Visual Navigation Workshop

Jürgen Sturm and Jakob Engel

Joint work with
Christian Kerl and Daniel Cremers

Welcome

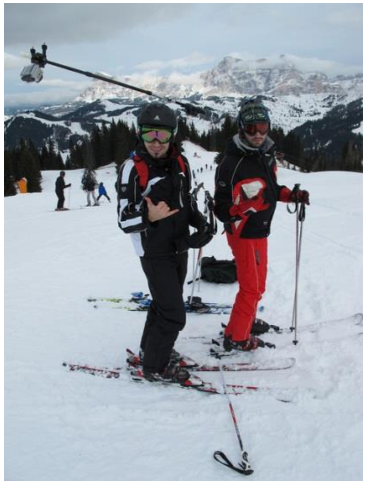
- Morning session
  - Talk: Introduction to quadrocopters
  - Hands-on Session: Manual flight
- Afternoon session
  - Talk: Visual navigation and 3D reconstruction
  - Hands-on Session: Autonomous flight

Motivation of our Research

- Imagine you have a flying camera
- What would you use it for?

Motivation

- Imagine you have a flying camera
- What would you use it for?

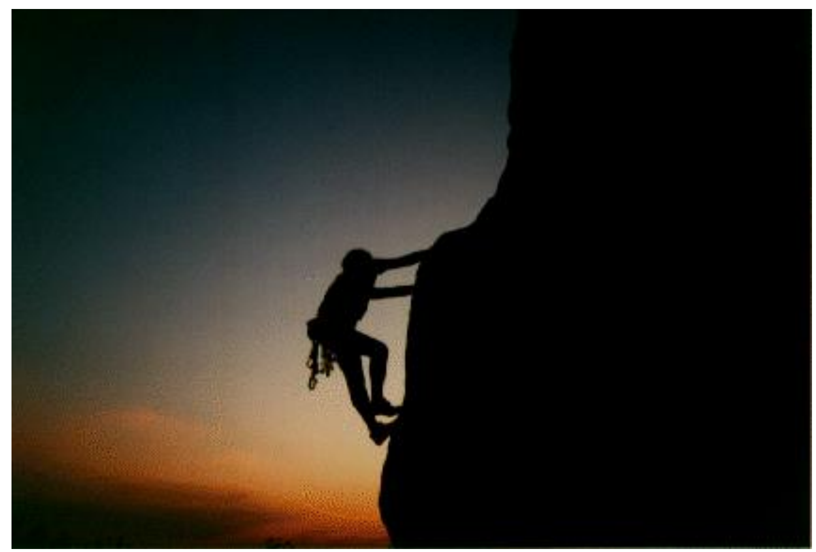



Motivation

- Imagine you have a flying camera
- What would you use it for?

Motivation

- Imagine you have a flying camera
- What would you use it for?

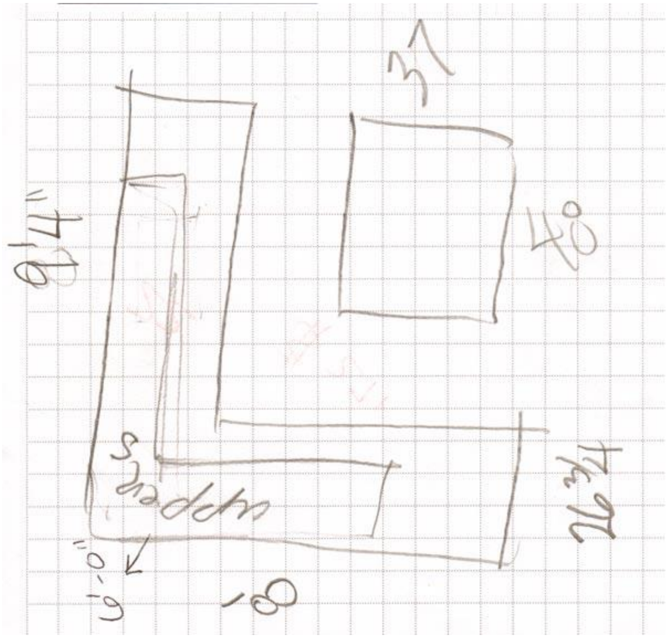


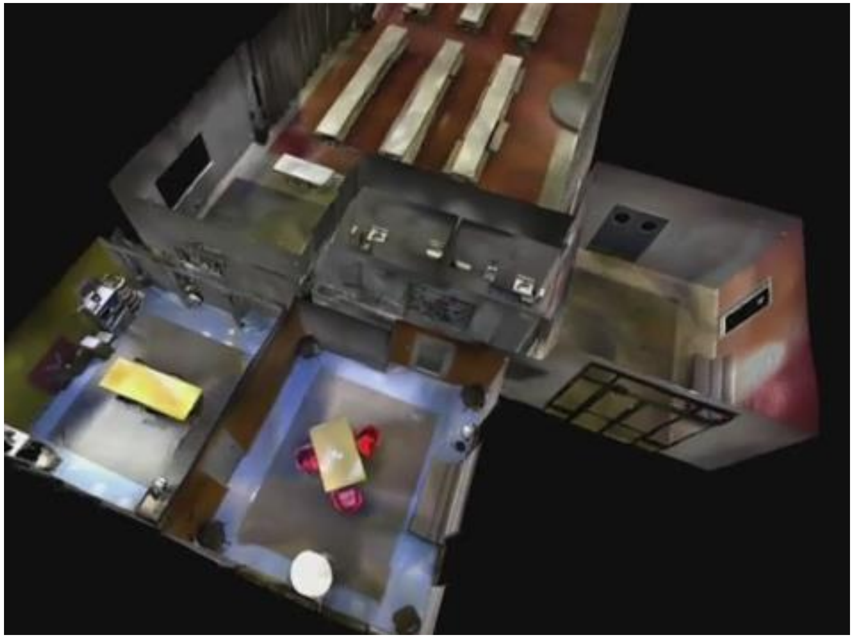
Motivation

- Aerial visual inspection

Motivation

- Mapping of buildings
- Architecture
- Factory planning





Motivation

- Search and rescue missions

Motivation

- Building inspections after earth quakes

Flying Cameras

- Potential:
  - Many useful tasks
  - Large commercial potential

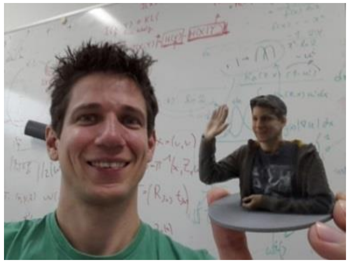
- Challenge:
  - Requires a skilled human pilot
  - High cognitive load
  - Safety and privacy issues

Motivation

- Our research goal:
  Enable flying robots to operate autonomously in 3D environments using onboard cameras
- Use cameras because light weight and rich data
- Navigation, localization, mapping, exploration, people following, …

Who Are We?

- Computer Vision Group at the Technical University of Munich
- 1 professor, 3 postdocs, 11 PhD students
- Research topics:
  - Quadrocopters
  - Kinect / RGB-D
  - 3D reconstruction
  - Image segmentation
  - Convex optimization

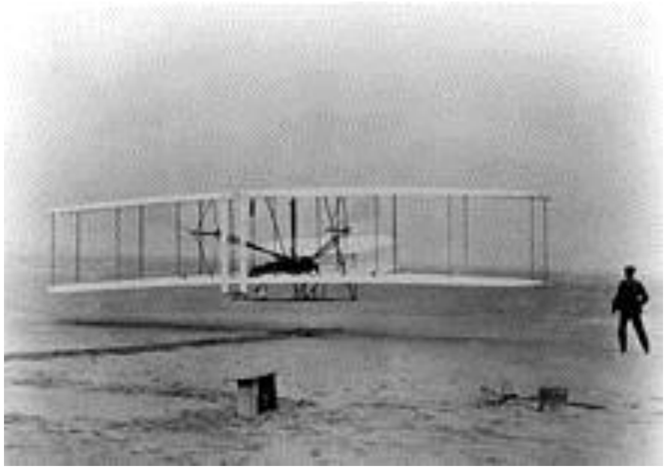


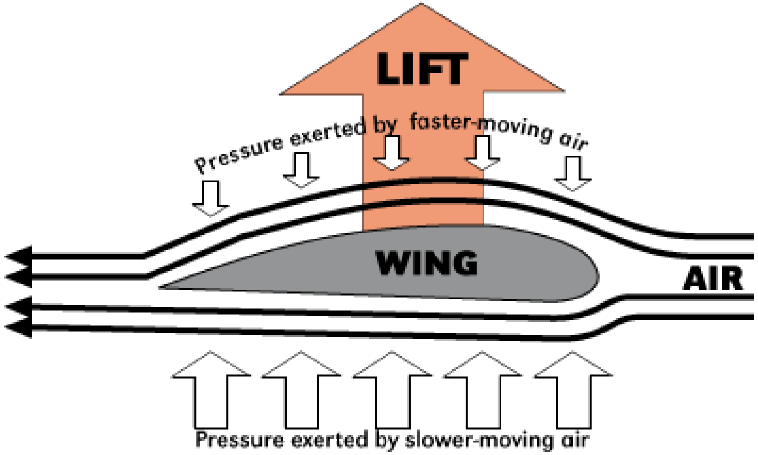
Outline of the Talk

- Morning session
  - Motivation ✓
  - Brief history of aviation
  - Quadrocopter tutorial
- Afternoon session
  - Dense visual odometry
  - Dense mapping
  - Dense SLAM

Fixed-Wing Airplanes

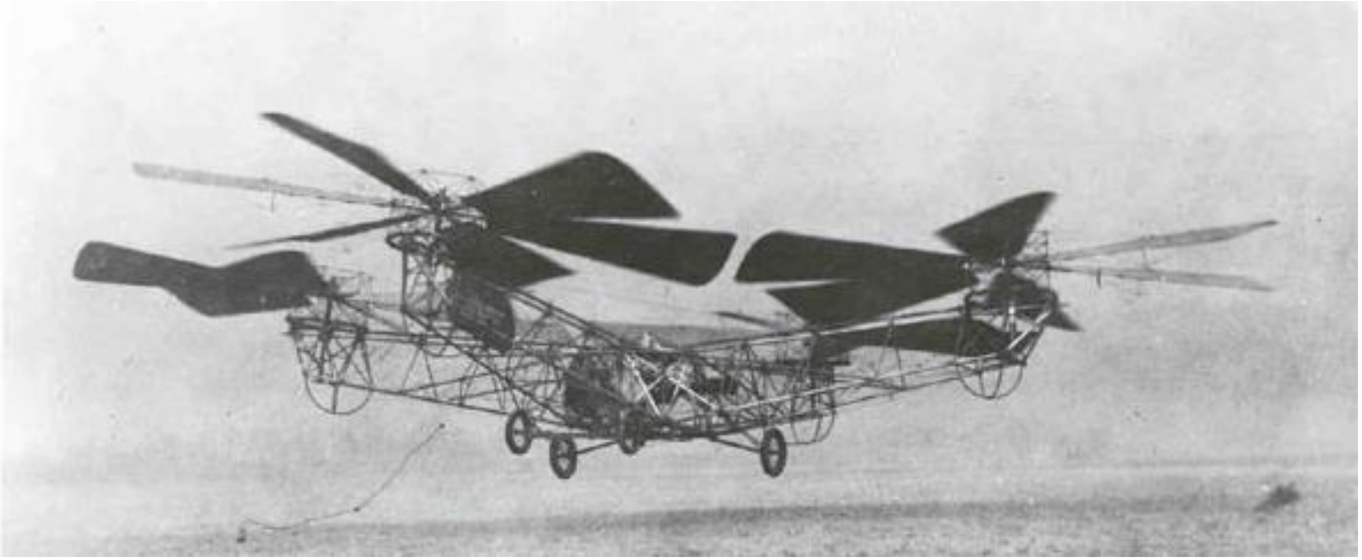
- First motorized flight: 1903 (Wright brothers)
- Generate lift through forward airspeed and the shape of the wings
- Attitude controlled by flaps





Quadrocopters

- First successful flight: 1924
- Vertical take-off and landing (VTOL)
- Problems: stability, control



Helicopters

- First successful flight: 1936
- Swash plate adjusts pitch of propeller cyclically, controls pitch and roll
- Torque is compensated by tail rotor

Micro-Aerial Vehicles (MAVs)

- Attitude stabilization using MEMS sensors
- Remote-controlled quadrocopters
- Renaissance in the early 2000’s

Remote Controlled Flight (2001-)

