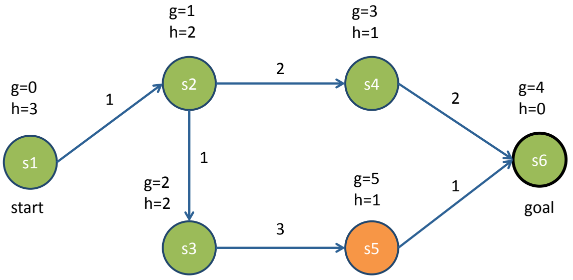
A* Example

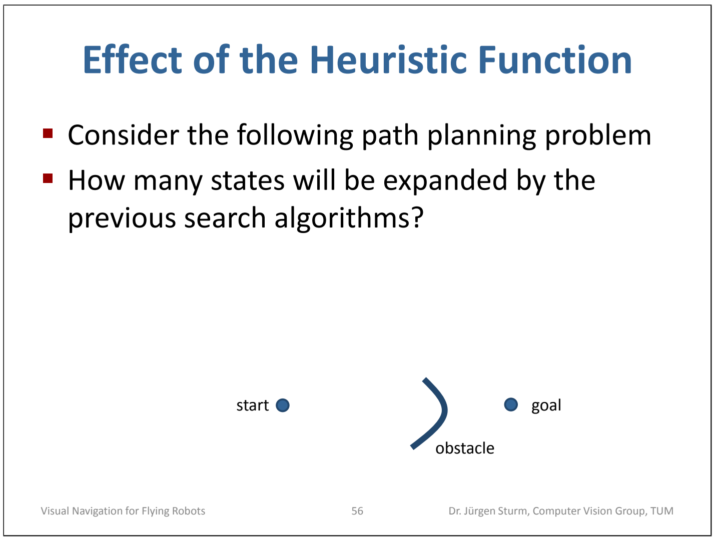
- OPEN = {s5,s6}
- CLOSED = {s1,s2,s3,s4}



A* Example

- OPEN = {s5}
- CLOSED = {s1,s2,s3,s4,**s6**}





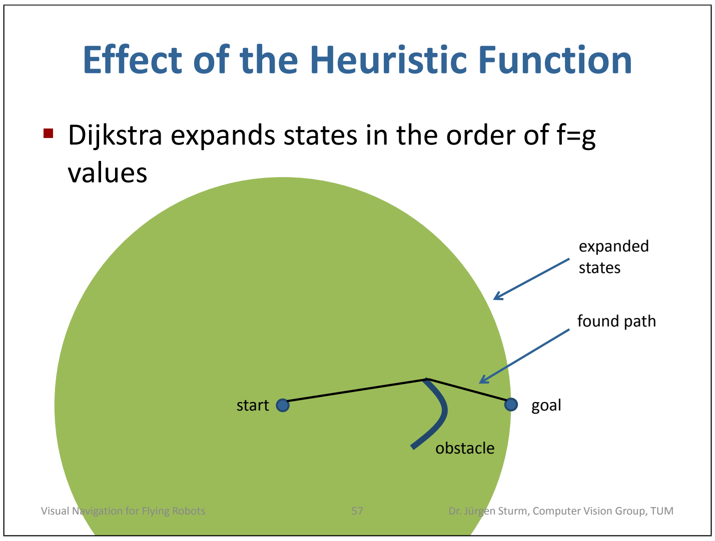
Effect of the Heuristic Function

- Consider the following path planning problem
- How many states will be expanded by the previous search algorithms?

start

obstacle

goal



Effect of the Heuristic Function

- Dijkstra expands states in the order of f=g values

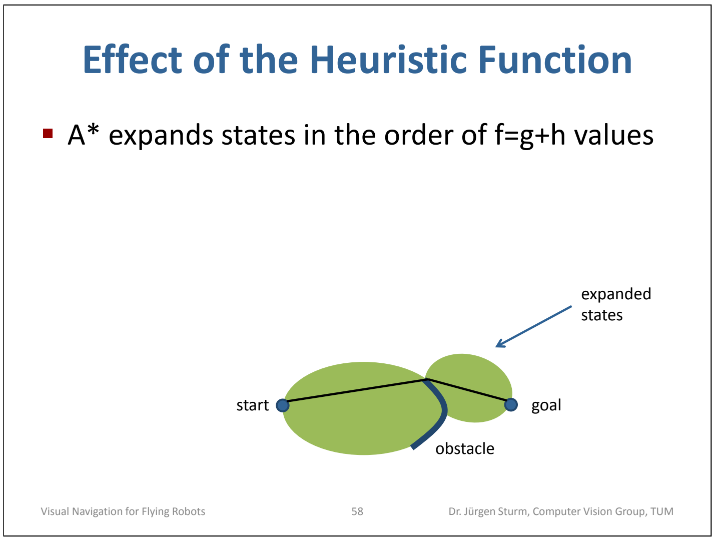
expanded states

found path

start

obstacle

goal



Effect of the Heuristic Function

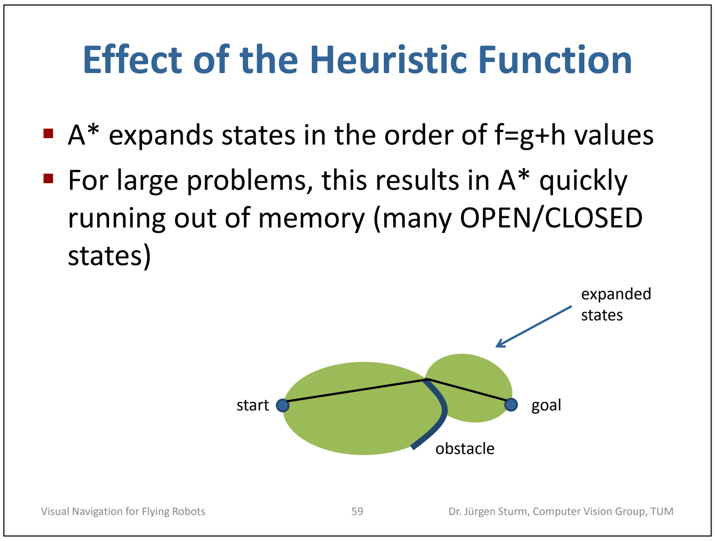
- A* expands states in the order of f=g+h values

expanded states

start

obstacle

goal



Effect of the Heuristic Function

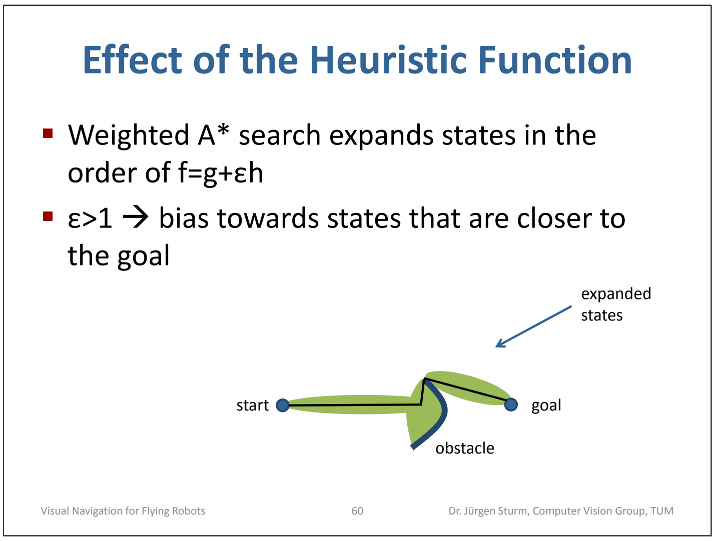
- A* expands states in the order of f=g+h values
- For large problems, this results in A* quickly running out of memory (many OPEN/CLOSED states)

expanded states

start

obstacle

goal



Effect of the Heuristic Function

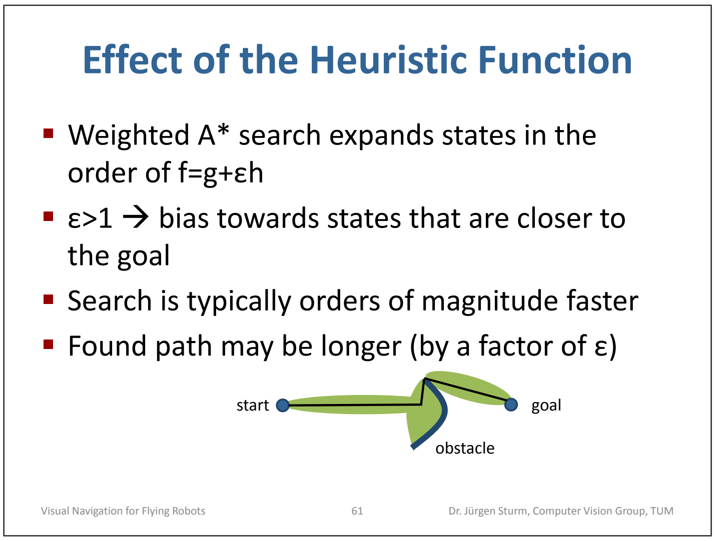
- Weighted A* search expands states in the order of f=g+εh
- ε>1 → bias towards states that are closer to the goal

expanded states

start

obstacle

goal



Effect of the Heuristic Function

- Weighted A* search expands states in the order of f=g+εh
- ε>1 → bias towards states that are closer to the goal
- Search is typically orders of magnitude faster
- Found path may be longer (by a factor of ε)

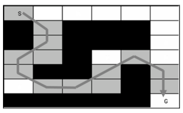
start

obstacle

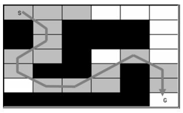
goal

Anytime A*

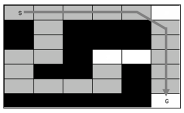
- Constructing anytime search based on A*
  - Find the best possible path for a given ε
  - Reduce ε and re-plan



ε=2.5
expansions: 13
moves: 11

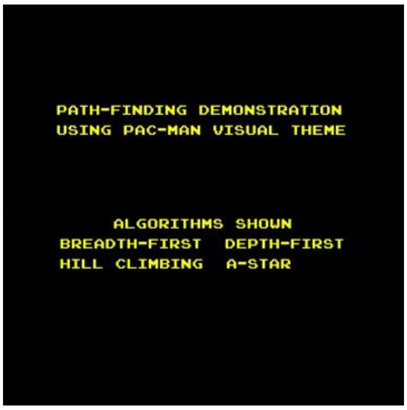


ε=1.5
expansions: 15
moves: 11



ε=1.0
expansions: 20
moves: 10

Comparison Search Algorithms



PATH-FINDING DEMONSTRATION
USING PAC-MAN VISUAL THEME

ALGORITHMS SHOWN
BREADTH-FIRST DEPTH-FIRST
HILL CLIMBING A-STAR

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

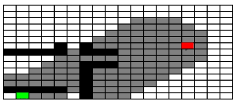
Main concepts

- **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**

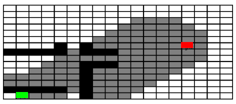
D* Example

- Initial search

Backwards A*



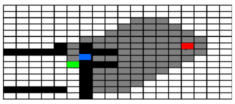
D* Lite



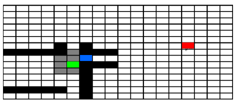
🟩 start
🟥 goal
▨ expanded cell
🟦 new obstacle

- Second search

Backwards A*



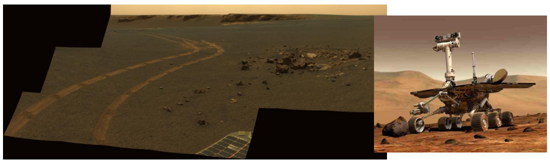
D* Lite



🟩 start
🟥 goal
▨ expanded cell
🟦 new obstacle

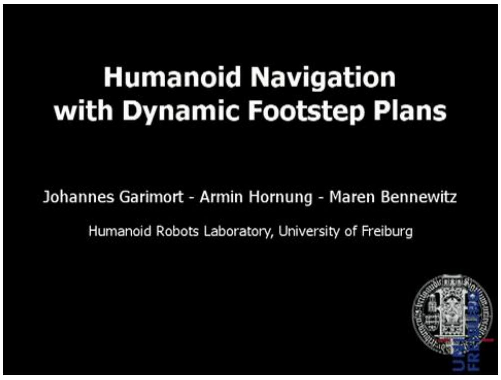
D* Search

- 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]



Humanoid Navigation with Dynamic Footstep Plans

Johannes Garimort - Armin Hornung - Maren Bennewitz

Humanoid Robots Laboratory, University of Freiburg

Problems on A*/D* 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

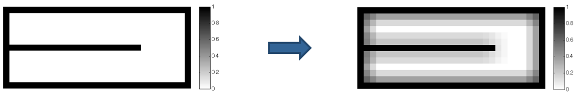
Problems on A*/D* on Grids

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

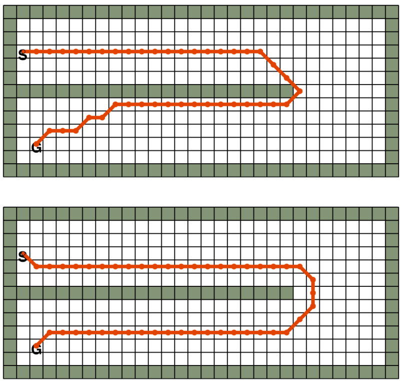
- **Problem:** Path gets close to obstacles
- **Solution:** Convolve the map with a kernel (e.g., Gaussian)



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

$$f(n) = g(s) \cdot p_{\mathrm{occ}}(s) + h(s)$$

Example: Map Smoothing

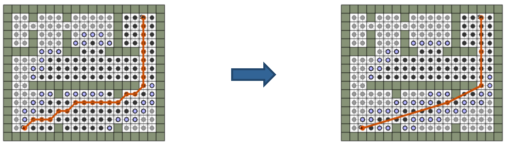


Path Smoothing

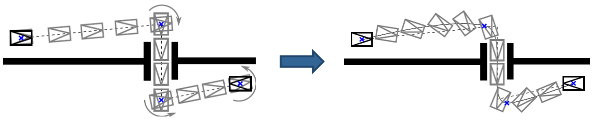
- **Problem:** Paths are aligned to grid structure (because they have to lie in the roadmap)
- Paths look unnatural and are sub-optimal
- **Solution:** Smooth the path after generation
  - Traverse path and find pairs of nodes with direct line of sight; replace by line segment
  - Refine initial path 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



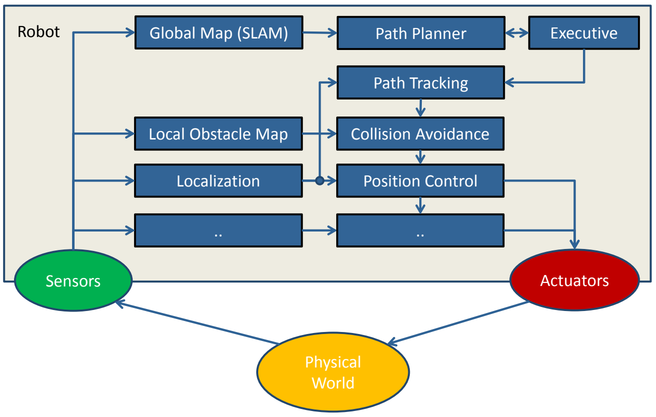
Real-Time Motion Planning

- 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

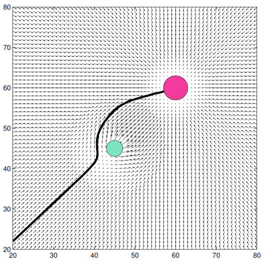
- An approximate **global planner** computes paths ignoring the kinematic and dynamic vehicle constraints (not real-time)
- An accurate **local planner** accounts for the constraints and generates feasible local trajectories in real-time (collision avoidance)