Video Goggles

Autonomous Quadrocopters

- Initially with external motion capture
- 200-500 fps
- 1mm accuracy

Learning of Flight Parameters

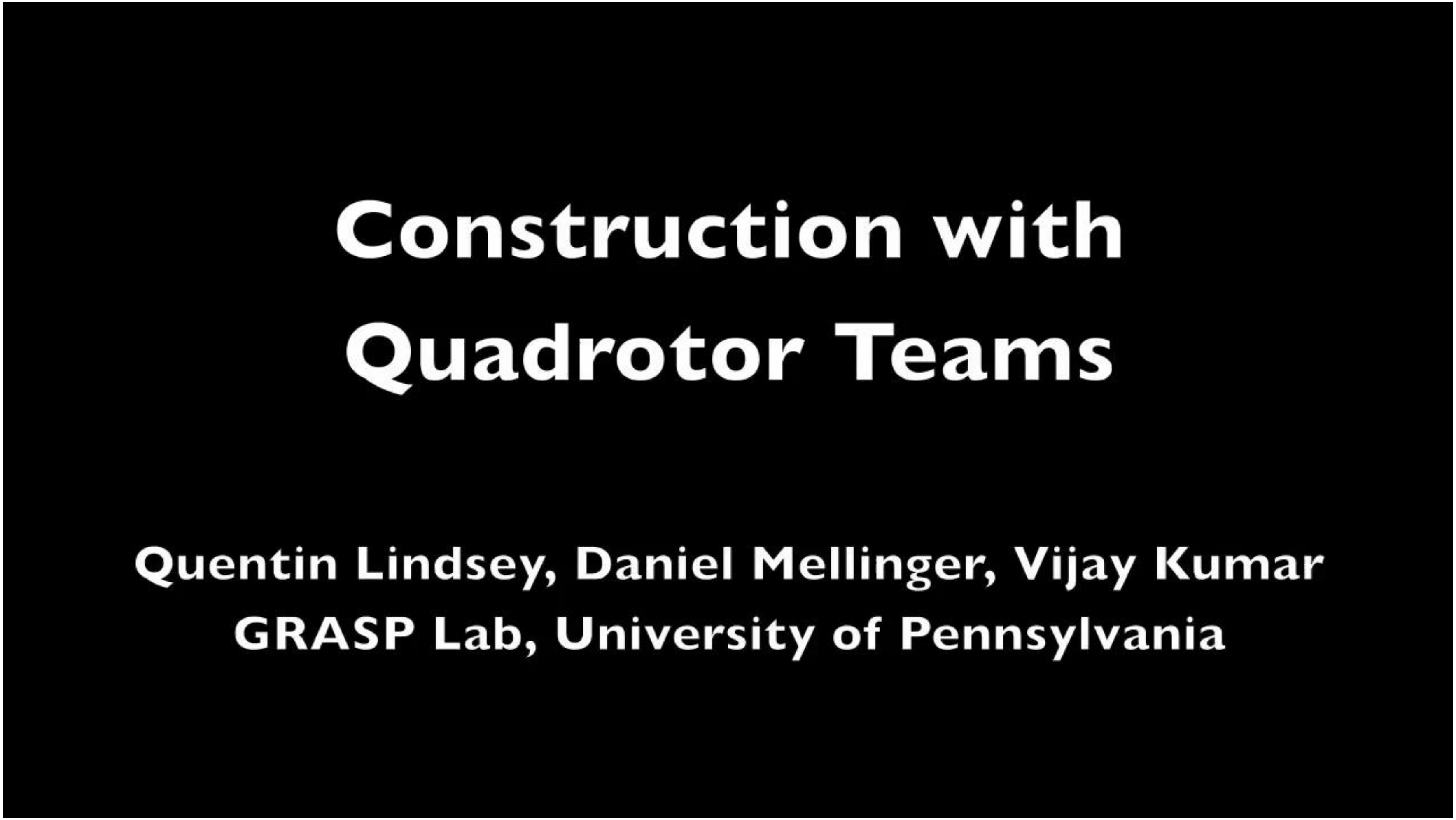
[Schoellig et al., ETH, 2012]

Aggressive Flight Maneuvers

[Mellinger et al., UPenn, 2010]

Aerial Construction

[Lindsey et al., UPenn, 2011]

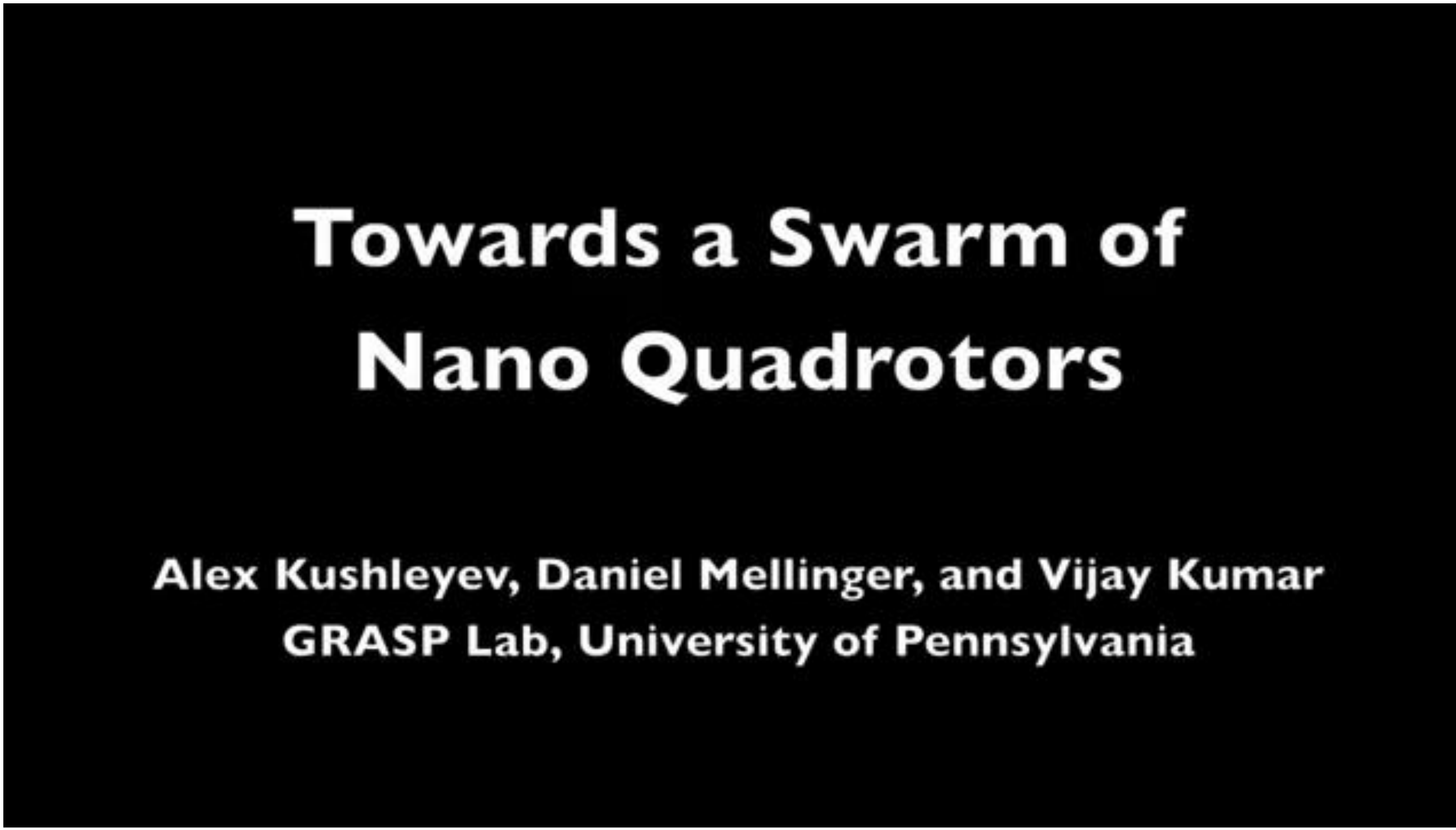


Quadrocopter Ball Juggling

[Müller et al., ETH, 2011]

Miniaturization

[Kushleyev et al., UPenn, 2012]



Interaction using a Kinect
[Ambühl, ETH, 2011]

Camera-Based Navigation

- Very cool results, but external motion capture systems are unpractical
- Is this also possible with onboard sensors?
  - Laser scanner
  - Cameras
  - Kinect

Challenges

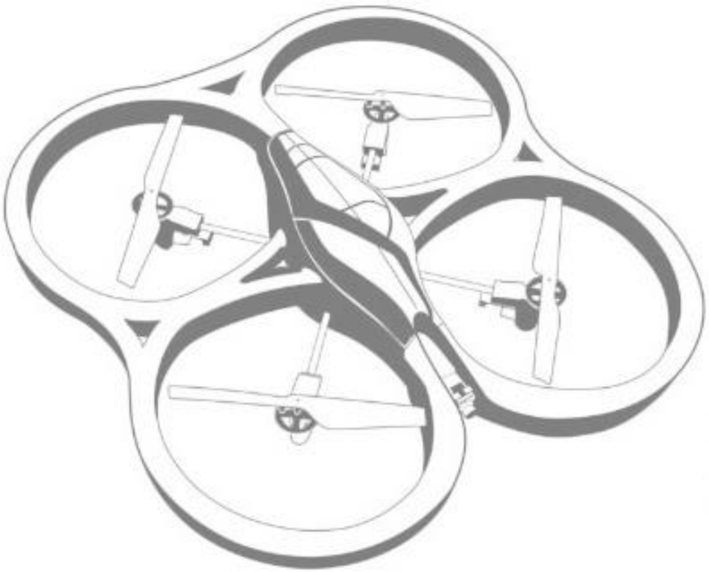
- Limited payload
  - Limited computational power
  - Limited sensors
- Limited battery life
- Fast dynamics, needs electronic stabilization
- Quadrocopter is always in motion
- Safety considerations

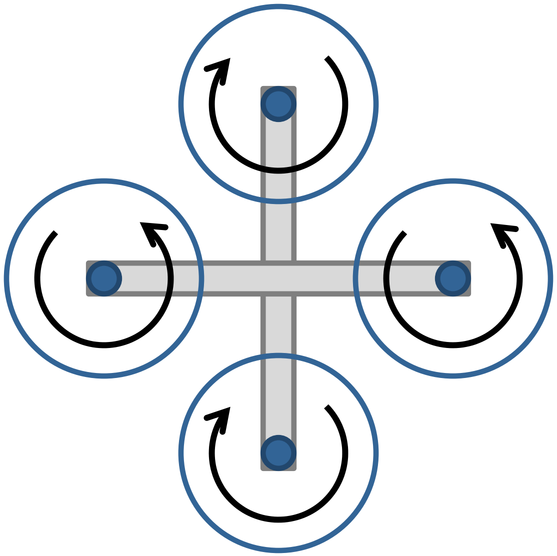
Platform: Parrot Ardrone

- Price: $300
- Controllable via smartphone
- Onboard attitude and drift stabilization

- Sensors
  - Front camera (320x240@18Hz)
  - Ground camera (176x144@18Hz)
  - Gyroscope and accelerometer (IMU)
  - Ultrasound altimeter (height)

Quadrocopter

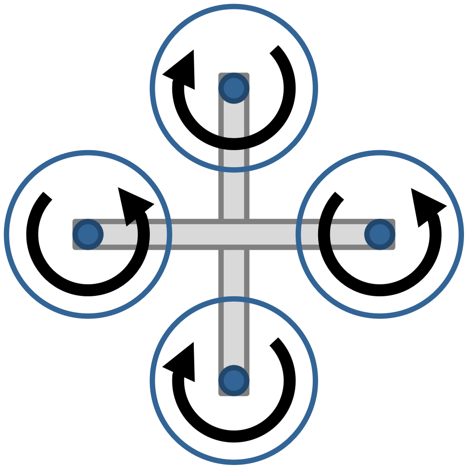




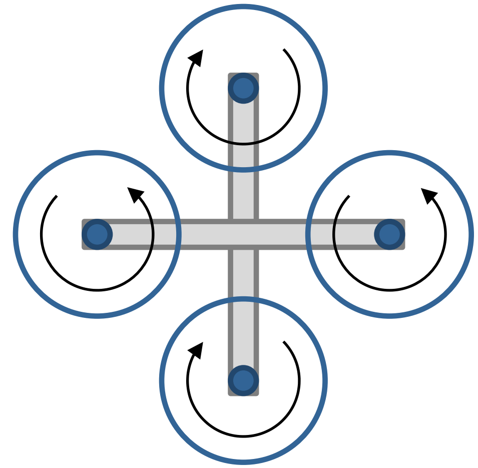
Keep position:

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

Quadrocopter: Basic Motions

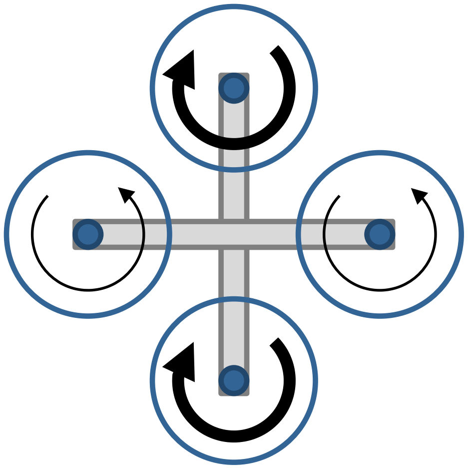


Ascend

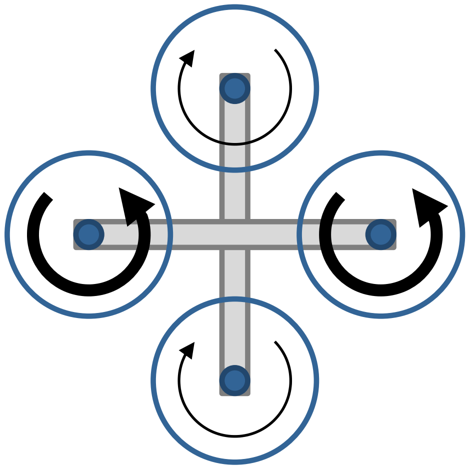


Descend

Quadrocopter: Basic Motions

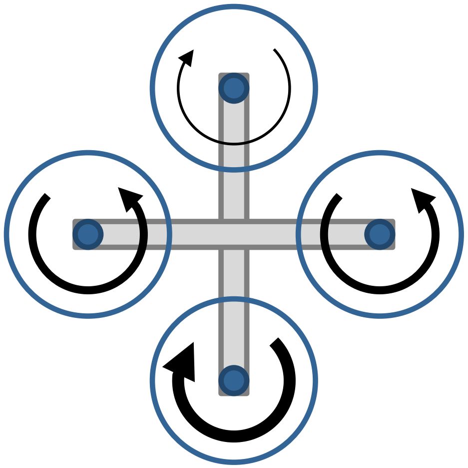


Turn Left

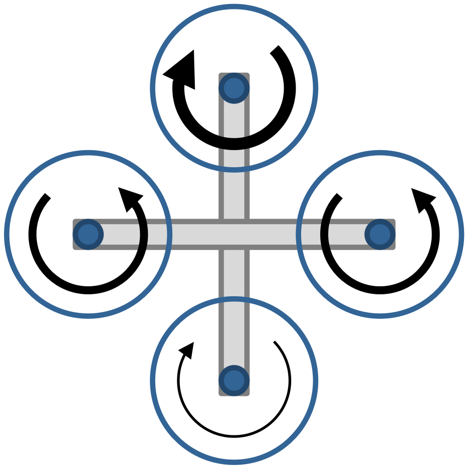


Turn Right

Quadrocopter: Basic Motions

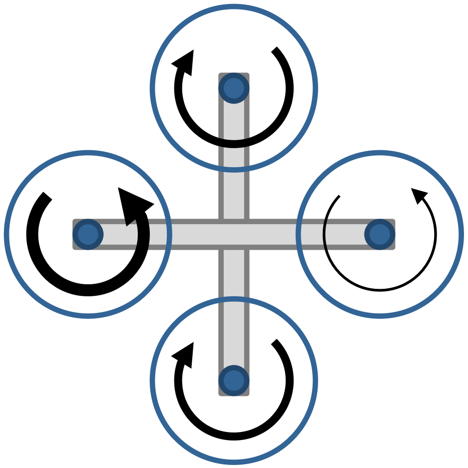


Accelerate Forward

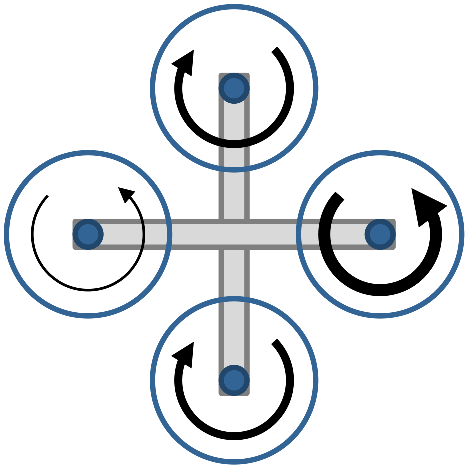


Accelerate Backward

Quadrocopter: Basic Motions



Accelerate
to the Right

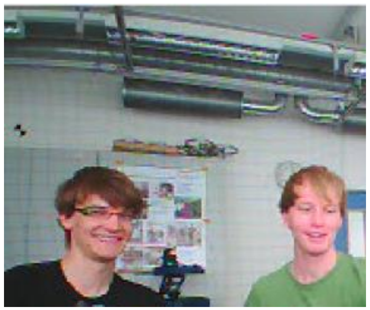


Accelerate
to the Left

Lecture at TUM

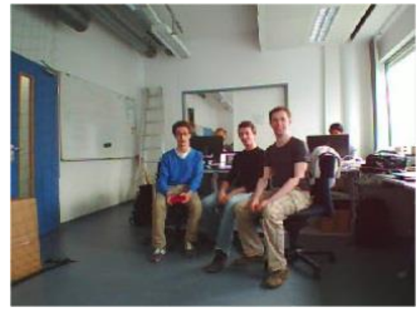
- "Visual Navigation for Flying Robots"
  - State estimation and linear control
  - Mapping, SLAM, 3D reconstruction
  - Obstacle avoidance and path planning
  - Exploration and multi-robot coordination
- Website: http://vision.in.tum.de/
- Lecture recordings, slides, exercises, source code

First Exercise: Self Portrait



Team Brezel

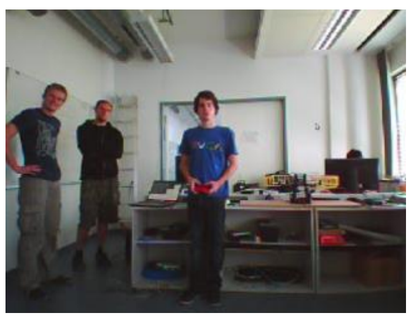
Team Dragonsheep



Team Crash Pilots

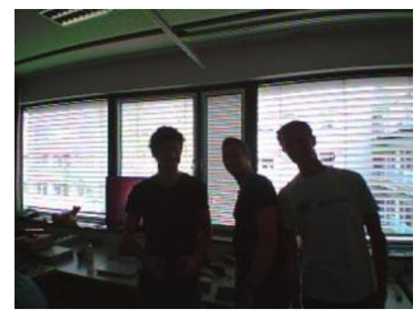
Team Red One

Team Roter Baron



Team Beer

Team Weissbier



Team Weisswurst

Step 1: Manual Flight

- Ardrone
- Laptop
- Joystick
- ROS

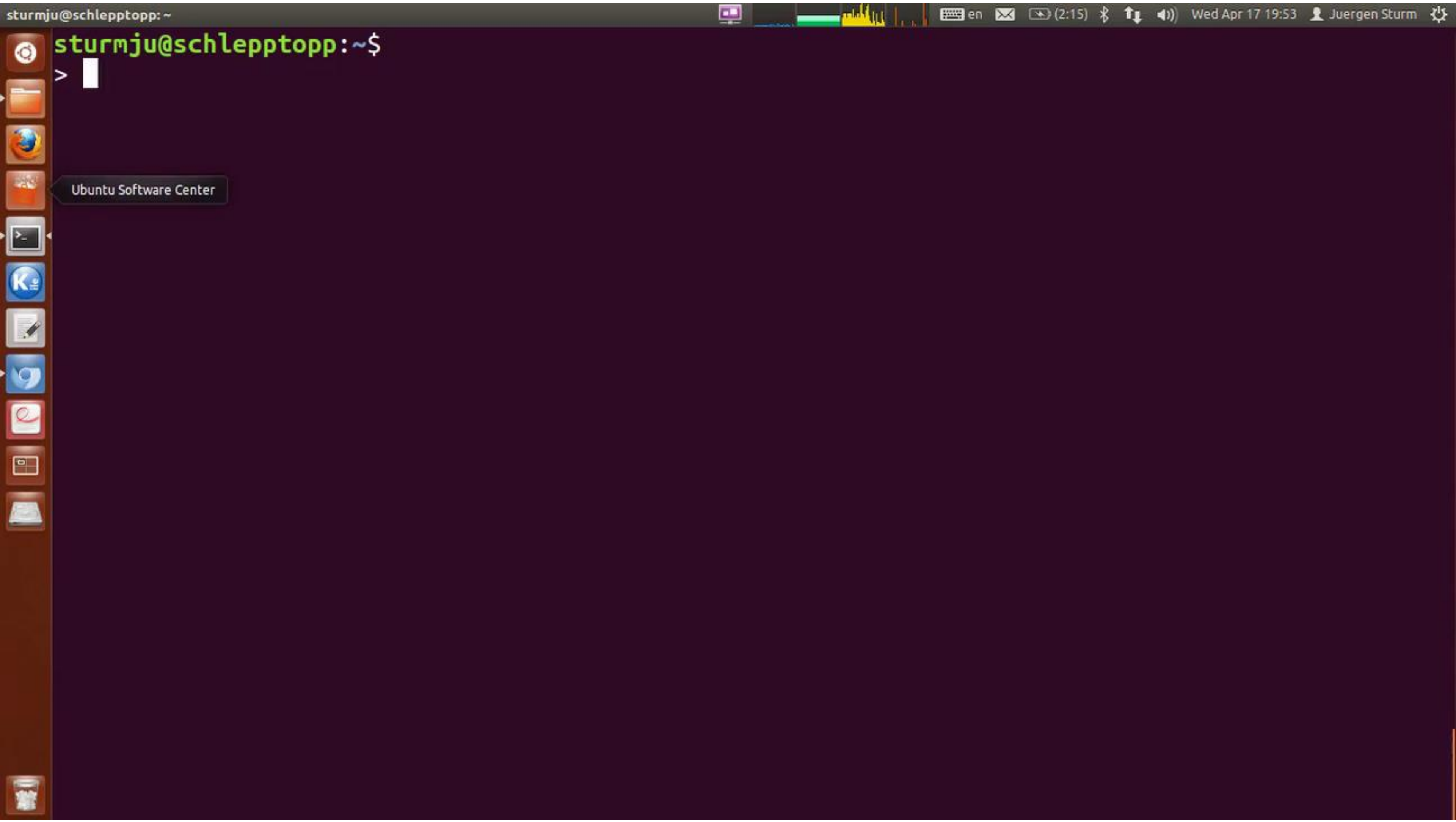
What is ROS?

- **R**obot **O**perating **S**ystem
- Middleware for robots
- Drivers, communication, package management, visualization and debugging tools
- C++, Python, Java, JavaScript, …
- Open Source Robotics Foundation

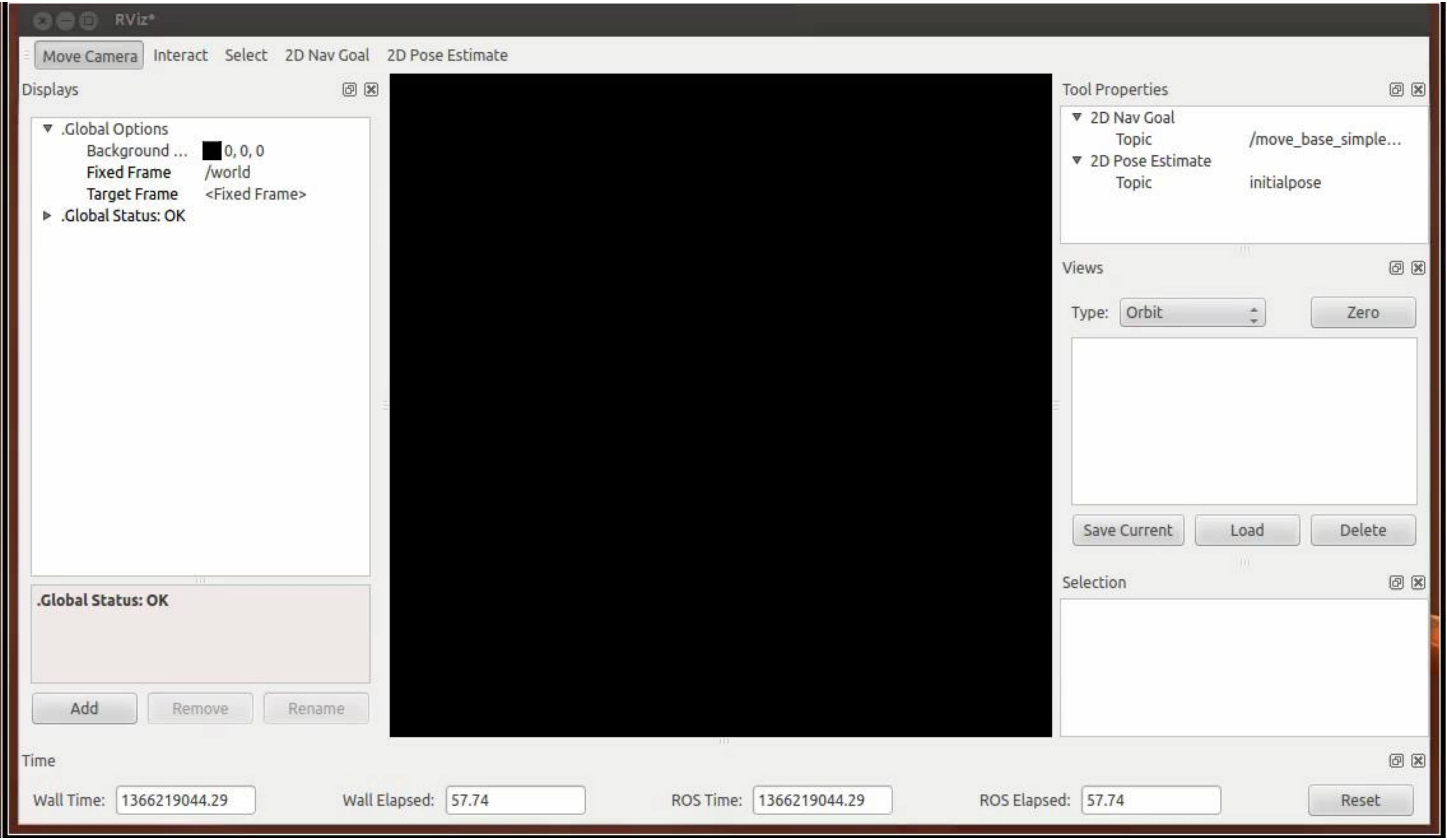
ROS in Numbers

- Currently most widely used robotics middleware
- Support for more than 90 robots
- More than 175 software repositories (universities, research institutes, private developers)
- More than 3500 software packages, mostly BSD licensed

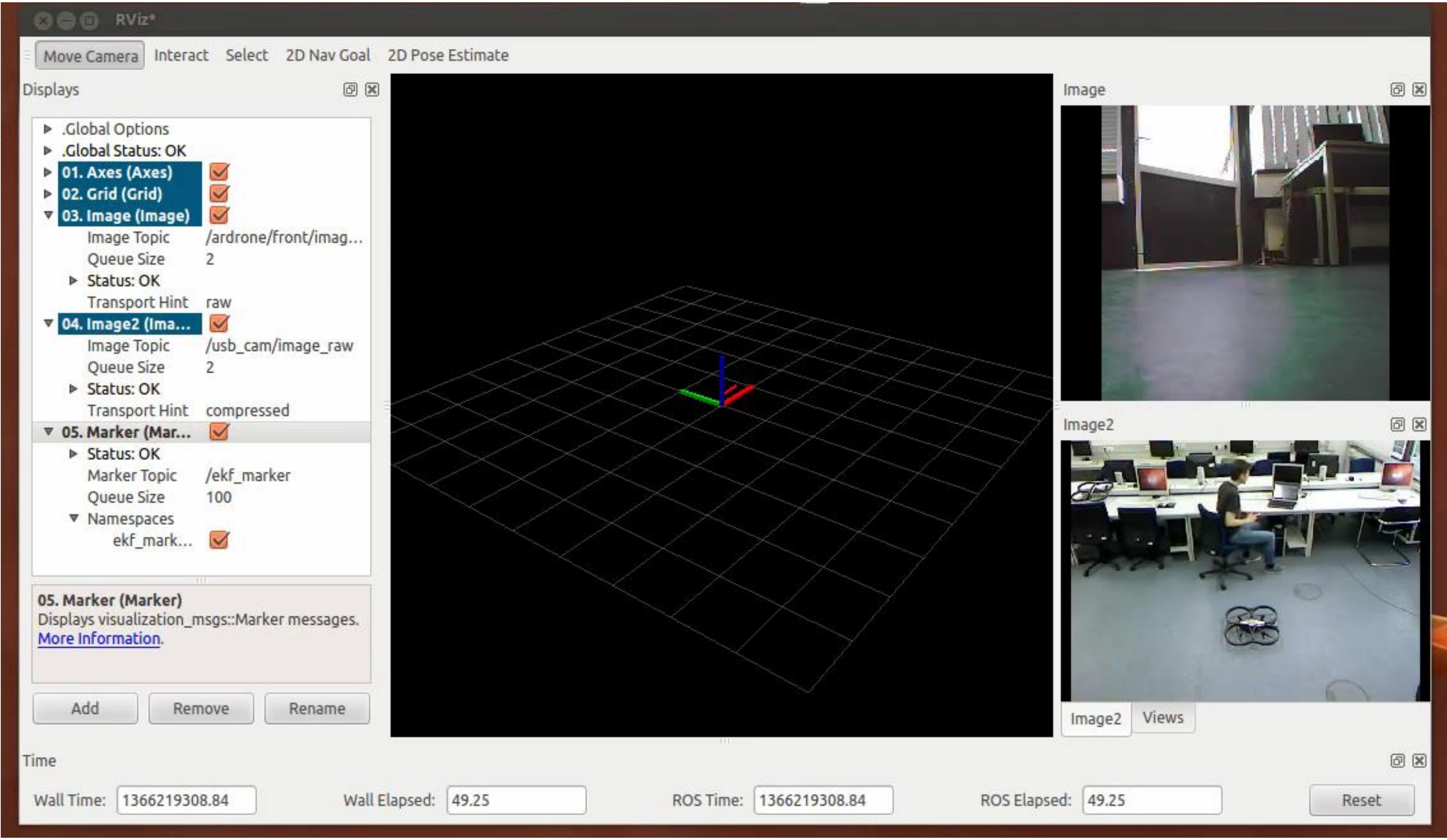
ROS Example


sturmju@schlepptopp: ~
en
(2:15)
Wed Apr 17 19:53
Juergen Sturm
sturmju@schlepptopp:~$
>
Ubuntu Software Center

RVIZ Visualization Tool

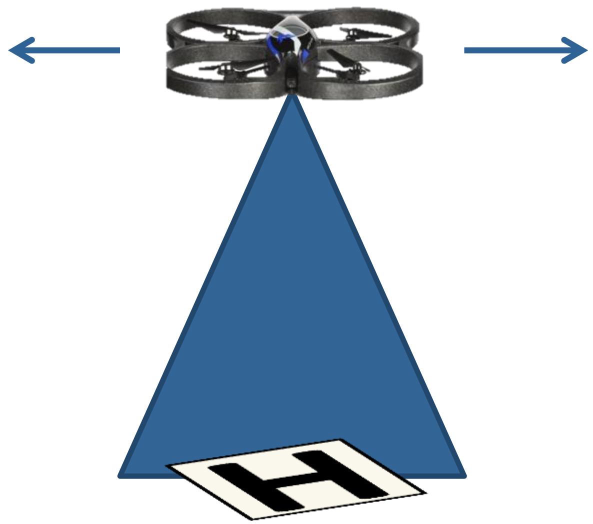


Manual Flight with Ardrone



Camera-based Localization

- The quadrocopter provides
  - Odometry (xy velocities, absolute height)
  - Image stream
- Odometry
  - Subject to drift
- Marker-based localization
  - 3D pose observations
  - Noisy, potentially missing
  - Artoolkit library



Problem Description

Given:

- Odometry readings $\mathbf{u} = (\dot{x}, \dot{y}, z, \phi)$
- Pose observations $\mathbf{z} = (x, y, z, \phi)$

Wanted:

- Estimate robot pose $\mathbf{x} = (x, y, z, \phi)$

How can we estimate the robot pose? What else do we need?

Motion and Observation Models

- Motion model

$$\mathbf{x}_t = g(\mathbf{x}_{t-1}, \mathbf{u}_t)$$

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

- Observation model

$$\mathbf{z}_t = h(\mathbf{x}_t) = \ldots$$

Extended Kalman Filter

For each time step, do

1. Apply motion model

$$\bar{\boldsymbol{\mu}}_t = g(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t)$$

$$\bar{\Sigma}_t = G_t \Sigma G_t^\top + Q \quad \text{with } G_t = \frac{\partial g(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t)}{\partial \boldsymbol{\mu}_{t-1}}$$

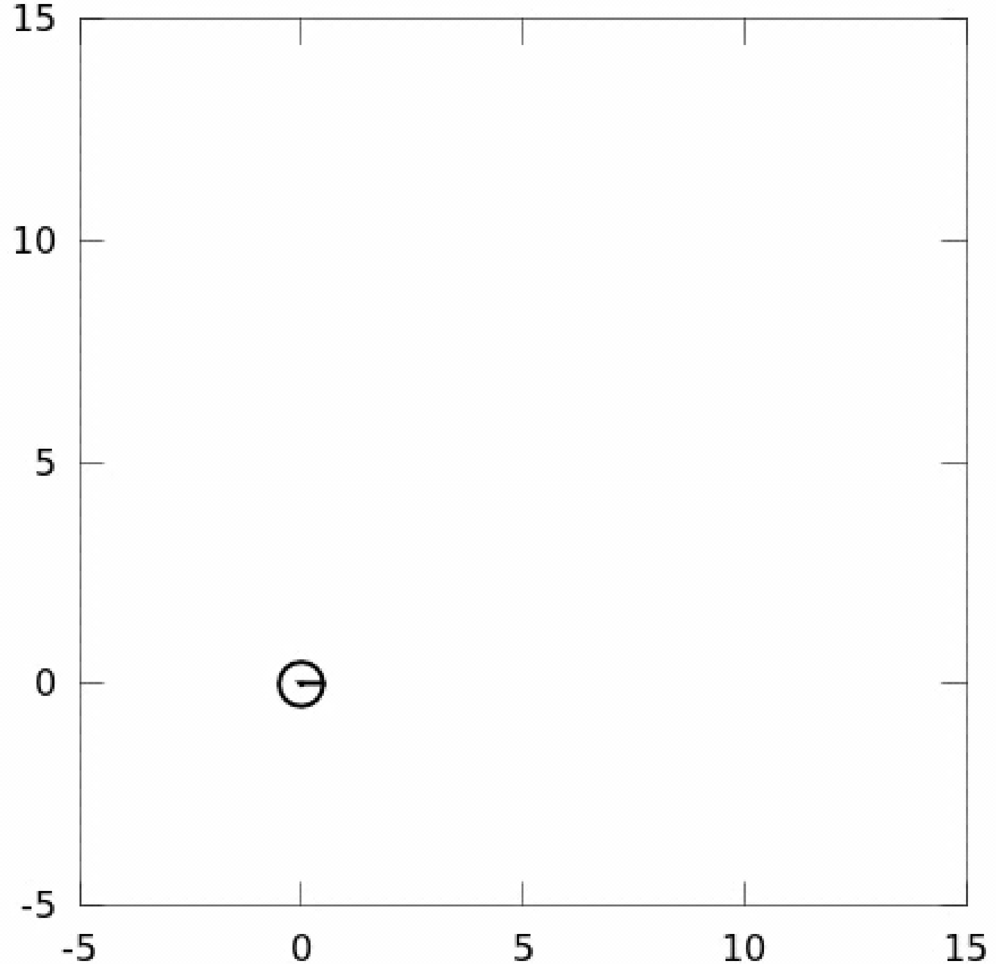
2. Apply sensor model

$$\mu_t = \bar{\boldsymbol{\mu}}_t + K_t(\mathbf{z}_t - h(\bar{\boldsymbol{\mu}}_t))$$

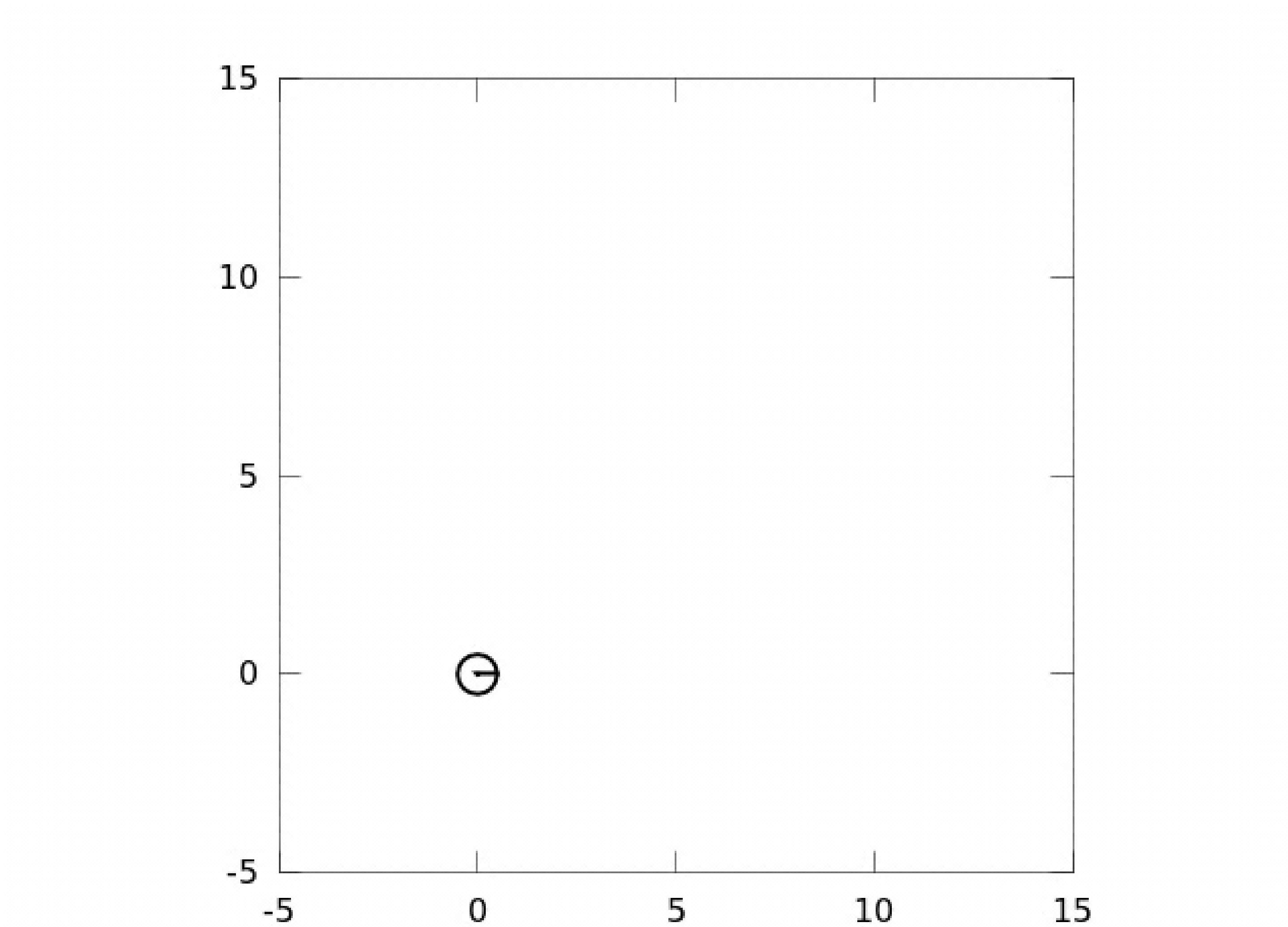
$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$$