Local Planner

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

Navigation with Potential Fields

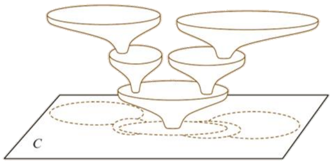
- Treat robot as a particle under the influence of a potential field
- **Pro:**
  - Easy to implement
- **Con:**
  - Suffers from local minima
  - No consideration of dynamic constraints



Navigation with Funnels

[Choi and Latombe, IROS 1991]

- Different regions of the configuration space need different potential fields
- Compose navigation function from overlapping local potential functions (the so-called **funnels**)



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

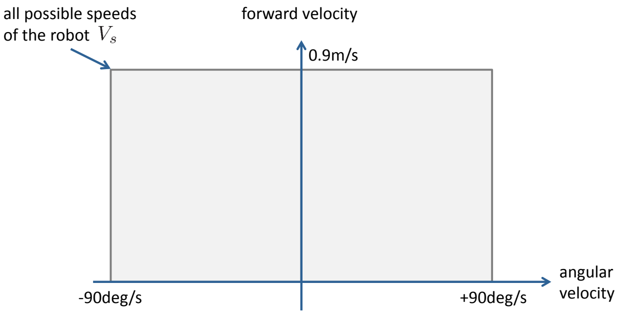
Algorithm:

1. Sample the robot’s control space
2. Simulate each sample for a short period of time
3. Score each sample based on
   - proximity to obstacles
   - proximity to goal
   - proximity to global path
   - speed
4. Pick the highest-scoring control command

Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

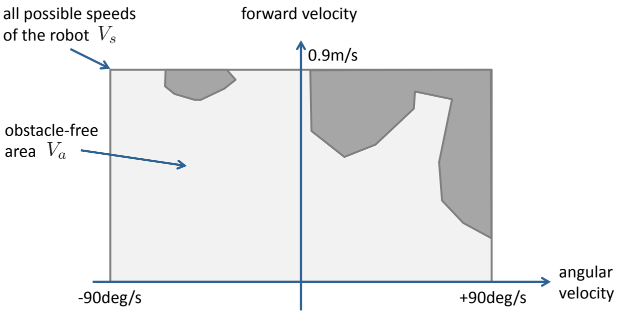
- Consider a 2DOF planar robot



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

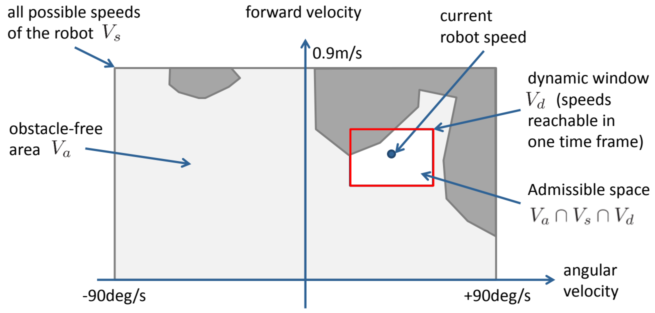
- Consider a 2DOF planar robot + 2D environment



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Consider additionally dynamic constraints



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

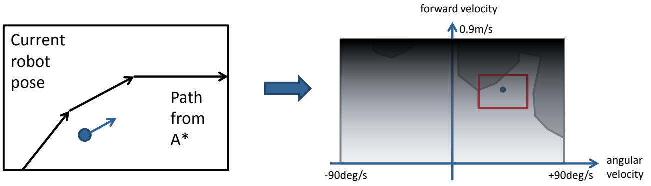
Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Navigation function (potential field)

$$f(n) = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity



Dynamic Window Approach

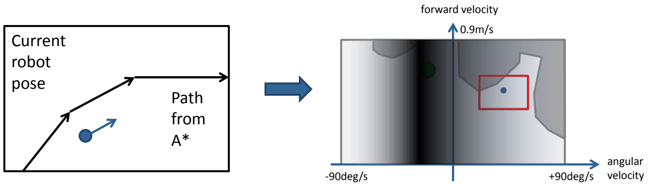
[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Navigation function (potential field)

$$f(n) = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity

Rewards alignment to A* path gradient



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

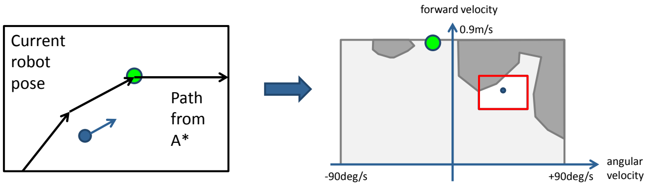
- Navigation function (potential field)

$$f(n) = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity

Rewards alignment to A* path gradient

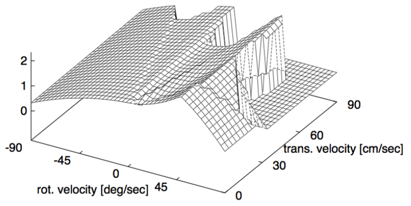
Rewards large advances on A* path



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

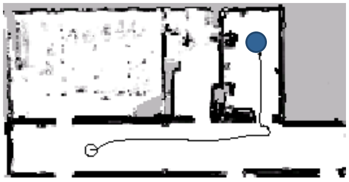


Example: Dynamic Window Approach

[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:

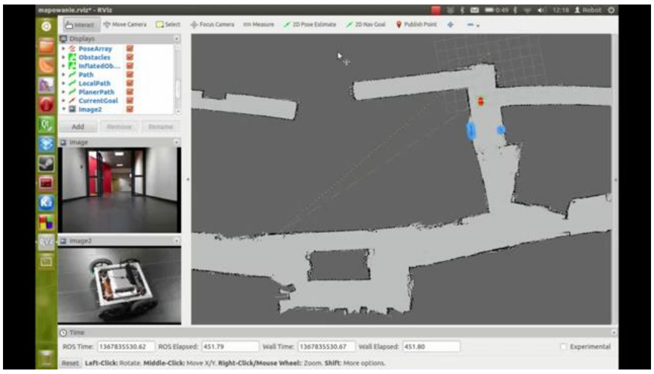


- Can you think of a solution?
- **Note:** General case requires global planning

Example: Motion Planning in ROS

- Executive: state machine (**`move_base`**)
- Global costmap: grid with inflation (**`costmap_2d`**)
- Global path planner: Dijkstra (Dijkstra, **`navfn`**)
- Local costmap (**`costmap_2d`**)
- Local planner: Dynamic window approach (**`base_local_planner`**)

Example: Motion Planning in ROS



ROS Time: 1367835530.62 ROS Elapsed: 451.79 Wall Time: 1367835530.67 Wall Elapsed: 451.80

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.

Lessons Learned Today

- How to sample roadmaps and probabilistic random trees
- How to efficiently compute a path between the start and goal node
- How to update plan efficiently
- How to follow and execute a path in real-time

Lecture:
Autonomous micro aerial vehicles

Friedrich Fraundorfer

Remote Sensing Technology

TU München

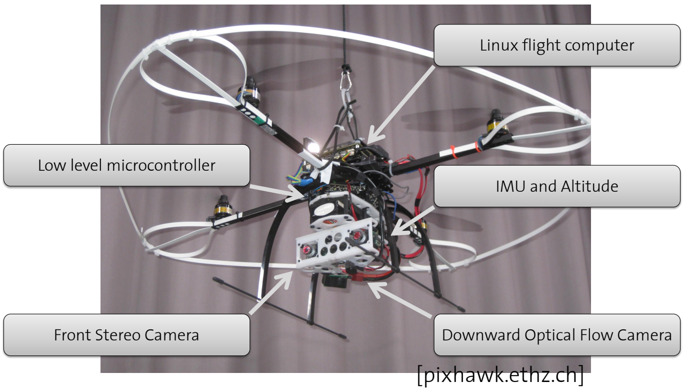
Autonomous operation@ETH Zürich

Autonomous operation@ETH Zürich

Outline

- MAV system
- Optical flow for pose estimation
- 3D mapping for navigation
- Navigation
  - Frontier based exploration
  - VFH+/State lattice

Sensor and System Setup


Linux flight computer

Low level microcontroller

IMU and Altitude

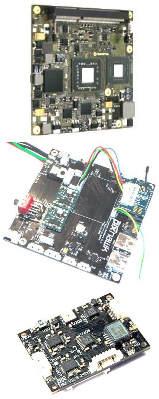
Front Stereo Camera

Downward Optical Flow Camera

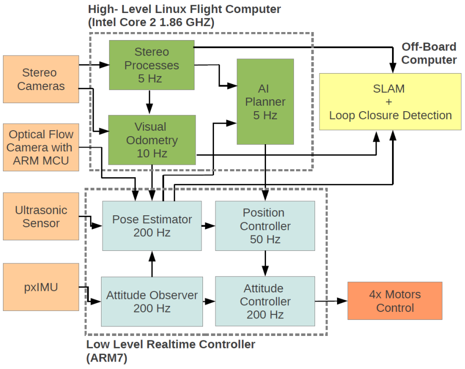
[pixhawk.ethz.ch]

Onboard Electronics

- Computer-On-Module (COTS, 18-27W)
  - microETXexpress industry standard module
  - Intel Core 2 DUO 1.86 GHz / Core i7 2.0 GHz (18-40 W)
- Onboard flight computer (pxCOMex)
  - PIXHAWK mainboard design
  - 200g (incl. cooling, 27W Core 2, 40W Core i7)
- Inertial Measurement Unit (pxIMUv2.5)
  - 60 MHz ARM7
  - 3D gyroscope, accelerometer, compass
  - Barometric pressure
  - Kalman filtering and PID controllers
  - Triggers cameras



System Architecture


High- Level Linux Flight Computer (Intel Core 2 1.86 GHZ)

Off-Board Computer

Stereo Cameras

Optical Flow Camera with ARM MCU

Ultrasonic Sensor

pxIMU

Stereo Processes 5 Hz

Visual Odometry 10 Hz

AI Planner 5 Hz

SLAM + Loop Closure Detection

Pose Estimator 200 Hz

Position Controller 50 Hz

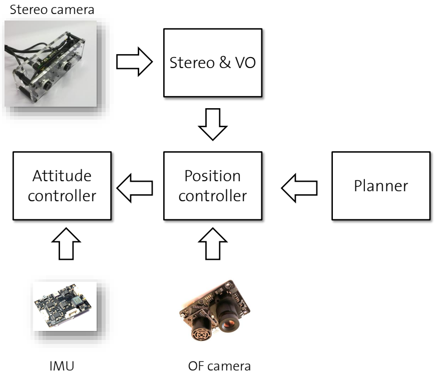
Attitude Observer 200 Hz

Attitude Controller 200 Hz

4x Motors Control

Low Level Realtime Controller (ARM7)

Autonomous Flight


Stereo camera

Stereo & VO

Attitude controller

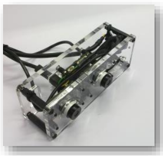
Position controller

Planner

IMU

OF camera

Important sensors



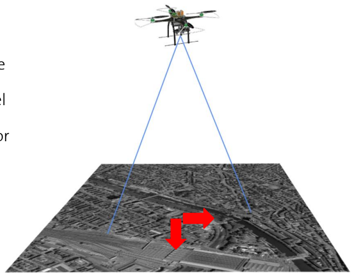
- Stereo camera (timestamped and synchronized with IMU measurements, max. 30fps)

- Optical flow camera

The optical flow idea

Assumptions:

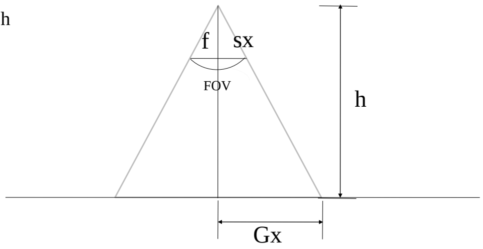
- Planar scene
- Camera plane parallel to ground plane
- Optical flow measures x,y shift in pixel
- Z and metric scale from altitude sensor



The optical flow idea

$$\frac{\mathrm{Gx}}{\mathrm{h}} = \frac{sx}{f} = \tan(\frac{\mathrm{FOV}}{2})$$

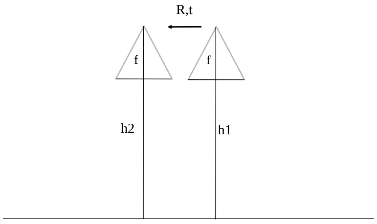
$$\mathrm{Gx} = \frac{sx}{f} h = \tan(\frac{\mathrm{FOV}}{2})\mathrm{h}$$



640 pixel, but how many meters?

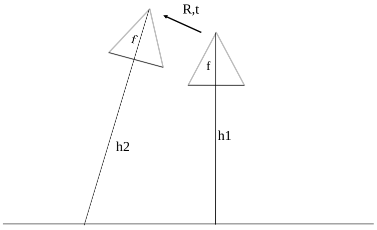
The optical flow idea

$$\mathrm{Gx} = \frac{sx}{f} h$$



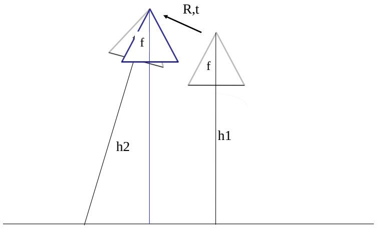
The optical flow idea

$$\mathrm{I2} = \mathrm{H} * \mathrm{I1}$$
$$\mathrm{p2} = \mathrm{H} * \mathrm{p1}$$
$$\mathrm{H} = \mathrm{R} + \frac{1}{\mathrm{d}} TN^T$$



The optical flow idea

$$\mathrm{I2} = \mathrm{H} * \mathrm{I1}$$
$$\mathrm{p2} = \mathrm{H} * \mathrm{p1}$$
$$\mathrm{H} = \mathrm{R} + \frac{1}{\mathrm{d}} TN^T$$

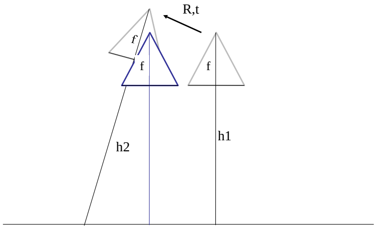


The optical flow idea

$$I2 = H * I1$$

$$p2 = H * p1$$

$$H = R + \frac{1}{d} T N^T$$



Optical flow camera -PX4Flow

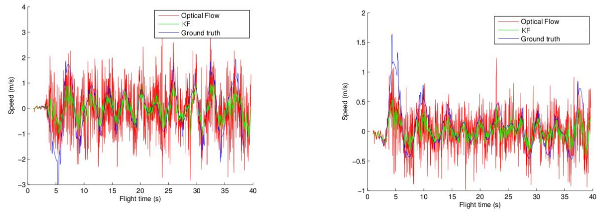
- Smart camera module
  - 752Hx480V (60fps), 188Hx120V (250fps), 16mm lens
  - ARM Cortex M4 (168 MHz, 192 KB RAM, single precision floating point operations)
  - MEMS gyroscope (L3GD20)
  - Ultrasound sensor
- Outputs speed
- Serial interface (Mavlink)
- ROS node (http://www.ros.org/wiki/px4flow_node)

Optical flow camera -PX4Flow

- 64x64 pixel, 250Hz
- SAD optical flow computation
- 8x8 pixel blocks within a 4x4 pixel search range
- Histogram voting
- Subpixel refinement
- Removal of orientation component
- Outputs speed

street texture as seen from the flow sensor from 0.8 m altitude through a 16 mm M12 lens.