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

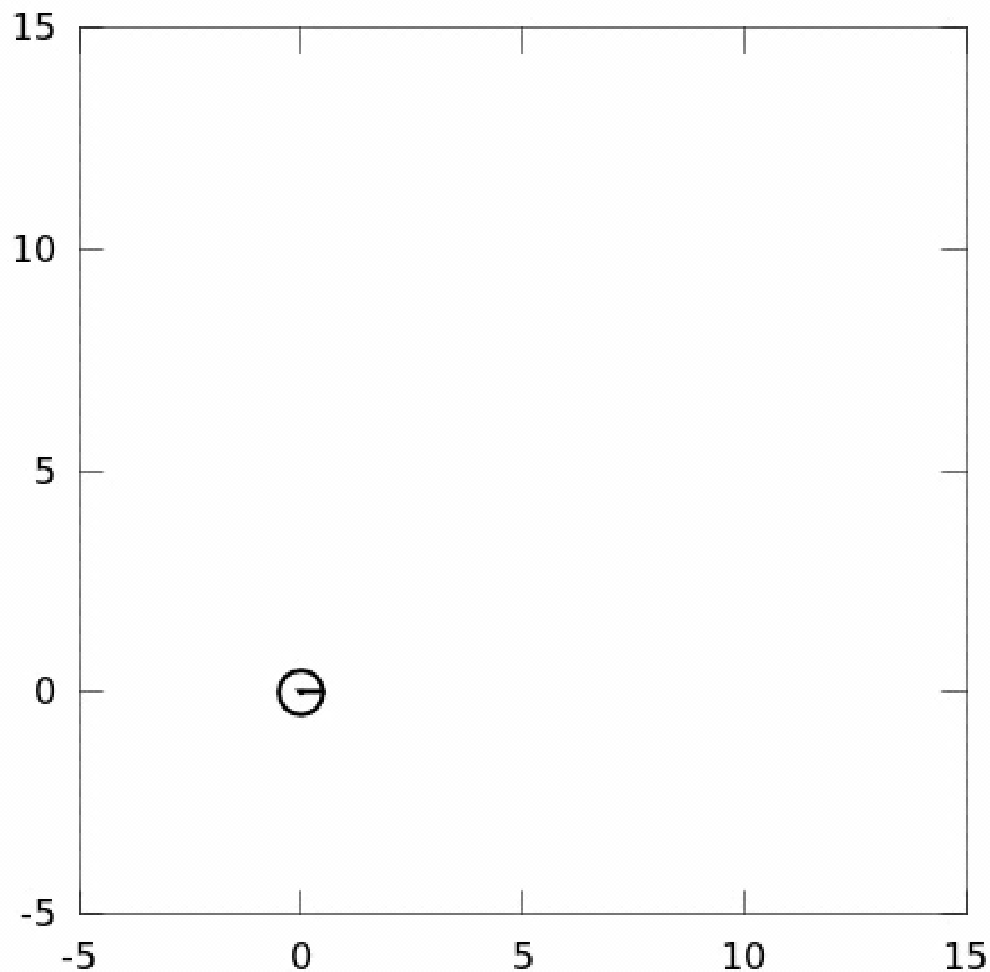
Example: Pure Odometry



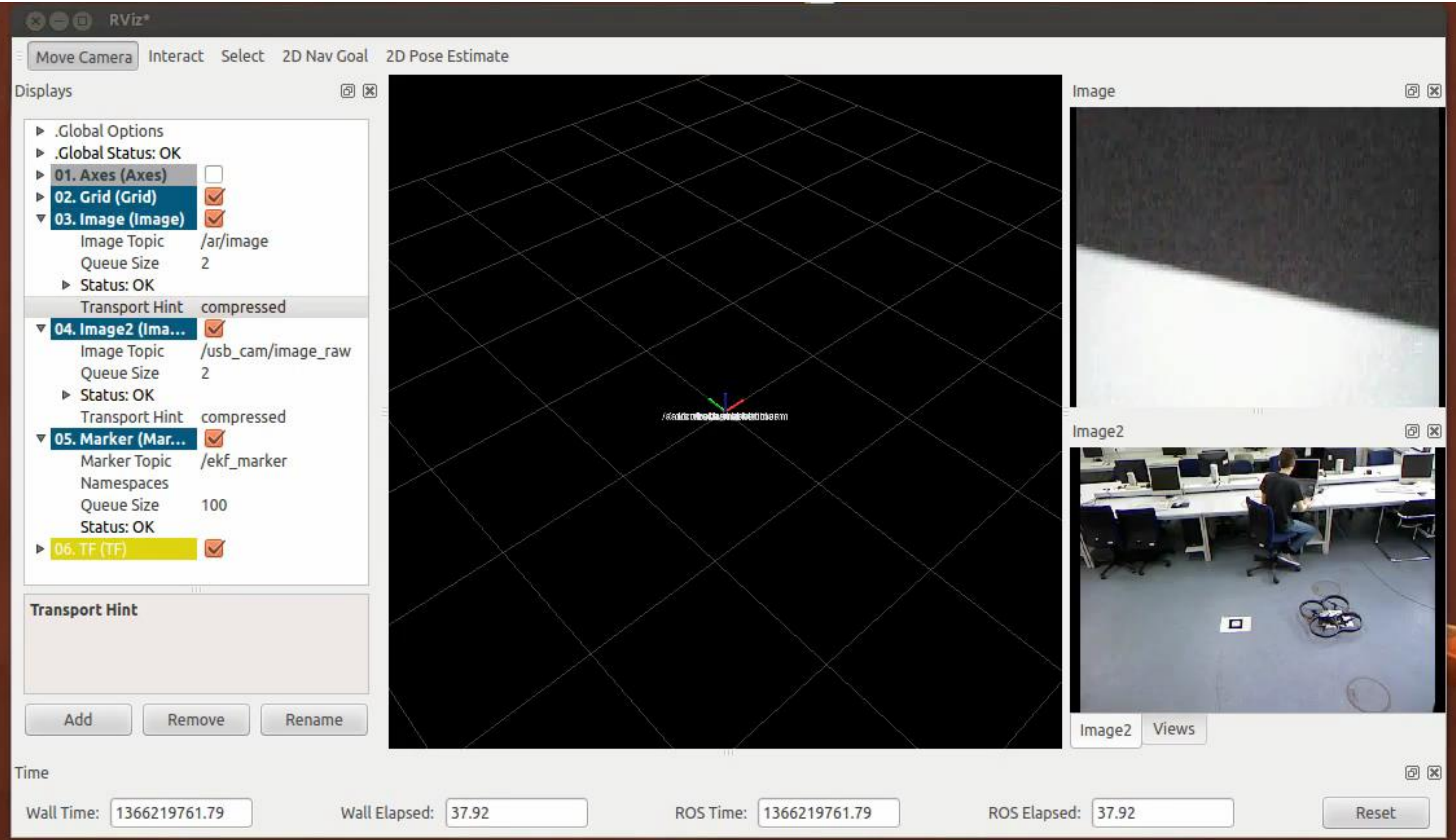
Example: With Landmark



Example: Wrong Initial Pose



Example: Ardrone



Position Control

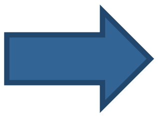
- We have:
  - Estimate of current pose (from EKF)
  - Goal location (from user)
- Which controls do we have to issue to move the robot to the goal?

Feedback Control

- Given:
  - Estimated state (from EKF) $\boldsymbol{\mu}$
  - Goal state $\mathbf{x}_{\text{goal}}$
- Wanted:
  - Control signal $\mathbf{a}$ to reach goal state

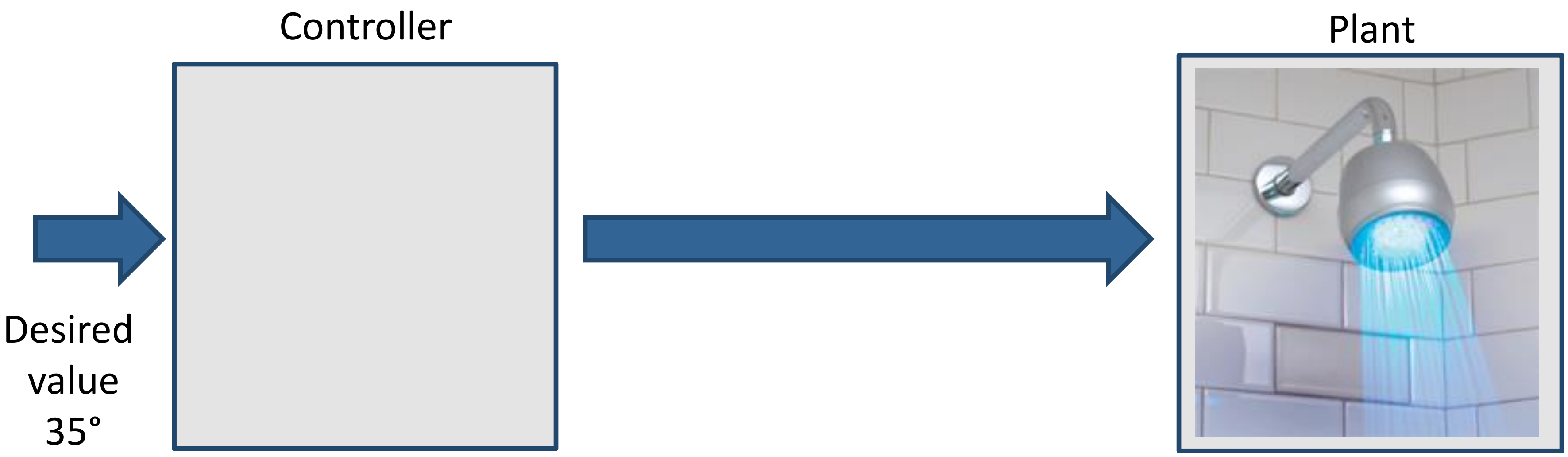
- How to compute the control signal?

Feedback Control - Generic Idea

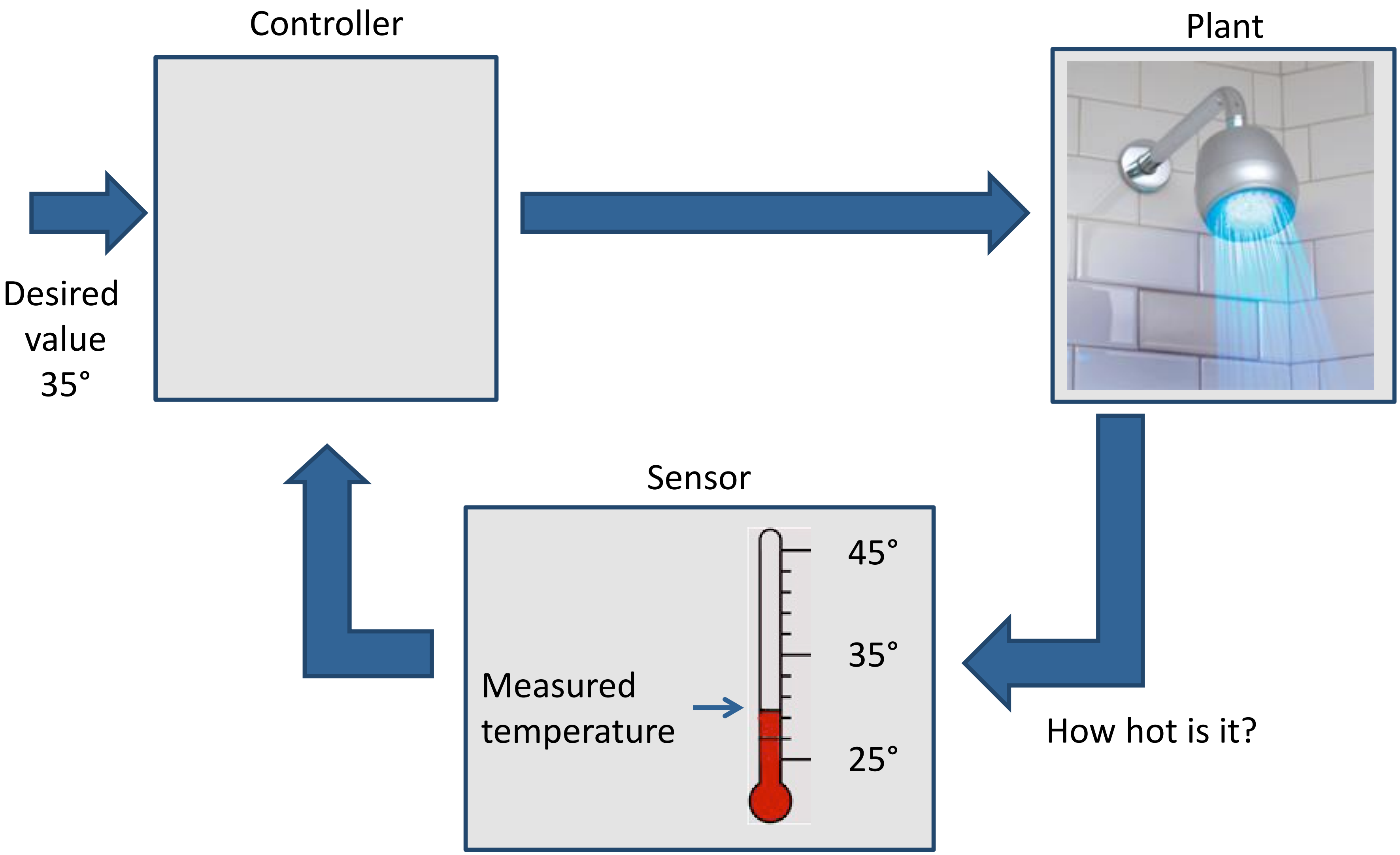


Desired
value
35°

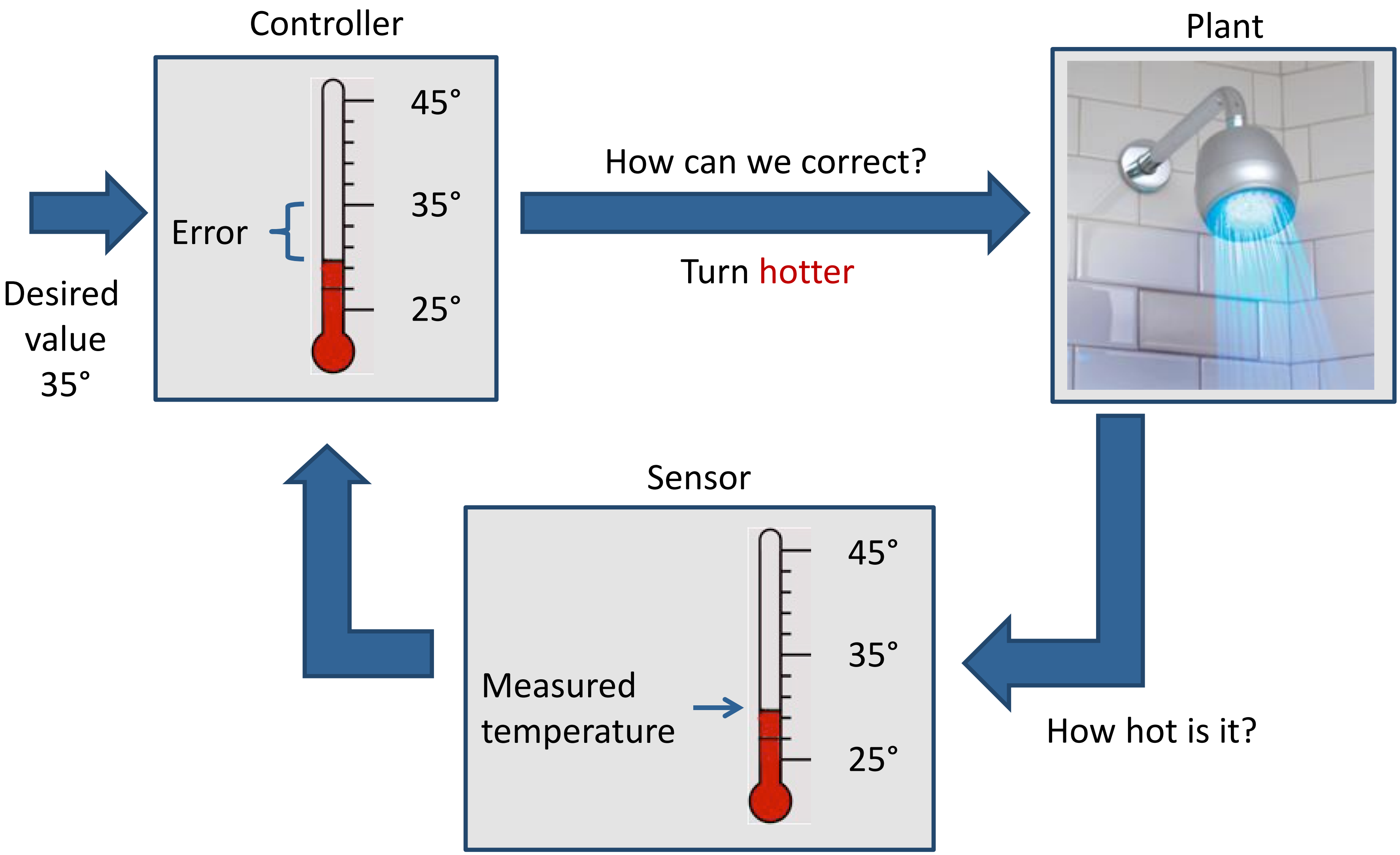
Feedback Control - Generic Idea



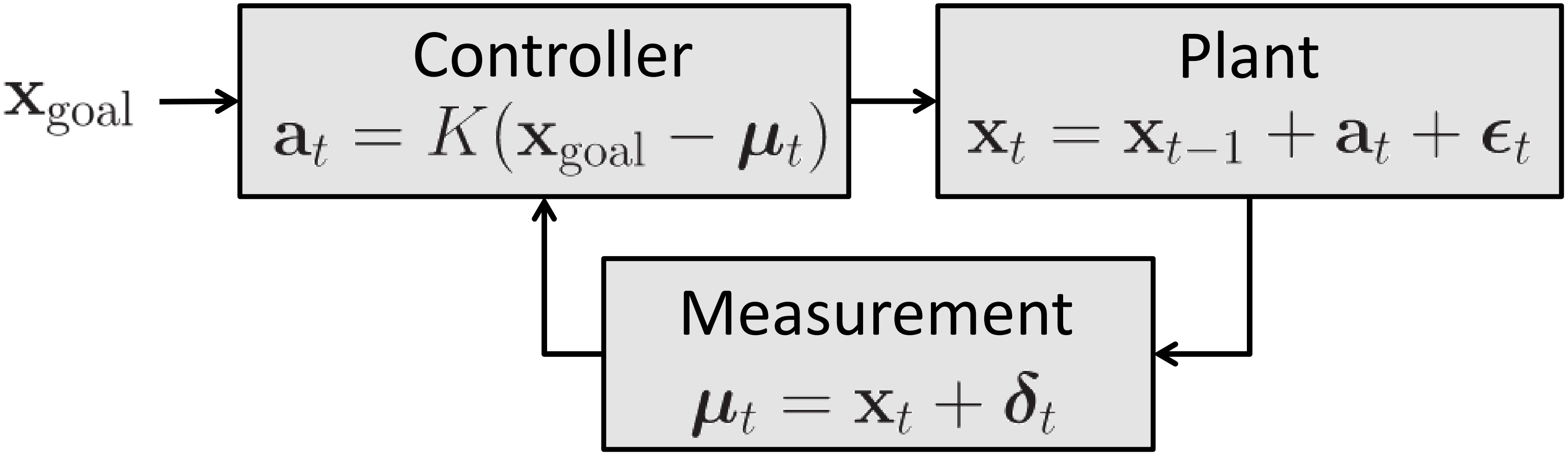
Feedback Control - Generic Idea



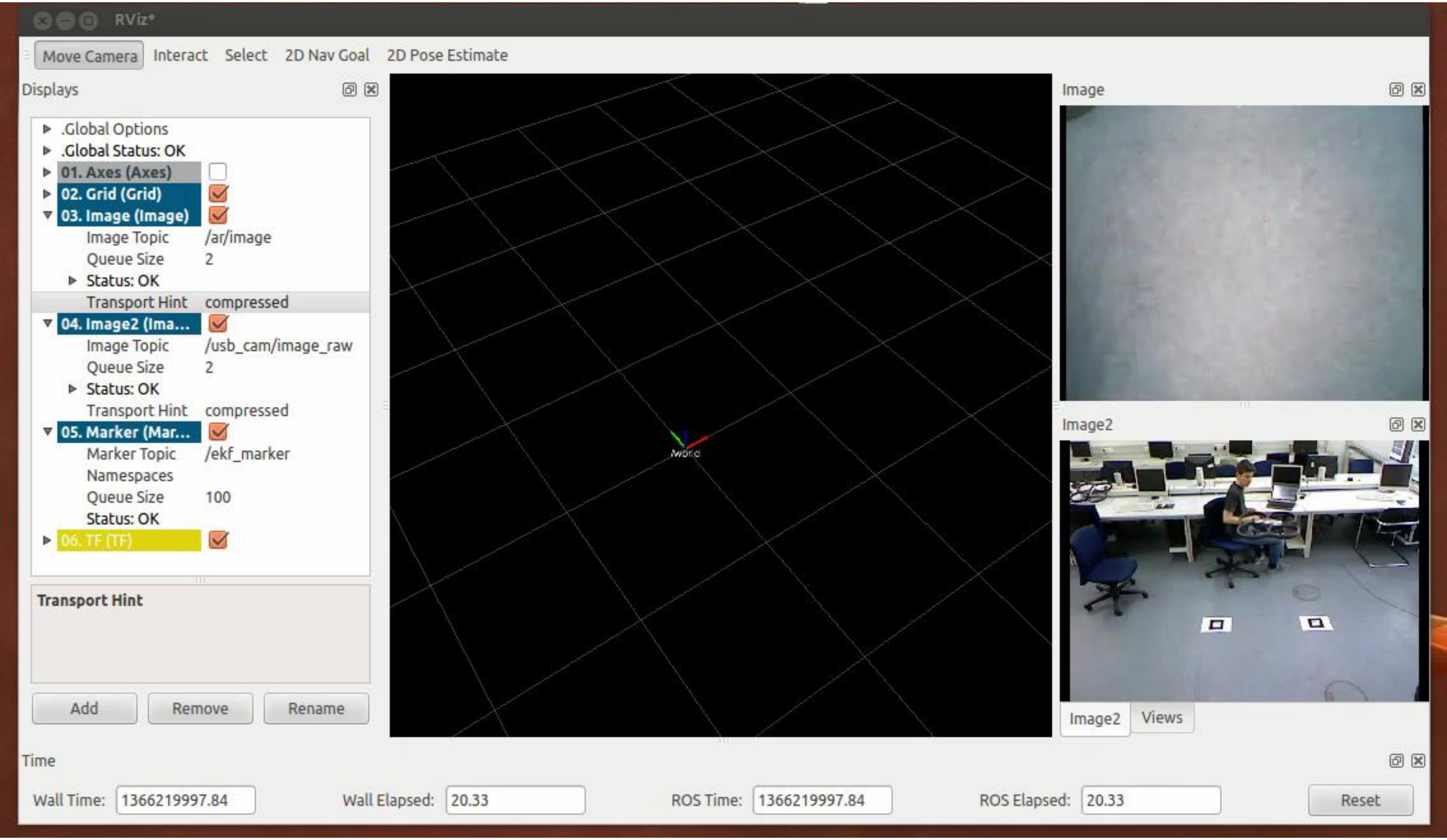
Feedback Control - Generic Idea



P-Control


$\mathbf{x}_{\text{goal}}$
Controller
$\mathbf{a}_t = K(\mathbf{x}_{\text{goal}} - \boldsymbol{\mu}_t)$
Plant
$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{a}_t + \boldsymbol{\epsilon}_t$
Measurement
$\boldsymbol{\mu}_t = \mathbf{x}_t + \boldsymbol{\delta}_t$

P-Control on the Ardrone



Intermediate Result

- Exercise sheets with more information
- Code available (C++)
- Pro:
  - Autonomous, camera-based flight
  - Simple approach
- Con:
  - Needs visual markers
  - Overshoots
- Afternoon session: How to improve on this

Hands-On: Morning Session

- Team up (2-3 persons in each team)
- Goal for the morning: **Manual Flight**
- This includes:
  - Setting up your laptop
  - Connect the Ardrone over wireless
  - Show video stream and navigation data
  - Fly
  - Record cool flight video (or make a self-portrait)

Setup

- Website:
  http://vision.in.tum.de/teaching/ss2013/visnav_sweden
- Software
  - Option 1: VirtualBox + disk image (11GB)
  - Option 2: Ubuntu + ROS + git repository
- Hardware
  - Laptop / computer with WLAN
  - Ardrone, Batteries, Charger
  - PS3 Joystick

Let's go!