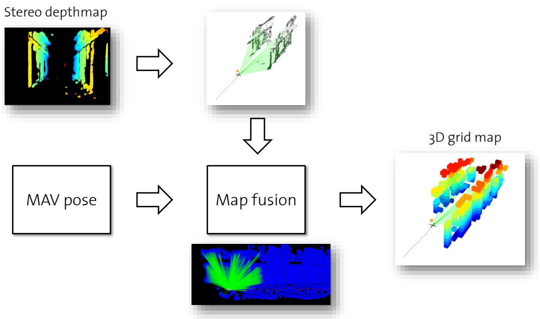
Optical flow camera -PX4Flow



- OF speed filtering necessary (Kalman filter)

Honegger et al. An Open Source and Open Hardware Embedded Metric Optical Flow CMOS Camera for Indoor and Outdoor Applications, ICRA 2013

3D Mapping



Stereo depthmap

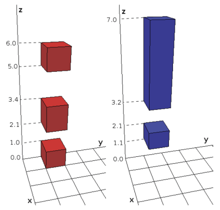
MAV pose

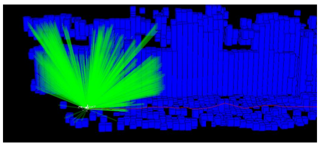
Map fusion

3D grid map

Multi-volume occupancy grid

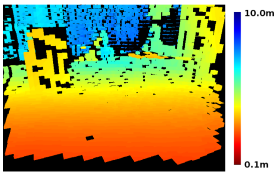
- Multi-volume occupancy grid implementation [Morris 2010].
  - Group sensor readings into continuous vertical volumes which are stored in a 2D grid.
  - Record both positive and negative readings.
    - Models free and occupied space.

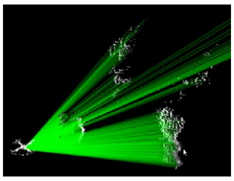




Multi-volume occupancy grid

- Update 3D grid with distance measurements
- Downsample range data from stereo / Kinect to a virtual scan.
  - Outlier removal and efficient occupancy grid updates.
- Each ray in a virtual scan measures the median distance to the range points falling in an angular interval.

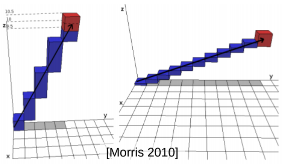




Multi-volume occupancy grid

Updating the map:

- For each ray in the virtual scan,
  - Traverse in order the cells intersected by the ray.
  - Insert negative volumes in the cells until we reach the endpoint of the ray at which we insert a positive volume.
  - Merge overlapping volumes (changes densities of volumes)



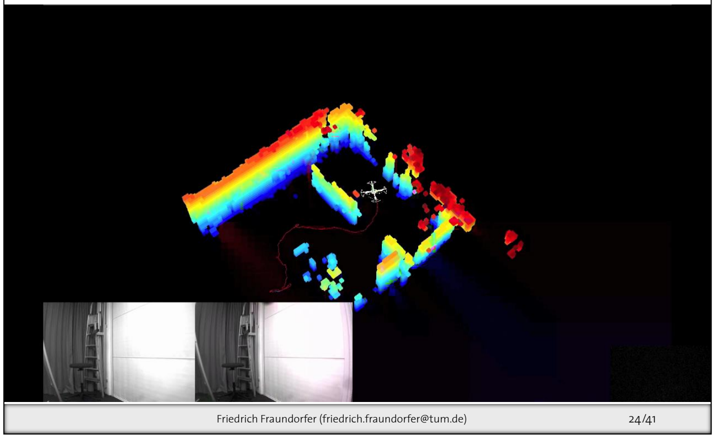
[Morris 2010]

Multi-volume occupancy grid

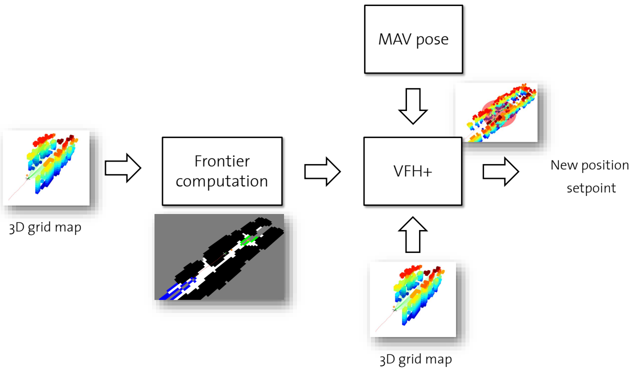
- Occupancy probabilities can be computed for any specific coordinate x,y,z
- Typically one can extract occupancy probabilities for different height planes

$$\text{Occupancy probability of } p = \frac{\text{Occupancy density of positive volume } p \text{ is in}}{\text{Occupancy density of positive \& negative volumes } p \text{ is in}}$$

Video – Online Mapping Test



Exploration



MAV pose

Frontier computation

VFH+

New position setpoint

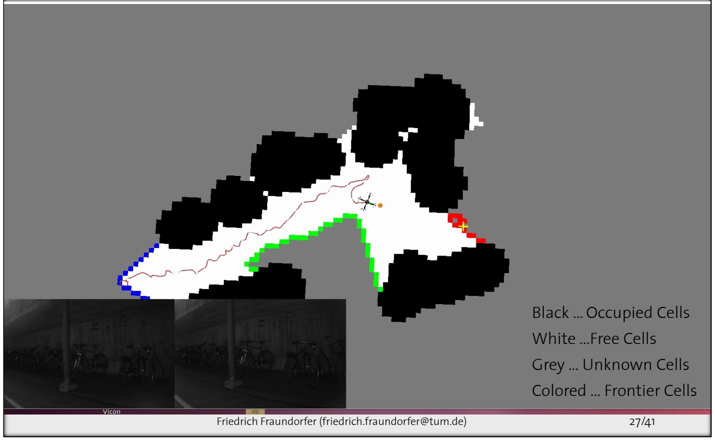
3D grid map

3D grid map

Frontier based exploration

- Method suitable for exploration in grid map (occupancy grid) environment representation
- Frontiers are boundaries between known (sensed) and unknown (unsensed) area.
- MAV is directed to centroid of frontiers

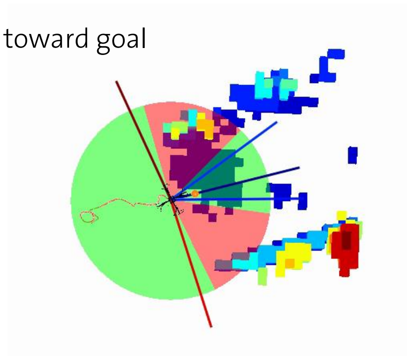
Frontier based exploration



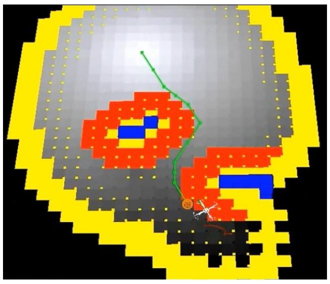
Black ... Occupied Cells
White ...Free Cells
Grey ... Unknown Cells
Colored ... Frontier Cells

Vector Field Histogram (VFH)

- Only 1 waypoint at a time
- Fly discrete steps toward goal



State lattice



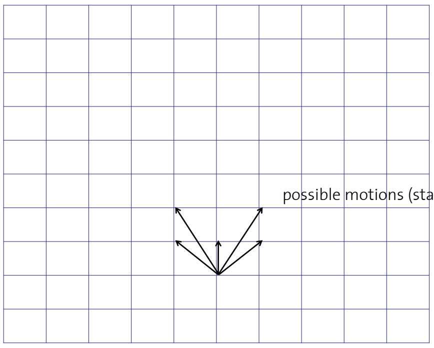
Blue Cell	Occupied cell in 3D occupancy grid map
Red Cell	Non-traversable cell
Yellow Cell	Unexplored cell
Grayscale Cell	Traversable cell (The more white, the nearer to the goal)
Green Line	Planned flight path
Red Line	Trajectory history
Orange Sphere	Current waypoint to follow

Safety Clearance = 0.75m

State lattice

- State lattice concept [Pivtoraiko et al., 2009] can include mobility constraints
- For the MAV we only want to allow forward motion, because of forward looking cameras
- A state lattice is a discretization of the state space and continuous motion primitives that connect these states with edges (graph structure)
- The MAV flies at a fixed attitude; each node represents a 3D state [x, y, θ].

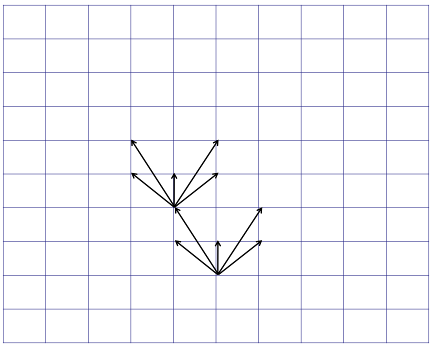
State lattice

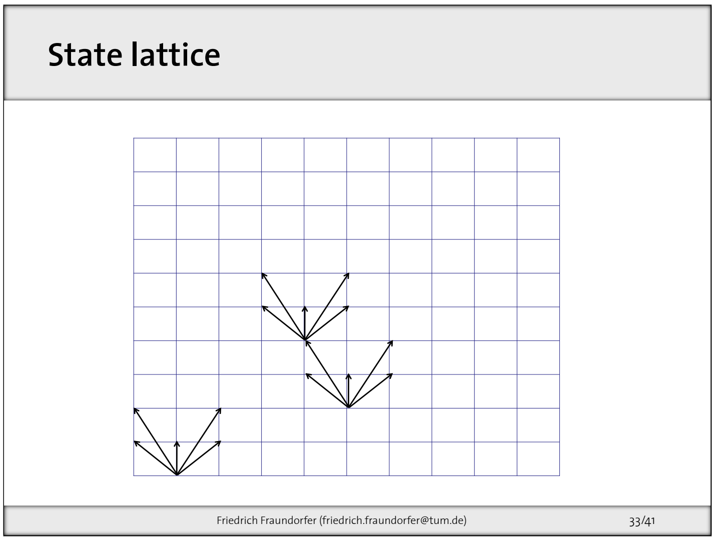


possible motions (state transitions)

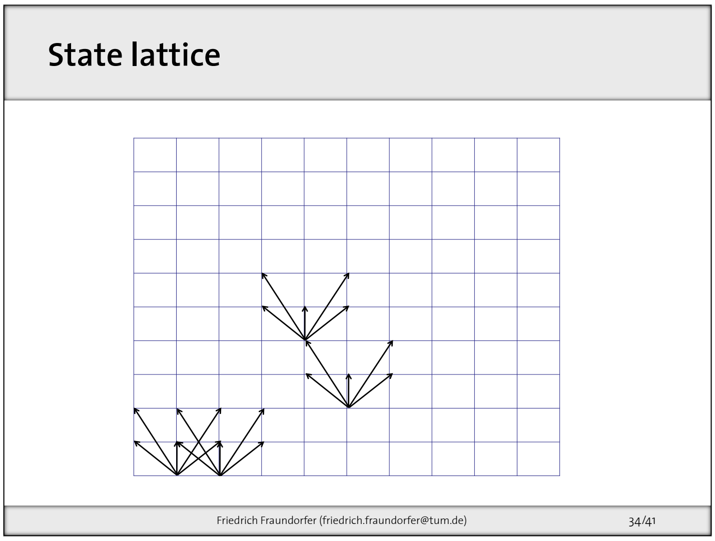
grid of all possible discrete states (here x,y)

State lattice

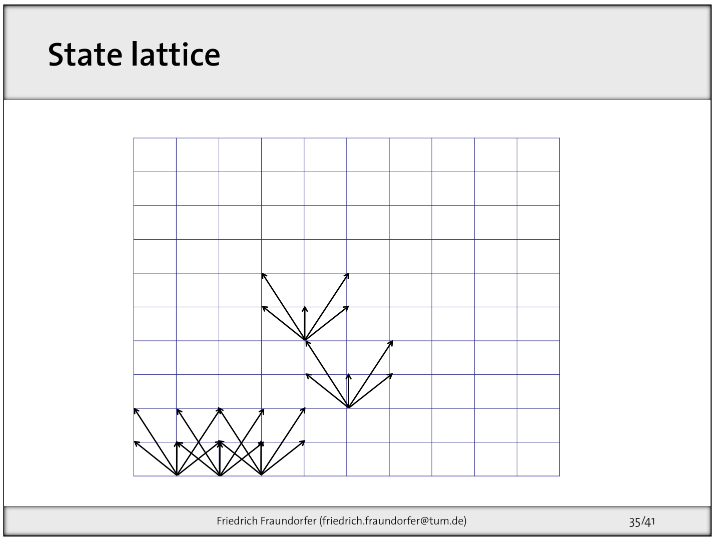




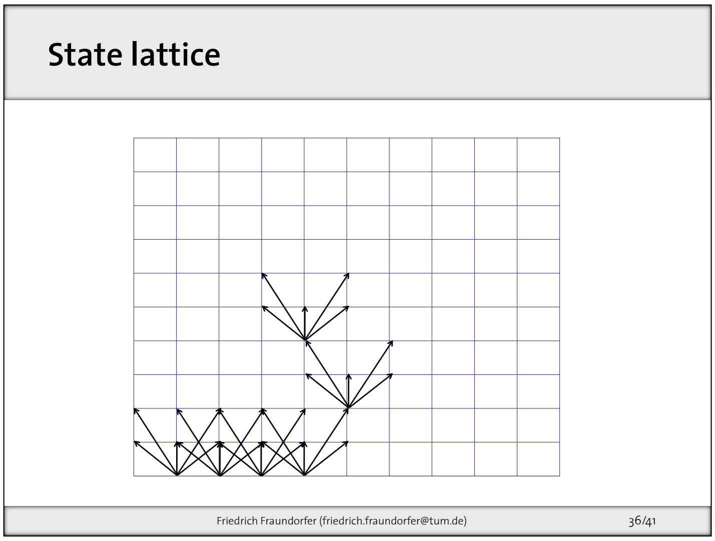
State lattice



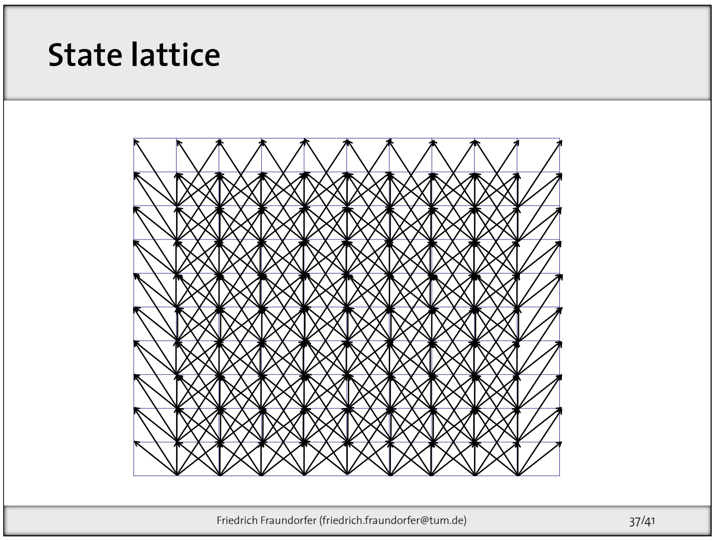
State lattice



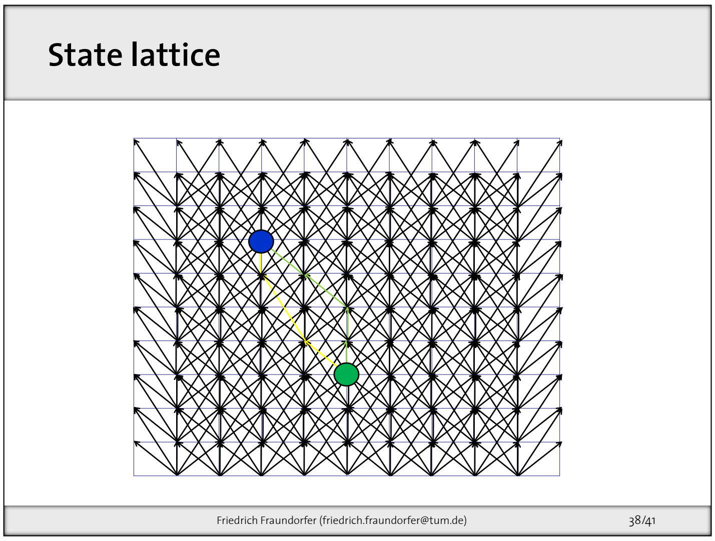
State lattice



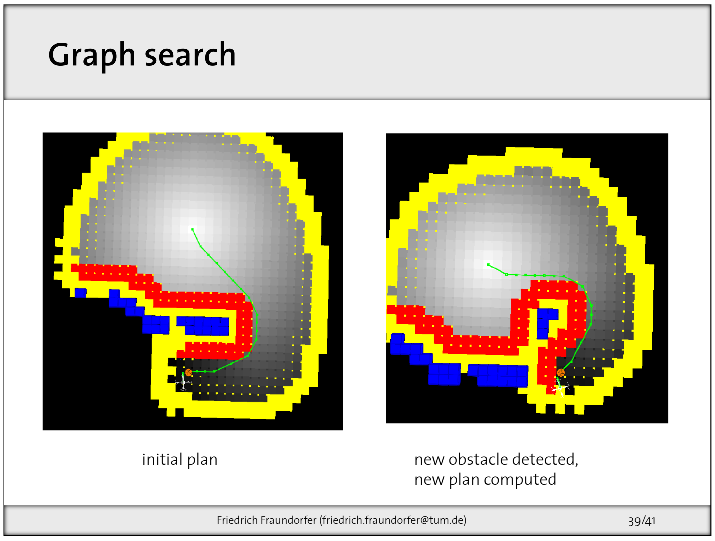
State lattice



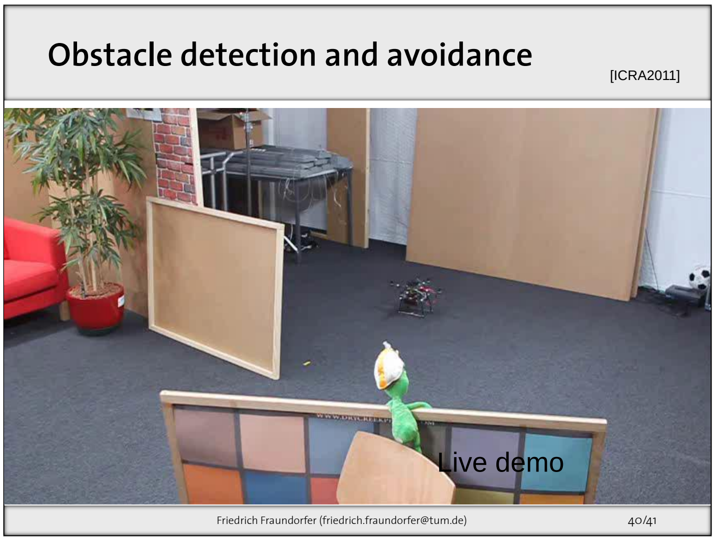
State lattice



State lattice

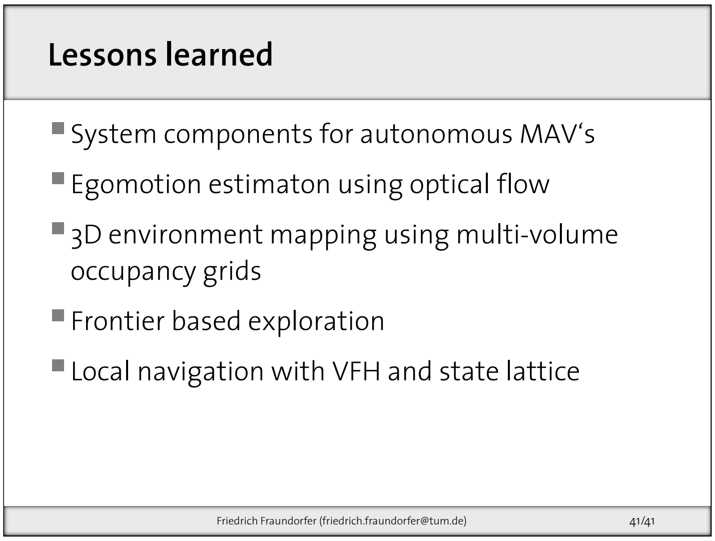


Graph search



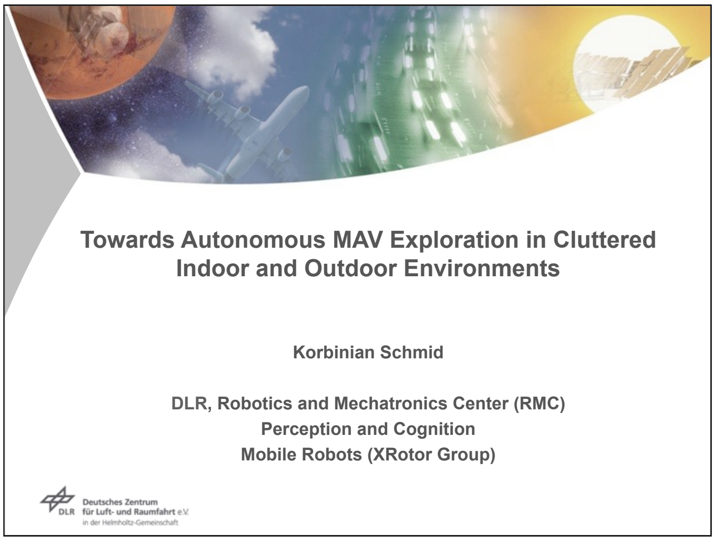
Obstacle detection and avoidance

[ICRA2011]



Lessons learned

- System components for autonomous MAV's
- Egomotion estimaton using optical flow
- 3D environment mapping using multi-volume occupancy grids
- Frontier based exploration
- Local navigation with VFH and state lattice

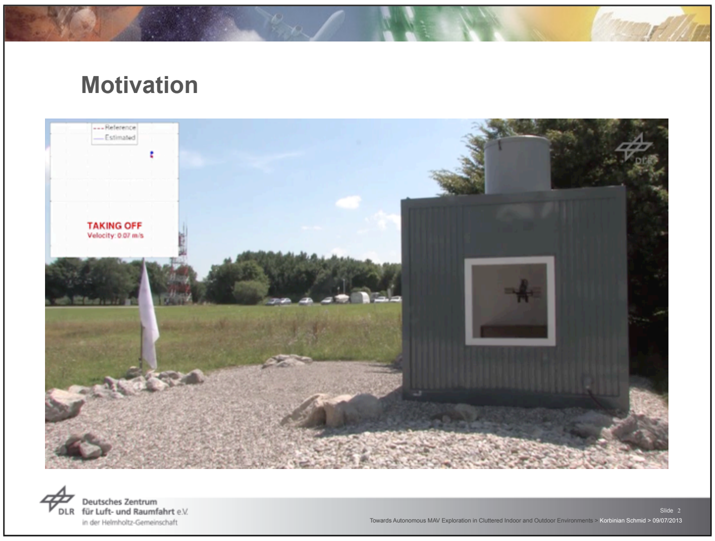


Towards Autonomous MAV Exploration in Cluttered Indoor and Outdoor Environments

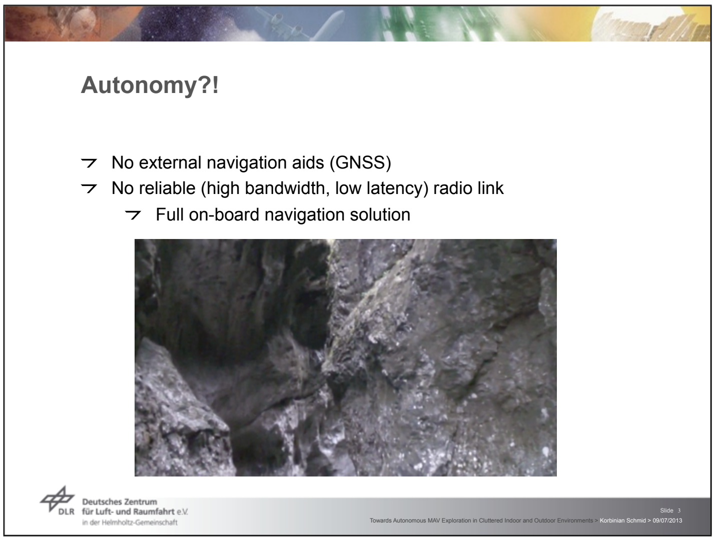
Korbinian Schmid

DLR, Robotics and Mechatronics Center (RMC)
Perception and Cognition
Mobile Robots (XRotor Group)

Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

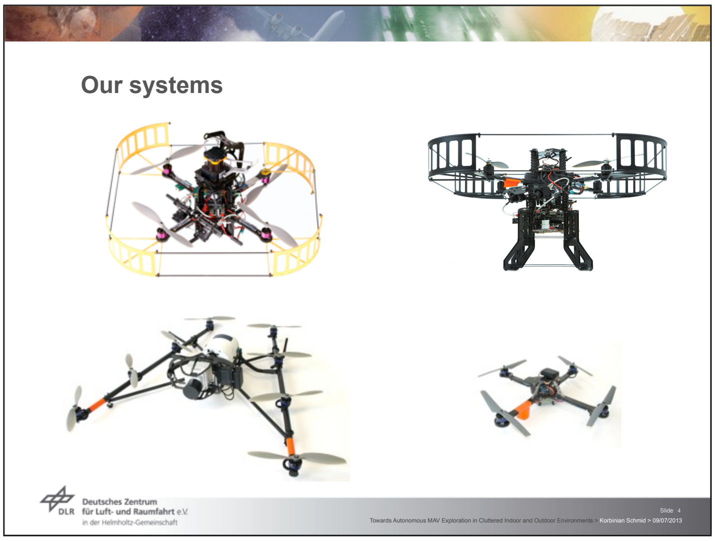


Motivation

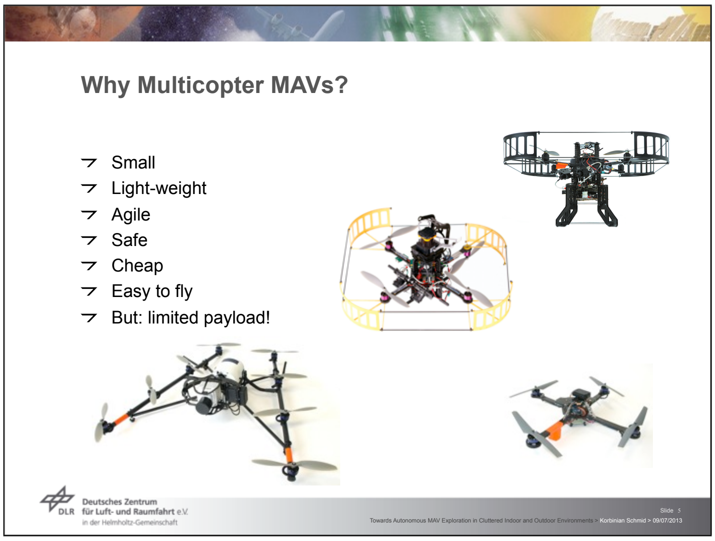


Autonomy?!

- No external navigation aids (GNSS)
- No reliable (high bandwidth, low latency) radio link
  - Full on-board navigation solution

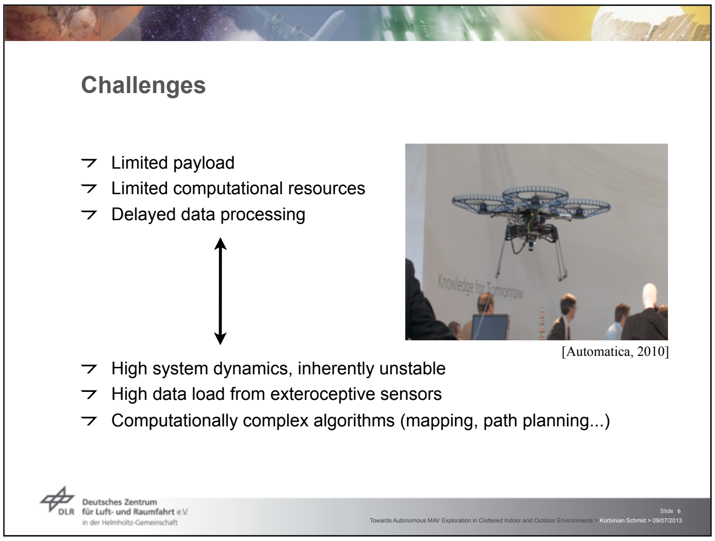


Our systems



Why Multicopter MAVs?

- Small
- Light-weight
- Agile
- Safe
- Cheap
- Easy to fly
- But: limited payload!