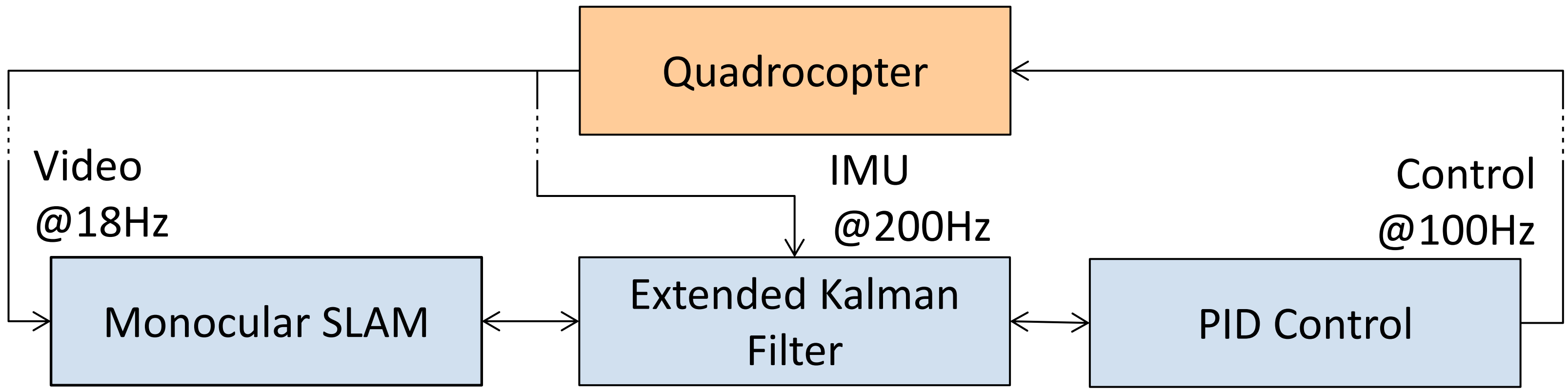
Visual Navigation Workshop
Afternoon Session

Jürgen Sturm

Joint work with Jakob Engel,
Frank Steinbrücker, Christian Kerl, Erik Bylow,
Tayyab Naseer, and Daniel Cremers

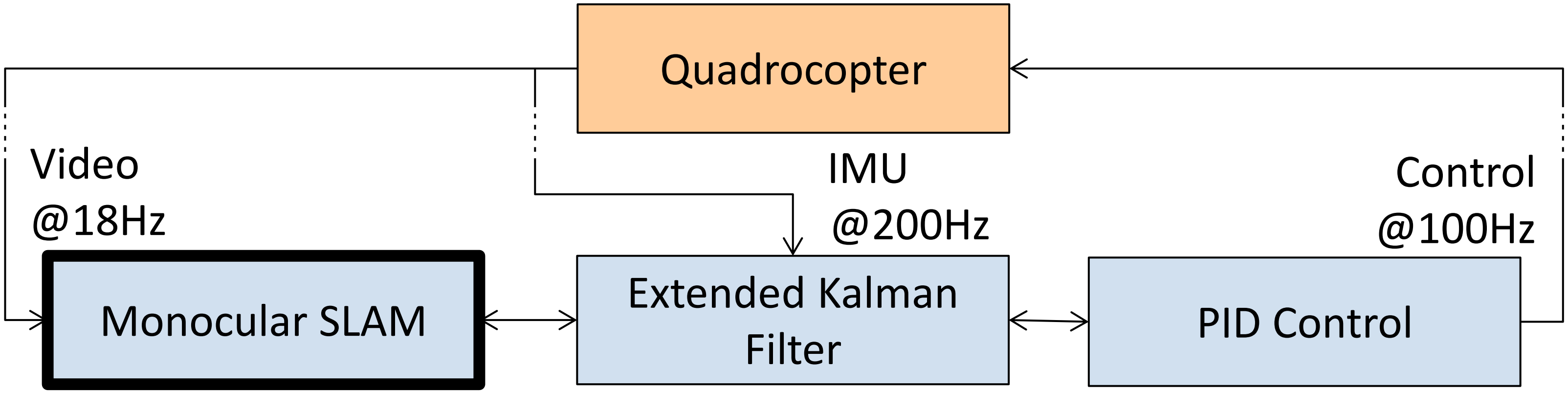
Camera-based Navigation

[Engel, Sturm, Cremers, IROS 2012]

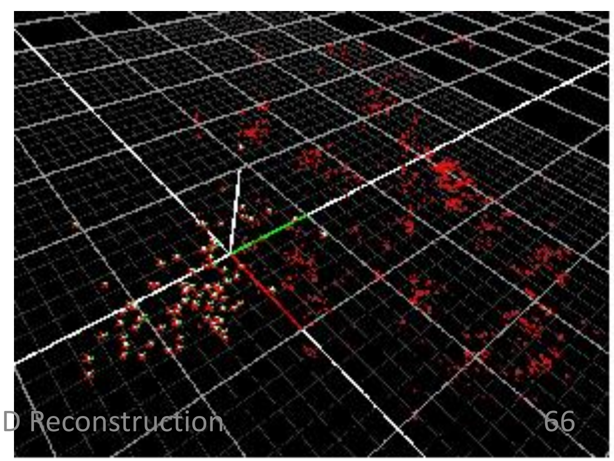
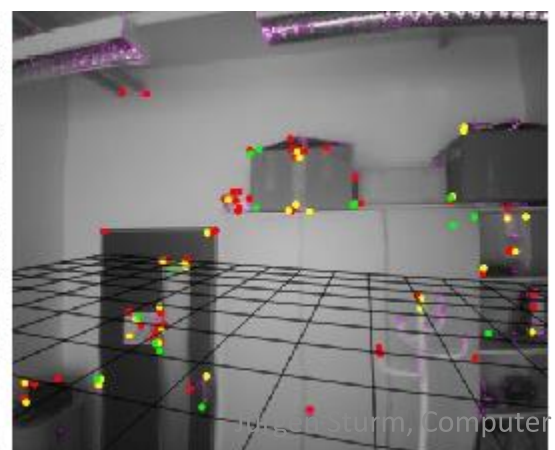


Camera-based Navigation

[Engel, Sturm, Cremers, IROS 2012]

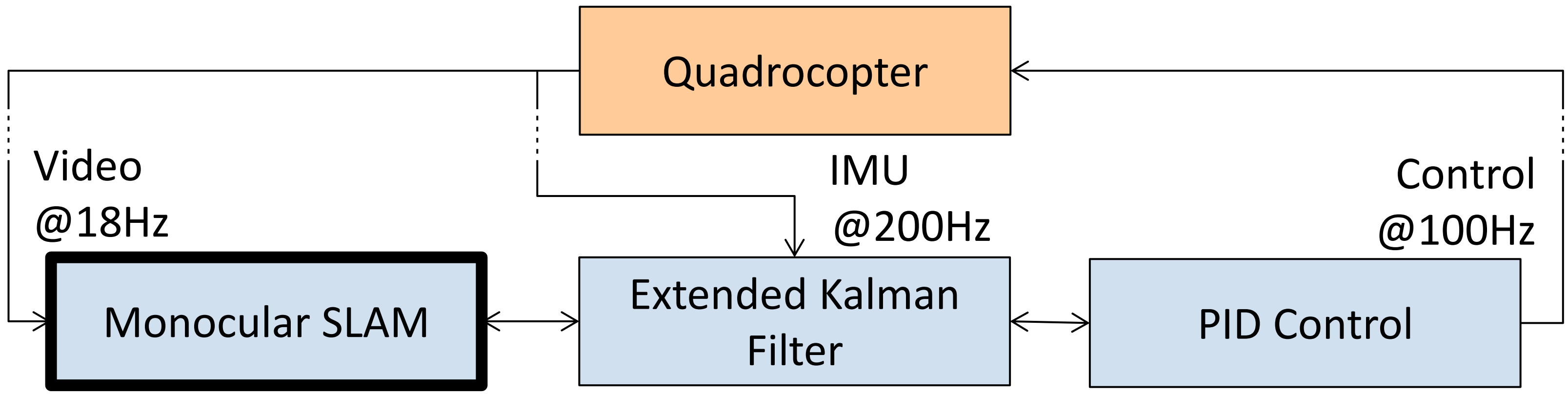


- Based on PTAM [Klein and Murray, ISMAR '07]
  Key-frame based SLAM, efficient, open-source

Camera-based Navigation

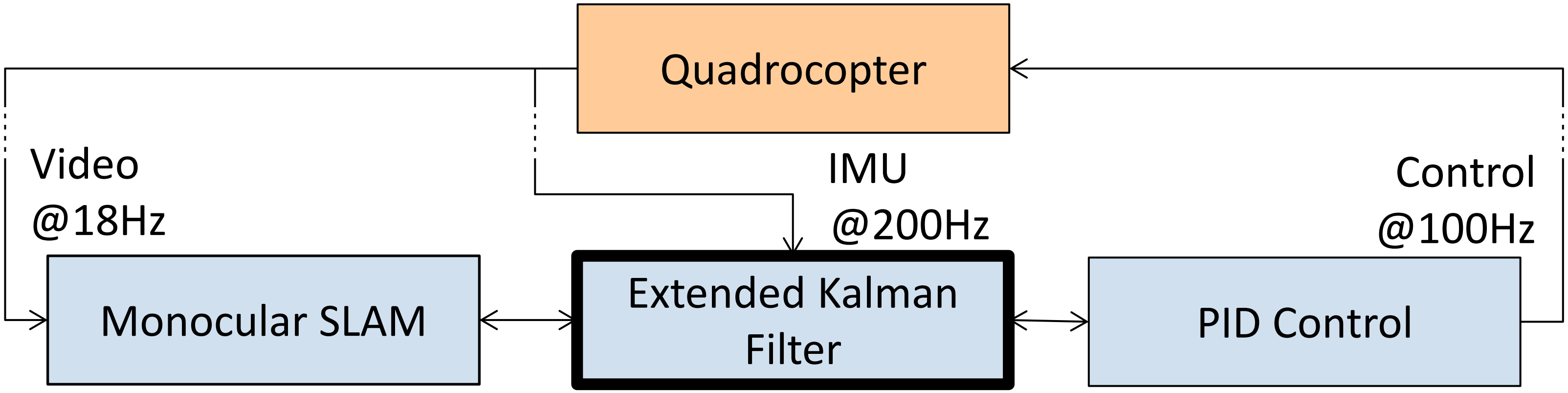
[Engel, Sturm, Cremers, IROS 2012]



- Based on PTAM [Klein and Murray, ISMAR '07] Key-frame based SLAM, efficient, open-source
- Our contributions:
  - **Enhanced reliability** by incorporating IMU into PTAM
  - Maximum likelihood **scale estimation** from ultrasound altimeter and IMU