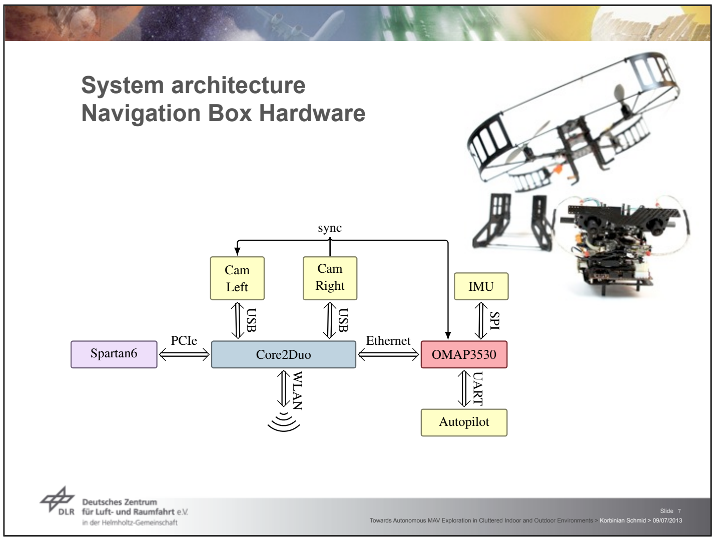


Challenges

- Limited payload
- Limited computational resources
- Delayed data processing

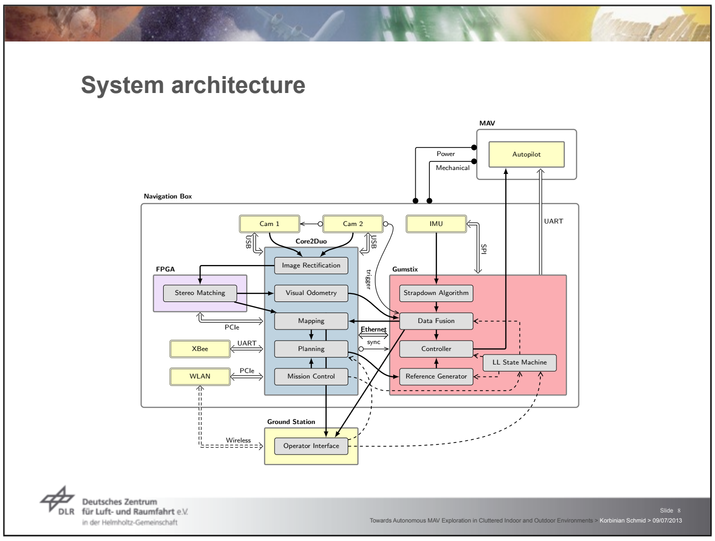
[Automatica, 2010]

- High system dynamics, inherently unstable
- High data load from exteroceptive sensors
- Computationally complex algorithms (mapping, path planning...)



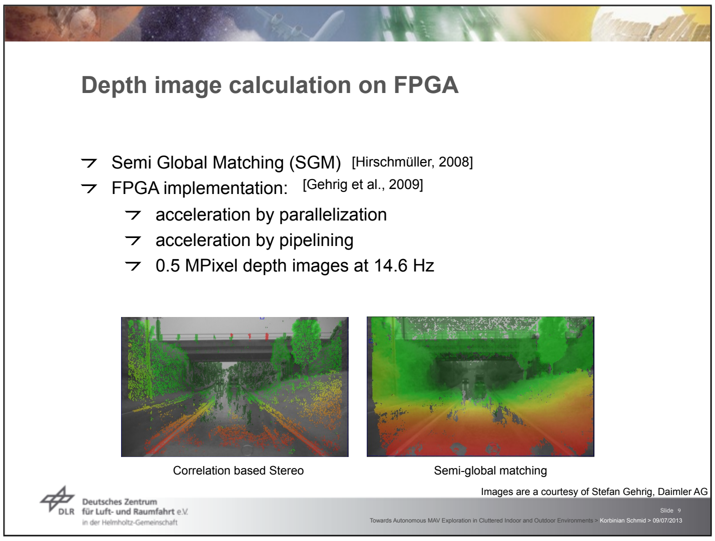
System architecture
Navigation Box Hardware

sync
Cam Left
Cam Right
IMU
USB
USB
SPI
Spartan6
PCIe
Core2Duo
Ethernet
OMAP3530
WLAN
UART
Autopilot



System architecture

MAV
Power
Mechanical
Autopilot
Navigation Box
UART
Cam 1
Cam 2
IMU
USB
Core2Duo
USB
SPI
FPGA
Image Rectification
trigger
Gumstix
Stereo Matching
Visual Odometry
Strapdown Algorithm
Mapping
Data Fusion
PCIe
Ethernet
sync
XBee
UART
Planning
Controller
LL State Machine
WLAN
PCIe
Mission Control
Reference Generator
Ground Station
Wireless
Operator Interface



Depth image calculation on FPGA

- Semi Global Matching (SGM) [Hirschmüller, 2008]
- FPGA implementation: [Gehrig et al., 2009]
  - acceleration by parallelization
  - acceleration by pipelining
  - 0.5 MPixel depth images at 14.6 Hz

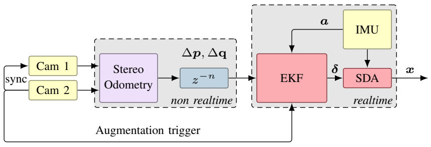
Correlation based Stereo

Semi-global matching

Images are a courtesy of Stefan Gehrig, Daimler AG

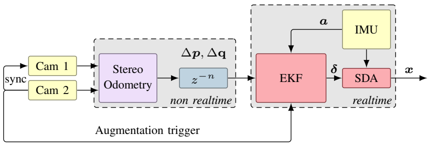
[Schmid et al. IROS 2012]

INS filter design



- Synchronization of realtime and non realtime modules by sensor hardware trigger
- Direct system state: $\boldsymbol{x} = \left(\boldsymbol{p}_{EB}^{E,T}\ \boldsymbol{v}_{EB}^{E,T}\ \boldsymbol{q}_{B}^{E,T}\ \boldsymbol{b}_a^T\ \boldsymbol{b}_\omega^T\right)^T \in \mathbb{R}^{16}$
- High rate calculation by „Strap Down Algorithm“ (SDA)
- Indirect system state: $\boldsymbol{\delta} = \left(\delta \boldsymbol{p}^T\ \delta \boldsymbol{v}^T\ \delta \boldsymbol{\sigma}^T\ \delta \boldsymbol{b}_a^T\ \delta \boldsymbol{b}_\omega^T\right)^T \in \mathbb{R}^{15}$
- Estimation by indirect Extended Kalman Filter (EKF)
- Measurement delay compensation by filter state augmentation

Indirect INS EKF advantages



- Separation of fast system dynamics from slow error dynamics
- Considering measurement time delays only in filter
- State calculation robust to filter divergence
- No system model
- Small angle approximation for attitude error (represented by an error angle vector of size 3 vs. quaternion representation of size 4)

Basic (direct) INS EKF with global position updates

- Prediction (state propagation):

$$\hat{\boldsymbol{x}}_{k+1}^- = \boldsymbol{\Phi}_k \hat{\boldsymbol{x}}_k^+ + \boldsymbol{B}_k \boldsymbol{u}_k$$
$$\boldsymbol{P}_{k+1}^- = \boldsymbol{\Phi}_k \boldsymbol{P}_k^+ \boldsymbol{\Phi}_k^T + \boldsymbol{G}_k \boldsymbol{Q}_k \boldsymbol{G}_k^T$$

- Update :

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{H_k} \boldsymbol{x}_k + \boldsymbol{n}_{\tilde{y}} = \begin{pmatrix} \boldsymbol{p_{E,B}^E} \\ \boldsymbol{q_B^E} \end{pmatrix} + \boldsymbol{n}_{\tilde{y}}$$

$$\boldsymbol{K}_k = \boldsymbol{P}_k^- \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}$$
$$\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_k^-$$
$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k (\tilde{\boldsymbol{y}}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_k^-)$$

Basic (indirect) INS EKF with global position updates

- Direct system state calculation by Strap Down Algorithm
- EKF:
  - Prediction step (state error propagation):

$$\boldsymbol{P}_{k+1}^- = \boldsymbol{\Phi}_k \boldsymbol{P}_k^+ \boldsymbol{\Phi}_k^T + \boldsymbol{G}_k \boldsymbol{Q}_k \boldsymbol{G}_k^T$$

  - Update :

$$\tilde{\boldsymbol{\delta}}_k = (\tilde{\boldsymbol{y}}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_k^-) + \boldsymbol{n}_{\tilde{y}} = \boldsymbol{H}_k \boldsymbol{\delta}_k + \boldsymbol{n}_{\tilde{y}}$$

$$\boldsymbol{K}_k = \boldsymbol{P}_k^- \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}$$
$$\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_k^-$$
$$\boldsymbol{\delta}_k = \boldsymbol{K}_k \tilde{\boldsymbol{\delta}}_k$$

- Correction of direct state

State augmentation (stochastic cloning)

- General measurement: $\tilde{\boldsymbol{y}}_k = \boldsymbol{H_k} \boldsymbol{x}_k + \boldsymbol{n}_{\tilde{y}}$
- Odometry measurement (direct): $\tilde{\boldsymbol{y}}_k = \boldsymbol{t}_{n_2} - \boldsymbol{t}_{n_1} + \boldsymbol{n}_{\tilde{y}} = \boldsymbol{H}_k \begin{pmatrix} \boldsymbol{x}_{n_1} \\ \boldsymbol{x}_{n_2} \end{pmatrix} + \boldsymbol{n}_{\tilde{y}}$
- „Stochastic cloning“ [Roumeliotis, 2002]
- Augmentation at time of measurement [Schmid et al., 2012]:

Time: n_1 n_1+1 n_2 n_k

$$\hat{\boldsymbol{x}}_{n_1} = \hat{\boldsymbol{x}}_{n_1}$$

$$\hat{\bar{\boldsymbol{x}}}_{n_1} = \begin{pmatrix} \hat{\boldsymbol{x}}_{n_1} \\ \hat{\boldsymbol{x}}_{n_1} \end{pmatrix} \qquad \hat{\bar{\boldsymbol{x}}}_{n_1+1} = \begin{pmatrix} \hat{\boldsymbol{x}}_{n_1} \\ \hat{\boldsymbol{x}}_{n_1+1} \end{pmatrix}$$

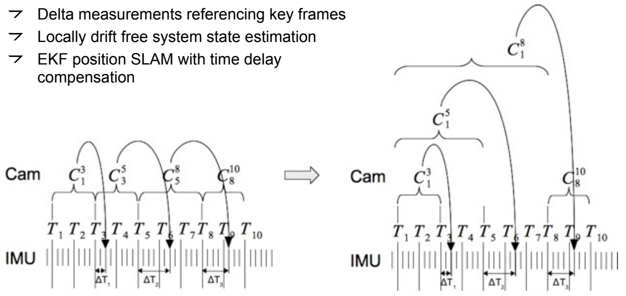
$$\hat{\bar{\boldsymbol{x}}}_{n_2} = \begin{pmatrix} \hat{\boldsymbol{x}}_{n_1} \\ \hat{\boldsymbol{x}}_{n_2} \\ \hat{\boldsymbol{x}}_{n_2} \end{pmatrix} \qquad \hat{\bar{\boldsymbol{x}}}_{k} = \begin{pmatrix} \hat{\boldsymbol{x}}_{n_1} \\ \hat{\boldsymbol{x}}_{n_2} \\ \hat{\boldsymbol{x}}_{n_k} \end{pmatrix}$$

INS EKF with relative time delayed measurements

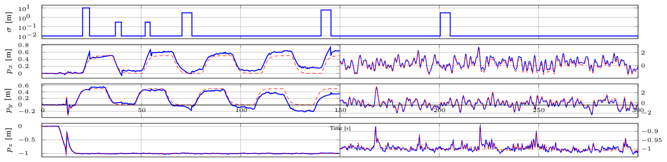
- EKF:
  - State augmentation at exact time of measurement:
    - Saving direct system state
    - Cloning of indirect filter state
  - Prediction step as basic INS EKF
  - Update:
    - Calculate delta pose from saved direct states
    - Calculate error residual from measurement
    - Standard EKF update referencing cloned indirect states in filter
  - Correction of direct states
- Instant processing of arriving (time delayed) measurements

Key frame based stereo odometry

- Delta measurements referencing key frames
- Locally drift free system state estimation
- EKF position SLAM with time delay compensation

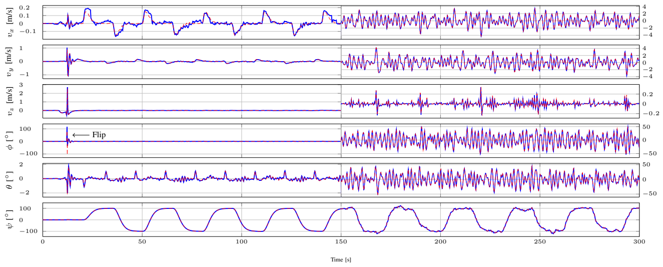

Cam
C_1^3 C_3^5 C_5^8 C_8^{10}
T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}
IMU
ΔT_1 ΔT_2 ΔT_3
Cam
C_1^8
C_1^5
C_1^3 C_8^{10}
T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}
IMU
ΔT_1 ΔT_2 ΔT_3

Simulation (Trajectory)



Simulation (velocity, attitude)

- Velocity up to 4m/s
- Roll/Pitch angles up to 50deg



Simulation: FPGA acceleration vs. RMSE

- RMSE for delay variation:
  - const for position
  - linear for velocity
- Acceleration by parallelization
  - higher frequency
  - lower latency

exponential improvement

- RMSE for frequency variation:
  - exponential for position
  - exponential for velocity

exponential improvement

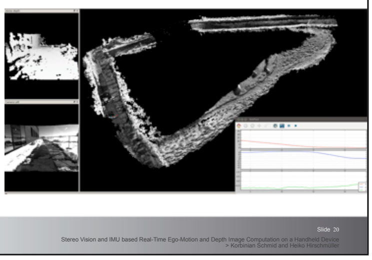
- Acceleration by pipelining
  - higher frequency
  - constant latency

- Estimator properties fit well acceleration by FPGA

[K. Schmid and H. Hirschmüller ICRA 2013]

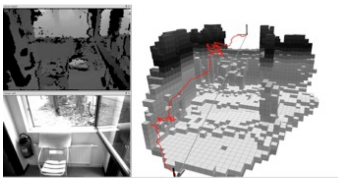
Robustness test

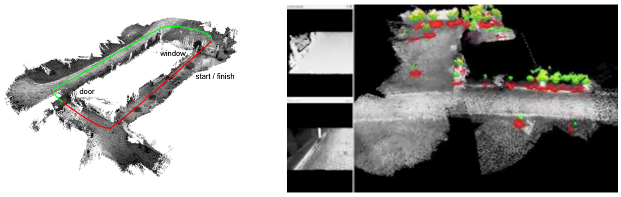
- 70 m trajectory
- Ground truth by tachymeter
- 5 s forced vision drop out with translational motion
- 1 s forced vision drop out with rotational motion
- Estimation error < 1.2 m
- Odometry error < 25.9 m
- Results comparable to runs without vision drop outs

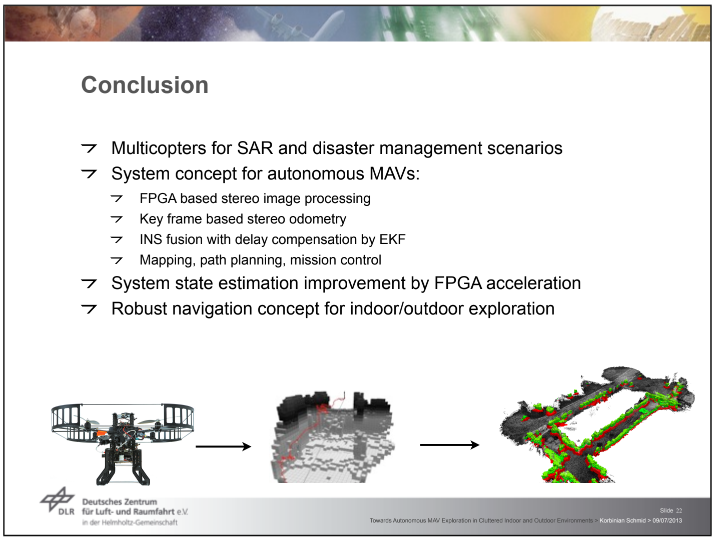


Mixed indoor/outdoor exploration

- Autonomous indoor/outdoor flight of 60m
- Mapping resolution: 0.1m
- Leaving through a window
- Returning through door

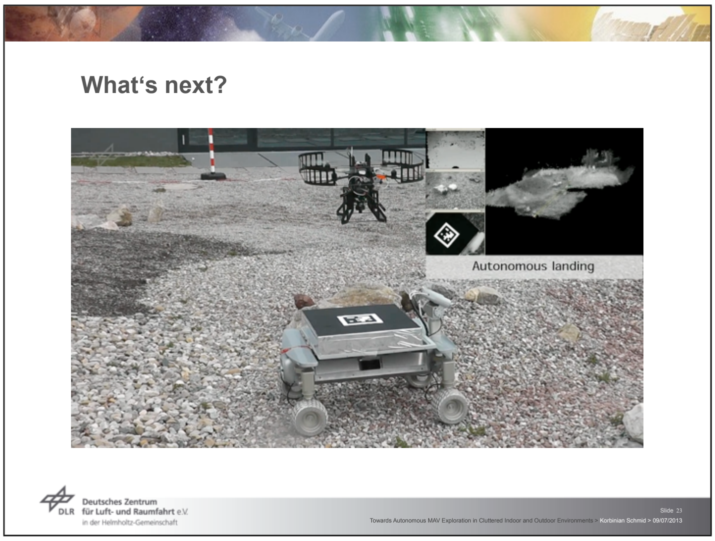






Conclusion

- Multicopters for SAR and disaster management scenarios
- System concept for autonomous MAVs:
  - FPGA based stereo image processing
  - Key frame based stereo odometry
  - INS fusion with delay compensation by EKF
  - Mapping, path planning, mission control
- System state estimation improvement by FPGA acceleration
- Robust navigation concept for indoor/outdoor exploration

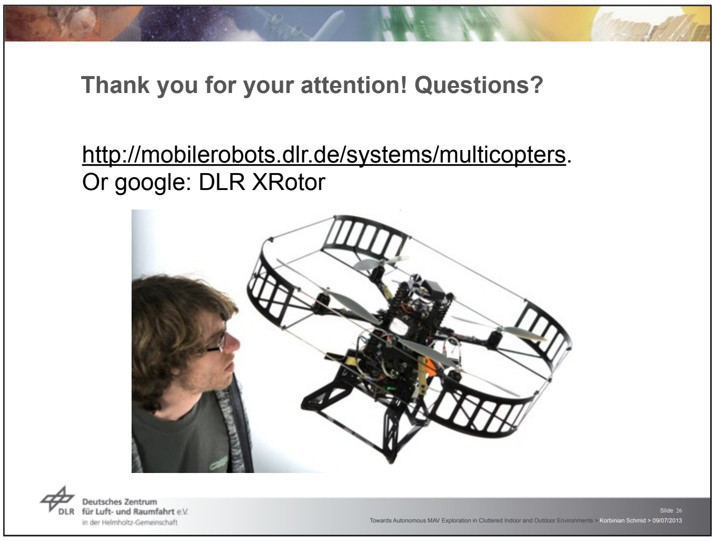


What‘s next?

Autonomous landing

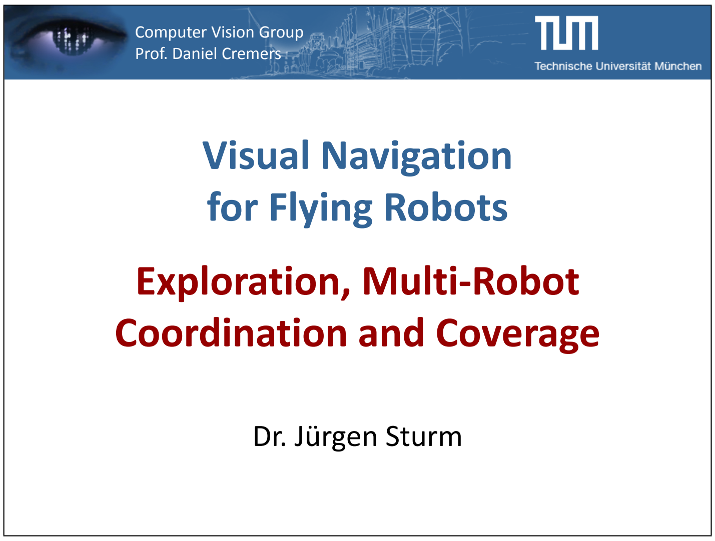
Multicopters in SAR and disaster management scenarios

The RMC XRotor team



Thank you for your attention! Questions?

http://mobilerobots.dlr.de/systems/multicopters.
Or google: DLR XRotor



Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Visual Navigation for Flying Robots

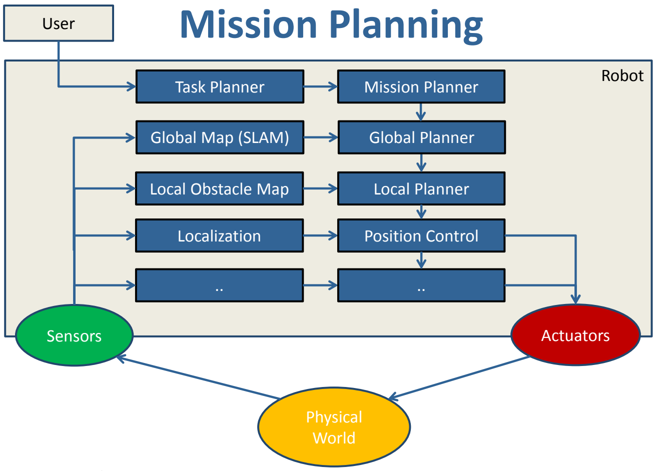
Exploration, Multi-Robot Coordination and Coverage

Dr. Jürgen Sturm

Agenda for Today

- Exploration with a single robot
- Coordinated exploration with a team of robots
- Coverage planning
- Benchmarking

Mission Planning



Mission Planning

- **Goal:** Generate and execute a plan to accomplish a certain (navigation) task
- Example tasks
  - Exploration
  - Coverage
  - Surveillance
  - Tracking
  - …

Task Planning

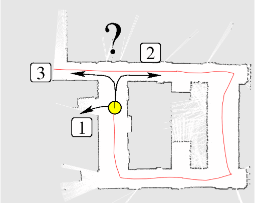
- **Goal:** Generate and execute a high level plan to accomplish a certain task
- Often symbolic reasoning (or hard-coded)
  - Propositional or first-order logic
  - Automated reasoning systems
  - Common programming languages: Prolog, LISP
- Multi-agent systems, communication
- Artificial Intelligence

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

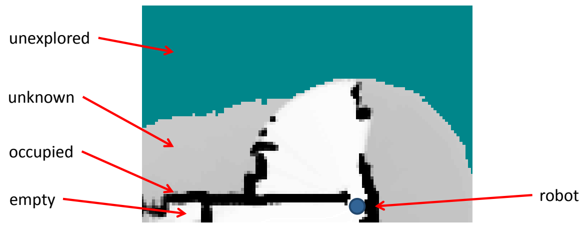
- Choose the action that maximizes utility

$$a^* = \arg\max_{a \in A} U(m, a)$$

- **Question:** How can we define utility?

Example

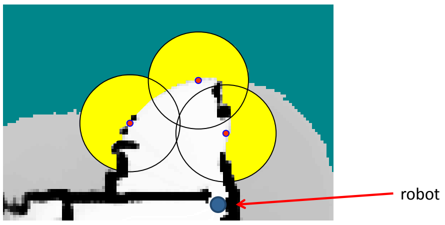
- Where should the robot go next?



Maximizing the Information Gain

- Pick the action a that maximizes the **information gain** given a map m

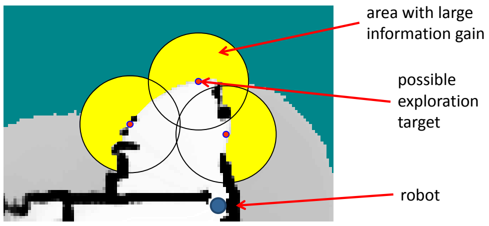
$$a^* = \arg\max_{a \in A} IG(m, a)$$



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 $X = x$

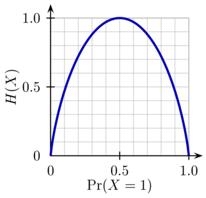
$$\begin{aligned} H(X) &= E(I(X)) \\ &= E(-\log p(X)) \\ &= -\sum_{i=1}^{n} p(x_i) \log p(x_i) \end{aligned}$$

Example: Binary Random Variable

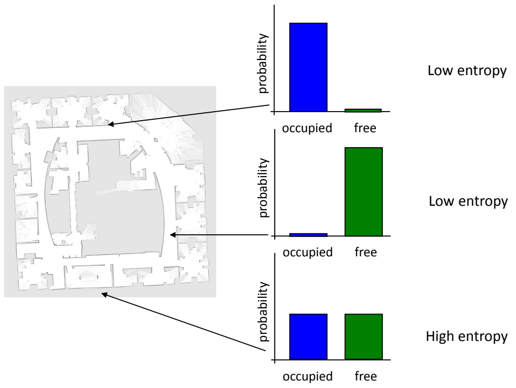
- Binary random variable $X \in \{0, 1\}$
- Probability distribution $P(X = 1) = p$
- How many bits do we need to transmit one sample of $p(X)$?
  - For p=0?
  - For p=0.5?
  - For p=1?

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

- **Entropy of a grid map**

$$H(p(x_t)) = -\sum_{c\in m} p(c)\log p(c) + (1-p(c))\log(1-p(c))$$

grid cells

probability that the cell is occupied

- **Information gain** = reduction in entropy

$$IG(t+1 \mid t) = H(p(x_t)) - H(p(x_{t+1}))$$

Maximizing the Information Gain

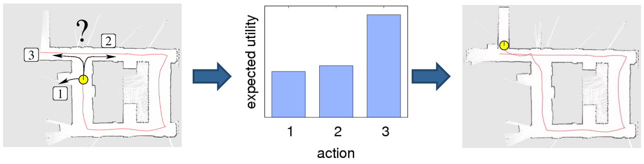
- To compute the information gain one needs to know the observations obtained when carrying out an action

$$a^* = \arg\max_{a\in A} IG(m,a)$$

- 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



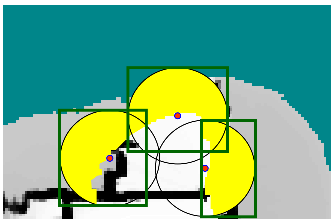
Exploration Costs

- 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

$$a^* = \arg\max_{a\in A} IG(m,a) - \alpha \cdot E(cost(m,a))$$

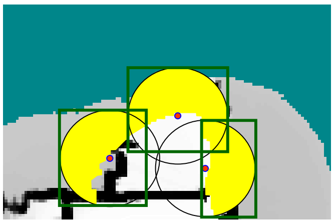
Exploration

- 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…



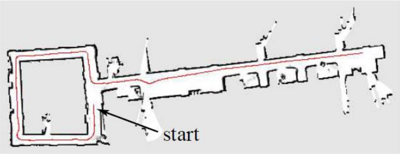
Exploration Actions

- 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)

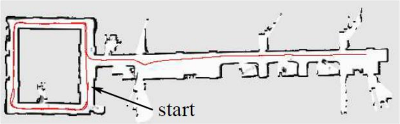
Example: Active Loop Closing

[Stachniss et al., 2005]

- Reduce map uncertainty

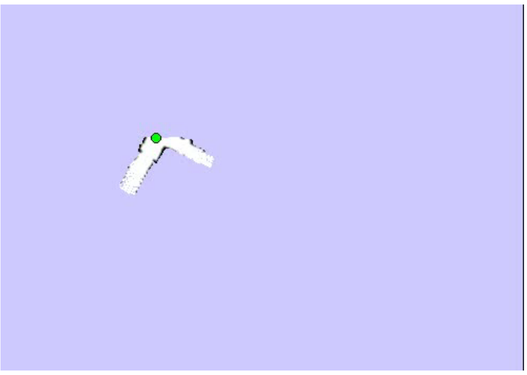


- Reduce map + path uncertainty



Example: Active Loop Closing

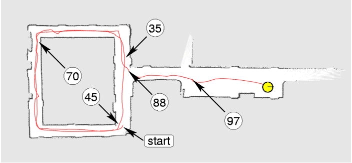
[Stachniss et al., 2005]

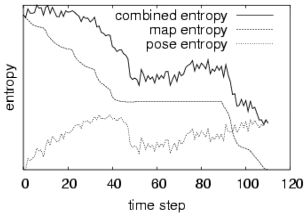


Example: Active Loop Closing

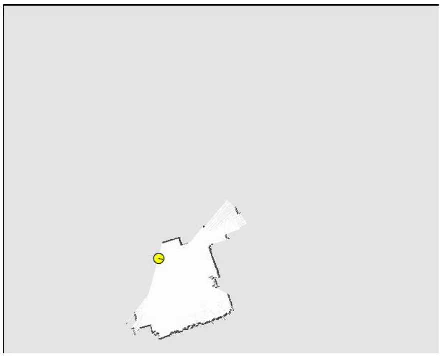
[Stachniss et al., 2005]

- Entropy evolution





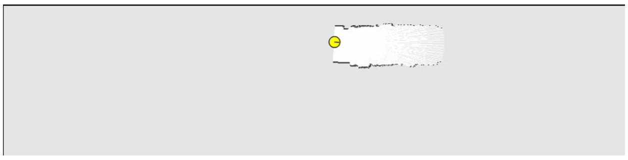
Example: Reduce uncertainty in map, path, and pose [Stachniss et al., 2005]



Selected target location

Corridor Exploration

[Stachniss et al., 2005]

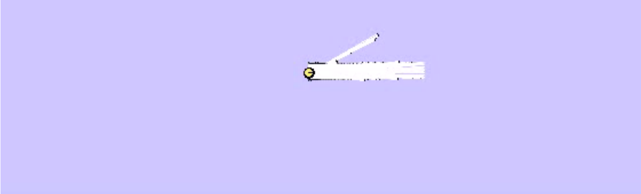


- The decision-theoretic approach leads to **intuitive behaviors:** “re-localize before getting lost”
- Some animals show a similar behavior

Multi-Robot Exploration

Given: Team of robots with communication

Goal: Explore the environment as fast as possible

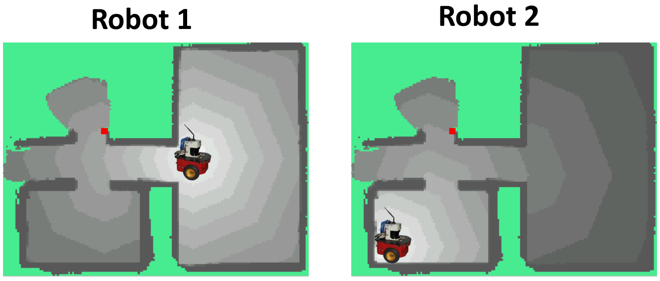


[Wurm et al., IROS 2011]

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?



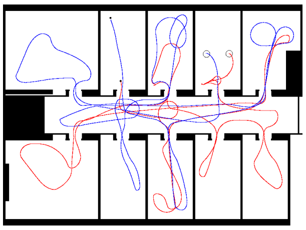
- Without coordination, two robots might choose the same exploration frontier

Levels of Coordination

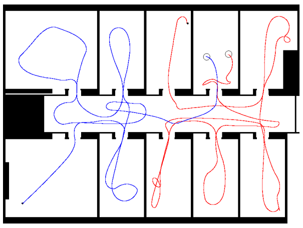
1. **No exchange of information**
2. **Implicit coordination**: Sharing a joint map
   - Communication of the individual maps and poses
   - Central mapping system
3. **Explicit coordination:** Determine better target locations to distribute the robots
   - Central planner for target point assignment
   - Minimize expected path cost / information gain / …

Typical Trajectories

Implicit coordination:

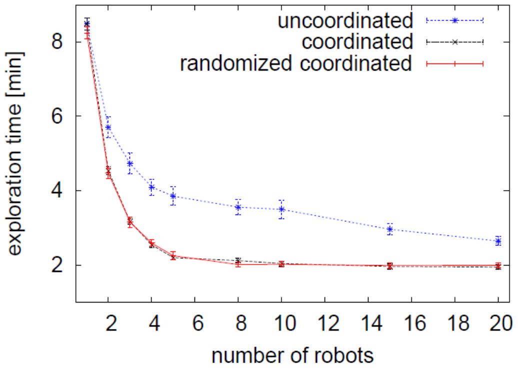


Explicit coordination:



Exploration Time

[Stachniss et al., 2006]



Coordination Algorithm

In each time step:

- Determine set of exploration targets

$$S = \{s_1, \ldots, s_n\}$$

- Compute for each robot i and each target j the expected cost/utility C_{ij}
- Assign robots to targets using the **Hungarian algorithm**

Hungarian Algorithm

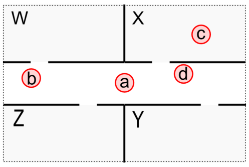
[Kuhn, 1955]

- 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

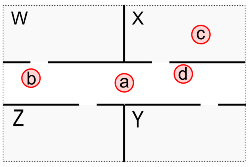


1. Compute the cost matrix (non-negative)

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



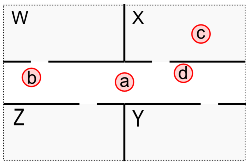
2. Find minimum element in each row

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



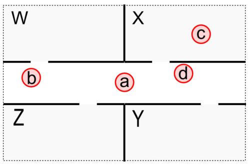
3. Subtract minimum from each row element

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



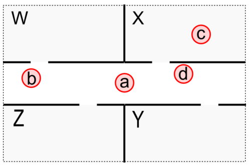
4. Find minimum element in each column

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



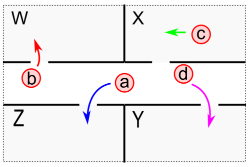
5. Subtract minimum from each column element

Hungarian Algorithm: Example

targets

robots

	W	X	Y	Z
a	1	0	0	$\boxed{0}$
b	$\boxed{0}$	3	3	1
c	6	$\boxed{0}$	1	4
d	4	0	$\boxed{0}$	3



6a. Assign (if possible)

Hungarian Algorithm: Example

targets

robots

	W	X	Y	Z
a	1	0	0	$\boxed{0}$
b	$\boxed{0}$	3	3	1
c	6	$\boxed{0}$	1	4
d	4	0	$\boxed{0}$	3

6b. If no assignment is possible:

- Connect all 0's by vertical/horizontal lines
- Find the minimum in all remaining elements and subtract
- Add to all double crossed out coefficients
- Repeat step 2 – 6

Hungarian Algorithm: Example

targets

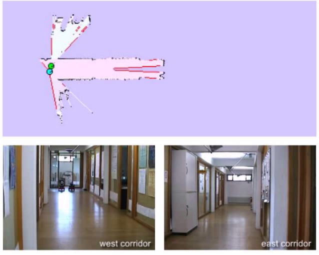
robots

	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

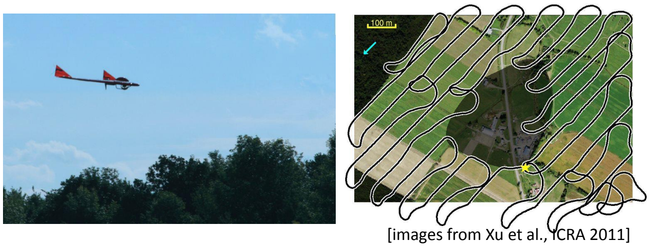


Summary: Exploration

- 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

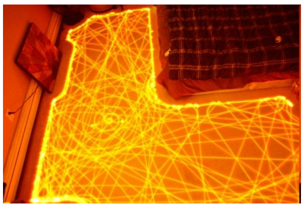
Coverage Path Planning

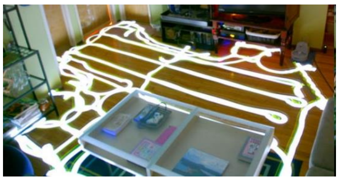
- **Given:** Known environment with obstacles
- **Wanted:** The shortest trajectory that ensures complete (sensor) coverage



[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

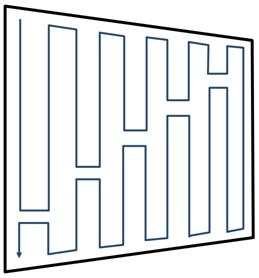
- What is a good coverage strategy?
- What would be a good cost function?
  - Amount of redundant traversals
  - Number of stops and rotations
  - Execution time
  - Energy consumption
  - Robustness
  - Probability of success
  - …

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

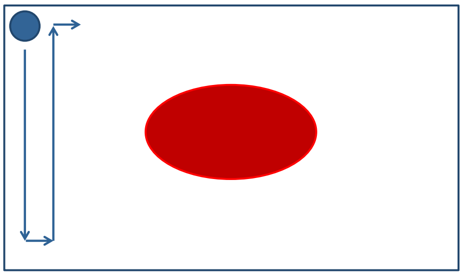
Coverage of Simple Shapes

- Approximately optimal solution often easy to compute for simple shapes (e.g., trapezoids)



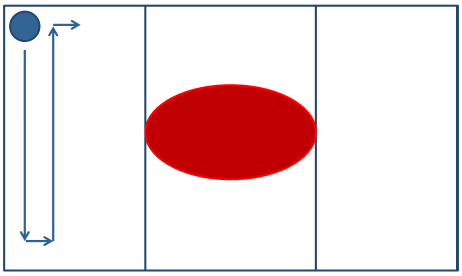
Idea

[Mannadiar and Rekleitis, ICRA 2011]



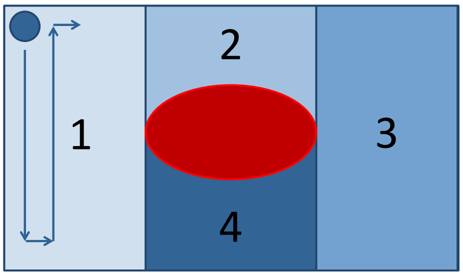
Idea

[Mannadiar and Rekleitis, ICRA 2011]



Idea

[Mannadiar and Rekleitis, ICRA 2011]



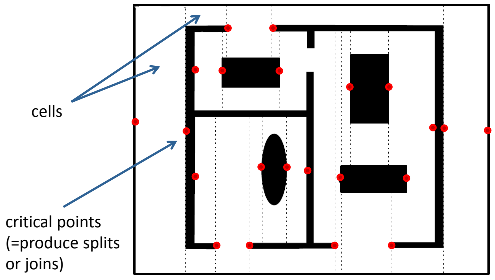
Coverage Based On Cell Decomposition
[Mannadiar and Rekleitis, ICRA 2011]

Approach:

1. Decompose map into “simple” cells
2. Compute connectivity between cells and build graph
3. Solve coverage problem on reduced graph

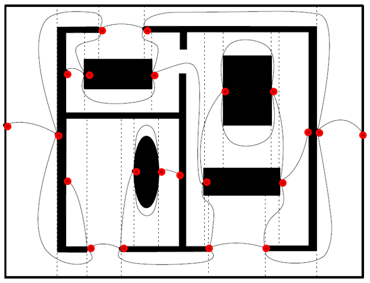
Step 1: Boustrophedon Cellular Decomposition [Mannadiar and Rekleitis, ICRA 2011]

- Similar to trapezoidal decomposition
- Can be computed efficiently in 2D



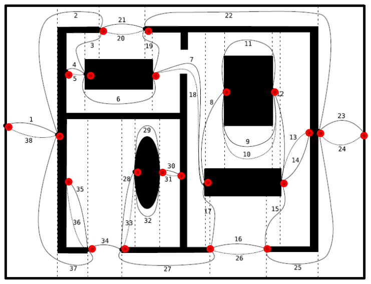
Step 2: Build Reeb Graph
[Mannadiar and Rekleitis, ICRA 2011]

- Vertices = Critical points (that triggered the split)
- Edges = Connectivity between critical points



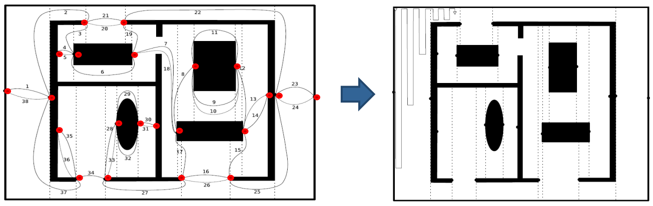
Step 3: Compute Euler Tour
[Mannadiar and Rekleitis, ICRA 2011]

- 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



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)

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

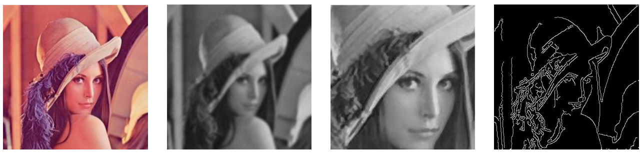
- Middlebury datasets (optical flow, stereo, …)
- Caltech-101, PASCAL (object recognition)
- Stanford bunny (3d reconstruction)

Robotics

- RoboCup competitions (robotic soccer)
- DARPA challenges (autonomous car)
- SLAM datasets

Image Denoising: Lenna Image

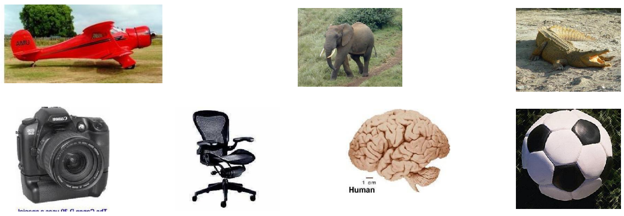
- 512x512 pixel standard image for image compression and denoising
- Lena Söderberg, Playboy magazine Nov. 1972
- Scanned by Alex Sawchuck at USC in a hurry for a conference paper



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



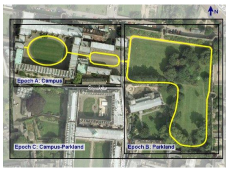
RoboCup Initiative

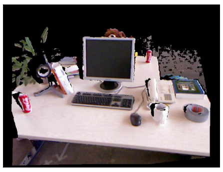
- Evaluation of full system performance
- Includes perception, planning, control, …
- Easy to understand, high publicity
- “By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, complying with the official rule of the FIFA, against the winner of the 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]
- …





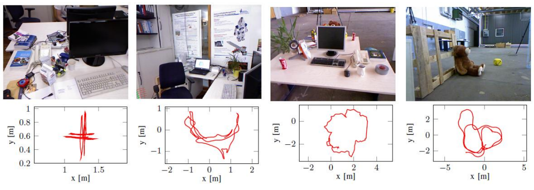
TUM RGB-D Dataset

[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

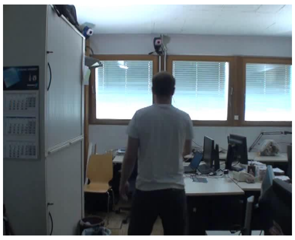
Calibration

Calibration of the overall system is not trivial:

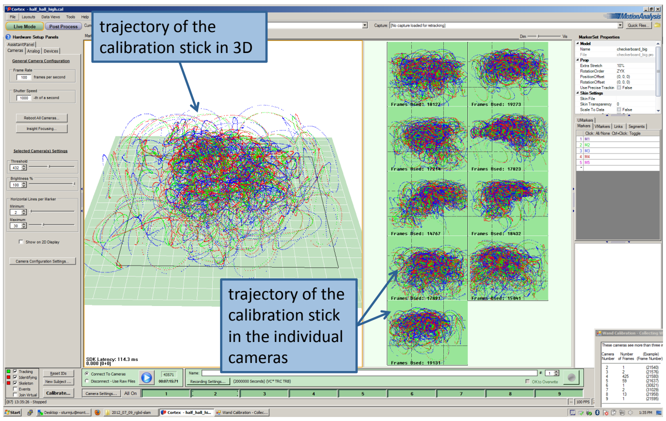
1. Mocap calibration
2. Kinect-mocap calibration
3. Time synchronization

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



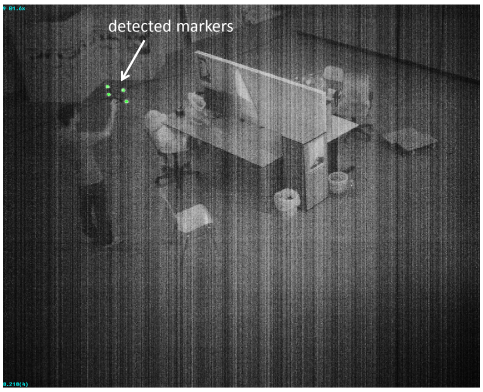
Calibration Step 1: Mocap



trajectory of the calibration stick in 3D

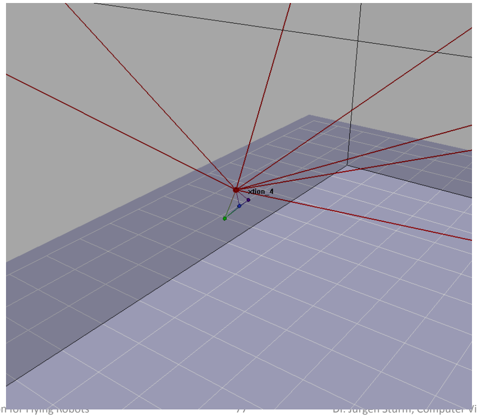
trajectory of the calibration stick in the individual cameras

Example: Raw Image from Mocap

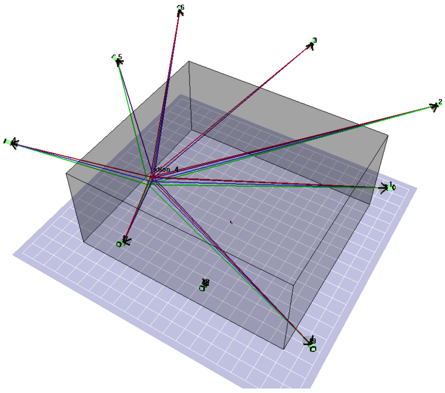


detected markers

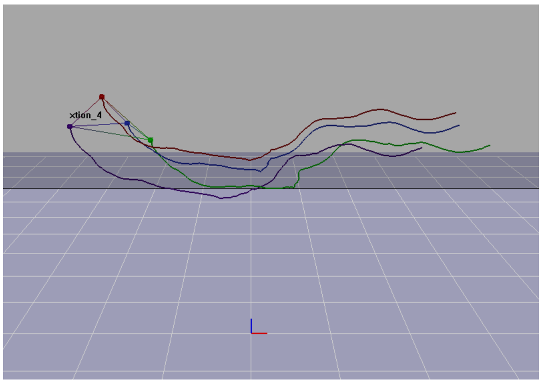
Example: Position Triangulation of a Single Marker



Example: Tracked Object (4 Markers)

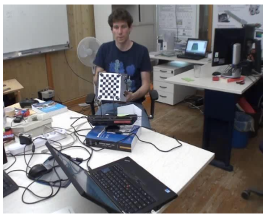


Example: Recorded Trajectory



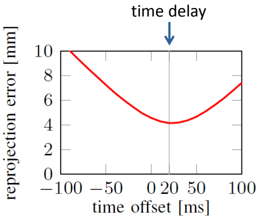
Calibration Step 2: Mocap-Kinect

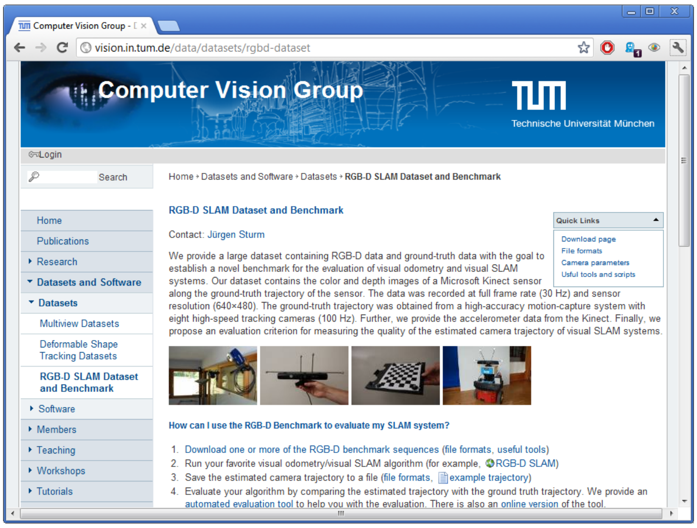
- Need to find transformation between the markers on the Kinect and the optical center
- Special calibration board visible both by Kinect and mocap system (manually gauged)



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




Computer Vision Group - [
vision.in.tum.de/data/datasets/rgbd-dataset
Computer Vision Group
Technische Universität München
Login
Search
Home
Publications
Research
Datasets and Software
Datasets
Multiview Datasets
Deformable Shape Tracking Datasets
RGB-D SLAM Dataset and Benchmark
Software
Members
Teaching
Workshops
Tutorials

Home › Datasets and Software › Datasets › **RGB-D SLAM Dataset and Benchmark**

RGB-D SLAM Dataset and Benchmark

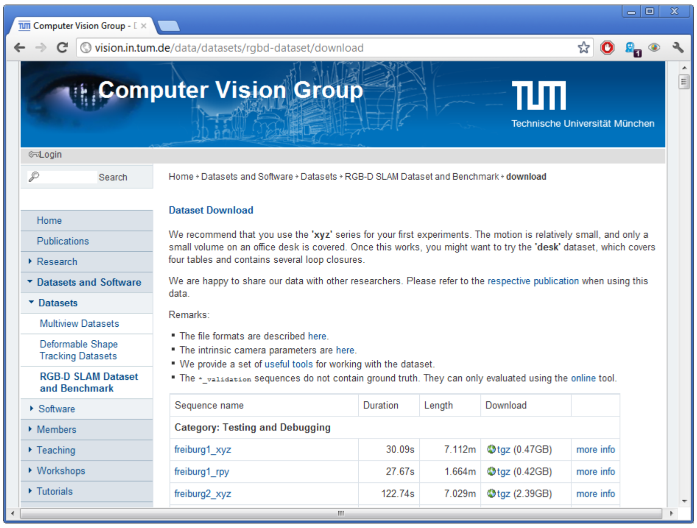
Quick Links
- Download page
- File formats
- Camera parameters
- Usful tools and scripts

Contact: Jürgen Sturm

We provide a large dataset containing RGB-D data and ground-truth data with the goal to establish a novel benchmark for the evaluation of visual odometry and visual SLAM systems. Our dataset contains the color and depth images of a Microsoft Kinect sensor along the ground-truth trajectory of the sensor. The data was recorded at full frame rate (30 Hz) and sensor resolution (640×480). The ground-truth trajectory was obtained from a high-accuracy motion-capture system with eight high-speed tracking cameras (100 Hz). Further, we provide the accelerometer data from the Kinect. Finally, we propose an evaluation criterion for measuring the quality of the estimated camera trajectory of visual SLAM systems.

How can I use the RGB-D Benchmark to evaluate my SLAM system?

1. Download one or more of the RGB-D benchmark sequences (file formats, useful tools)
2. Run your favorite visual odometry/visual SLAM algorithm (for example, RGB-D SLAM)
3. Save the estimated camera trajectory to a file (file formats, example trajectory)
4. Evaluate your algorithm by comparing the estimated trajectory with the ground truth trajectory. We provide an automated evaluation tool to help you with the evaluation. There is also an online version of the tool.


Computer Vision Group - [
vision.in.tum.de/data/datasets/rgbd-dataset/download
Computer Vision Group
Technische Universität München
Login
Search
Home
Publications
Research
Datasets and Software
Datasets
Multiview Datasets
Deformable Shape Tracking Datasets
RGB-D SLAM Dataset and Benchmark
Software
Members
Teaching
Workshops
Tutorials

Home › Datasets and Software › Datasets › RGB-D SLAM Dataset and Benchmark › **download**

Dataset Download

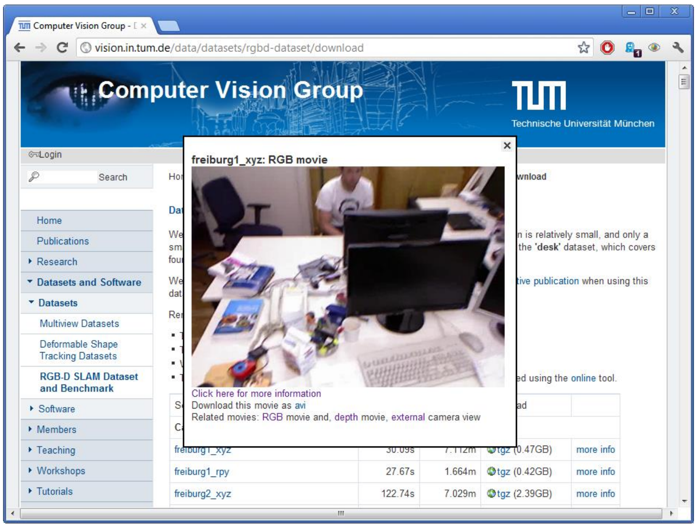
We recommend that you use the **'xyz'** series for your first experiments. The motion is relatively small, and only a small volume on an office desk is covered. Once this works, you might want to try the **'desk'** dataset, which covers four tables and contains several loop closures.

We are happy to share our data with other researchers. Please refer to the respective publication when using this data.

Remarks:

- The file formats are described here.
- The intrinsic camera parameters are here.
- We provide a set of useful tools for working with the dataset.
- The `*_validation` sequences do not contain ground truth. They can only evaluated using the online tool.

Sequence name	Duration	Length	Download	
Category: Testing and Debugging				
freiburg1_xyz	30.09s	7.112m	tgz (0.47GB)	more info
freiburg1_rpy	27.67s	1.664m	tgz (0.42GB)	more info
freiburg2_xyz	122.74s	7.029m	tgz (2.39GB)	more info


Computer Vision Group - [
vision.in.tum.de/data/datasets/rgbd-dataset/download
Computer Vision Group
Technische Universität München
Login
Search
Home
Publications
Research
Datasets and Software
Datasets
Multiview Datasets
Deformable Shape Tracking Datasets
RGB-D SLAM Dataset and Benchmark
Software
Members
Teaching
Workshops
Tutorials

freiburg1_xyz: RGB movie

Click here for more information
Download this movie as avi
Related movies: RGB movie and, depth movie, external camera view

freiburg1_xyz	30.09s	7.112m	tgz (0.47GB)	more info
freiburg1_rpy	27.67s	1.664m	tgz (0.42GB)	more info
freiburg2_xyz	122.74s	7.029m	tgz (2.39GB)	more info

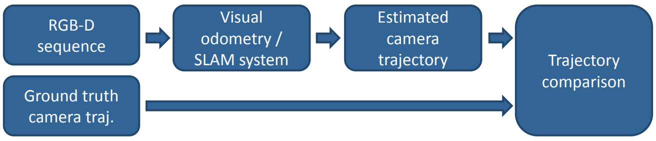
Dataset Website

- 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 $Q_1, \ldots, Q_n \in \mathrm{SE}(3)$
  - Estimate camera trajectory $P_1, \ldots, P_n \in \mathrm{SE}(3)$
- Two common evaluation metrics
  - Relative pose error (drift per second)
  - Absolute trajectory error (global consistency)

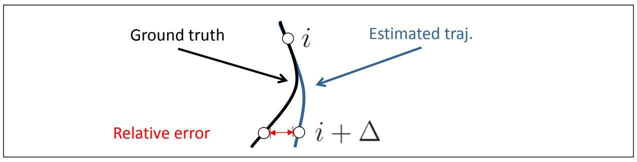

RGB-D sequence
Visual odometry / SLAM system
Estimated camera trajectory
Trajectory comparison
Ground truth camera traj.

Relative Pose Error (RPE)

- Measures the (relative) **drift**
- Recommended for the evaluation of visual odometry approaches

$$E_i := \underbrace{\left(Q_i^{-1} Q_{i+\Delta}\right)^{-1}}_{\text{True motion}} \underbrace{\left(P_i^{-1} P_{i+\Delta}\right)}_{\text{Estimated motion}}$$

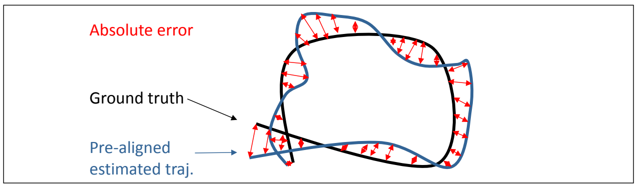
Relative error



Absolute Trajectory Error (ATE)

- Measures the **global error**
- Requires pre-aligned trajectories
- Recommended for SLAM evaluation

$$E_i := Q_i^{-1} S P_i$$

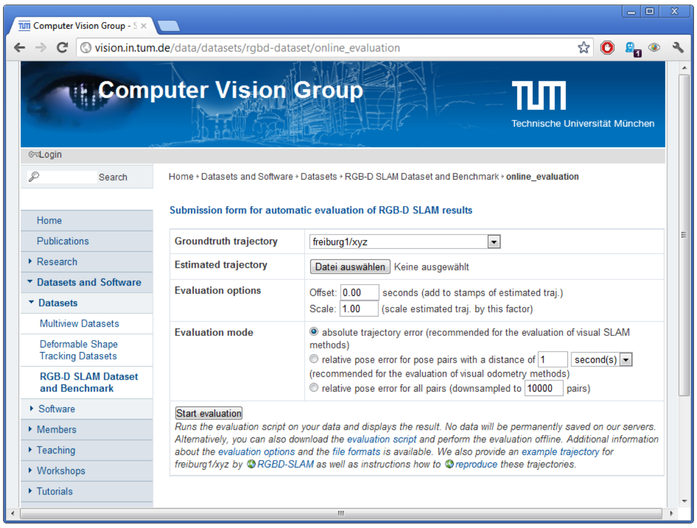


Evaluation metrics

- Average over all time steps

$$\mathrm{RMSE}(E_{1:n}) := \left(\frac{1}{m}\sum_{i=1}^{m} \|trans(E_i)\|^2\right)^{1/2}$$

- Reference implementations for both evaluation metrics available
- Output: RMSE, Mean, Median (as text)
- Plot (png/pdf, optional)


Computer Vision Group - S
vision.in.tum.de/data/datasets/rgbd-dataset/online_evaluation
Computer Vision Group
Technische Universität München
Login
Search
Home
Publications
Research
Datasets and Software
Datasets
Multiview Datasets
Deformable Shape Tracking Datasets
RGB-D SLAM Dataset and Benchmark
Software
Members
Teaching
Workshops
Tutorials

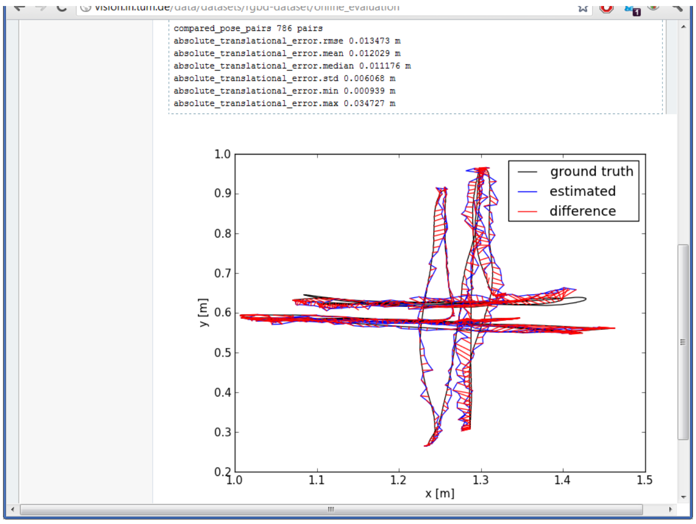
Home › Datasets and Software › Datasets › RGB-D SLAM Dataset and Benchmark › **online_evaluation**

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	freiburg1/xyz
Estimated trajectory	Datei auswählen Keine ausgewählt
Evaluation options	Offset: 0.00 seconds (add to stamps of estimated traj.) Scale: 1.00 (scale estimated traj. by this factor)
Evaluation mode	● absolute trajectory error (recommended for the evaluation of visual SLAM methods) ○ relative pose error for pose pairs with a distance of 1 second(s) (recommended for the evaluation of visual odometry methods) ○ relative pose error for all pairs (downsampled to 10000 pairs)

Start evaluation

Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers. Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM as well as instructions how to reproduce these trajectories.


vision.in.tum.de/data/datasets/rgbd-dataset/online_evaluation

```
compared_pose_pairs 786 pairs
absolute_translational_error.rmse 0.013473 m
absolute_translational_error.mean 0.012029 m
absolute_translational_error.median 0.011176 m
absolute_translational_error.std 0.006068 m
absolute_translational_error.min 0.000939 m
absolute_translational_error.max 0.034727 m
```

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)

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
- How to benchmark SLAM algorithms