Camera-based Navigation

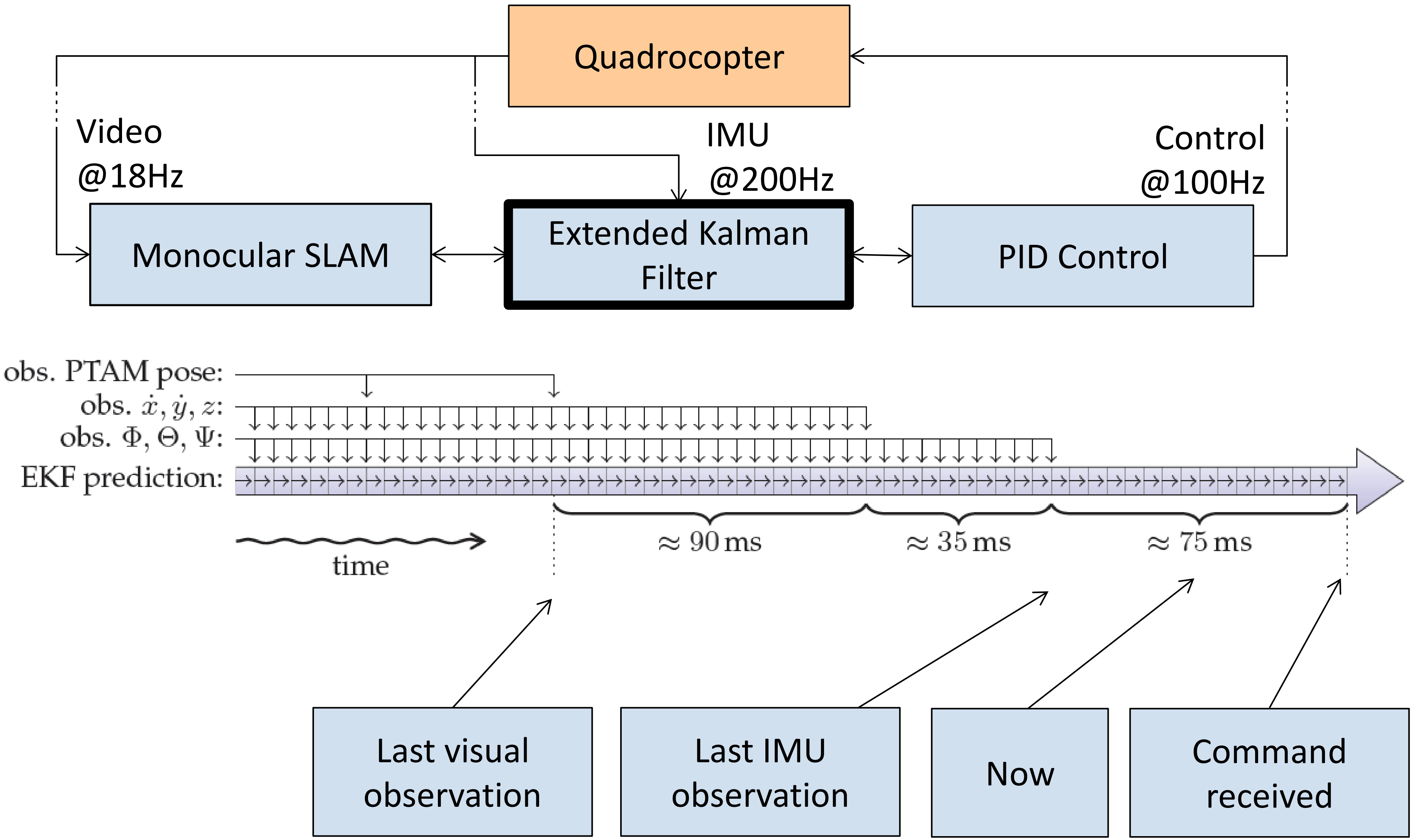
[Engel, Sturm, Cremers, IROS 2012]



- Input: PTAM estimate, IMU, controls
- Output: pose estimate
- State vector: $(x, y, z, \dot{x}, \dot{y}, \dot{z}, \Phi, \Theta, \Psi, \dot{\Psi})^T$
- Full, calibrated model of the flight dynamics
- Delay compensation (~200ms)

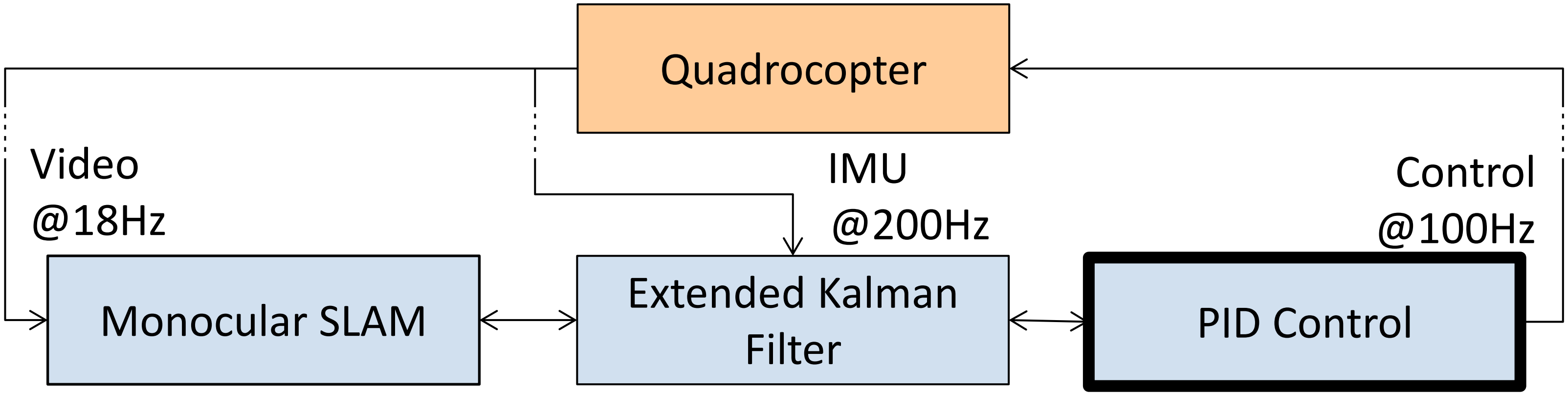
Camera-based Navigation

[Engel, Sturm, Cremers, IROS 2012]



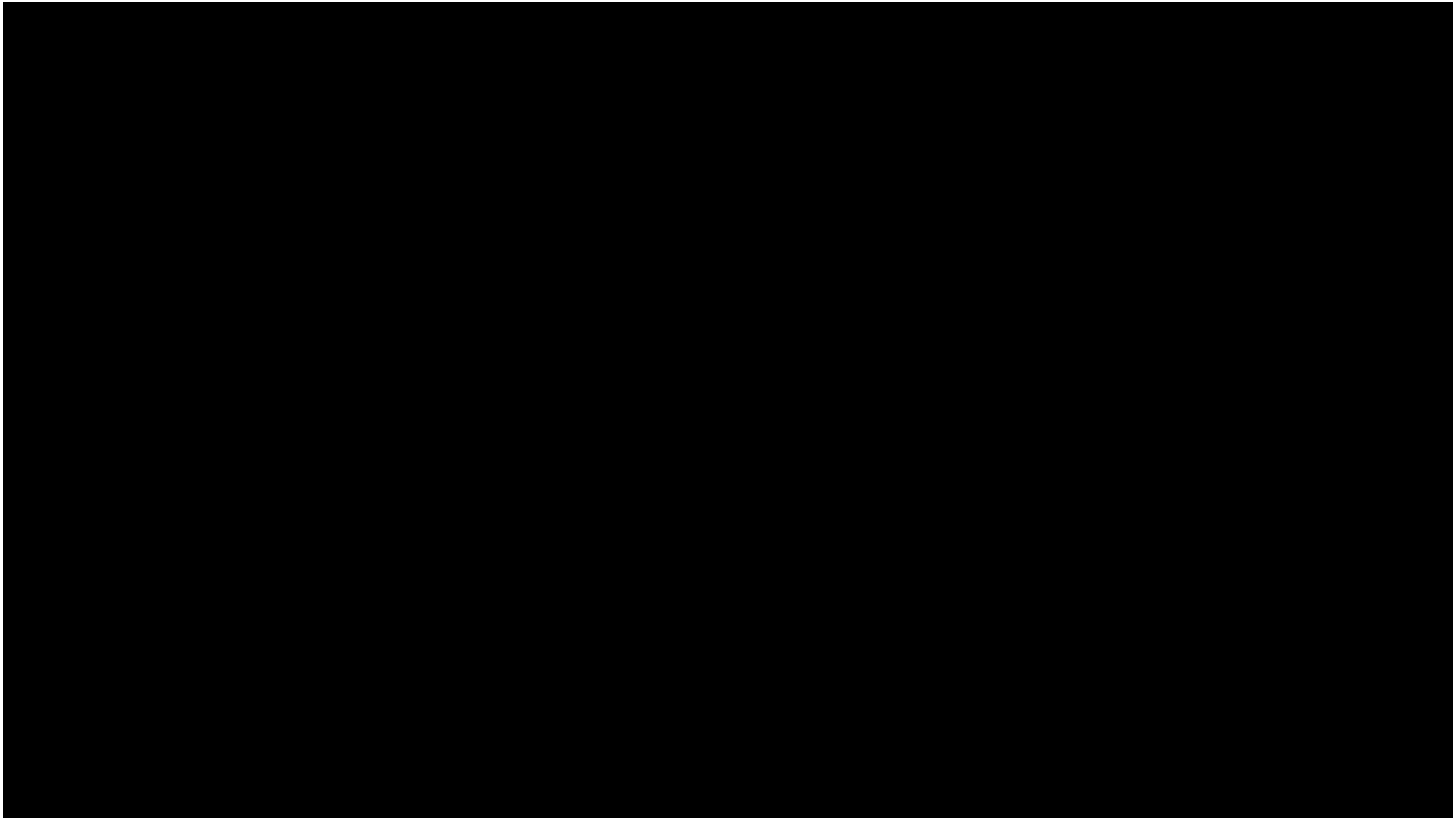
Camera-based Navigation

[Engel, Sturm, Cremers, IROS 2012]

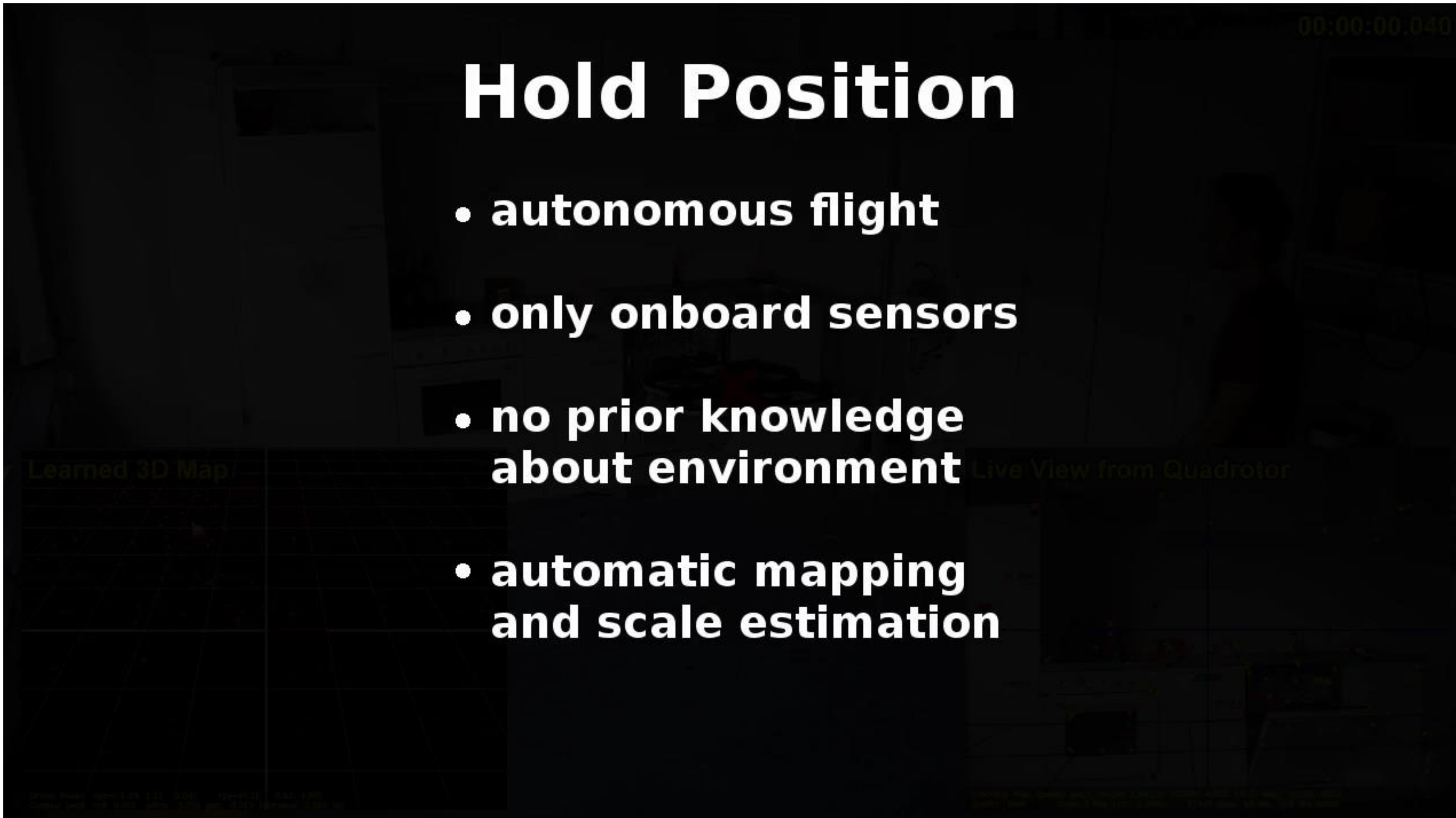


- Based on predicted state from EKF
- Approach and hold target position $(x, y, z, \Psi)^T$
- High level control:
  - Keep position
  - Assisted control (joystick in metric space)
  - Follow waypoints

Results



Results (cont.d)



Wrap-Up: Camera-Based Navigation

[Engel, Sturm, Cremers, IROS 2012]

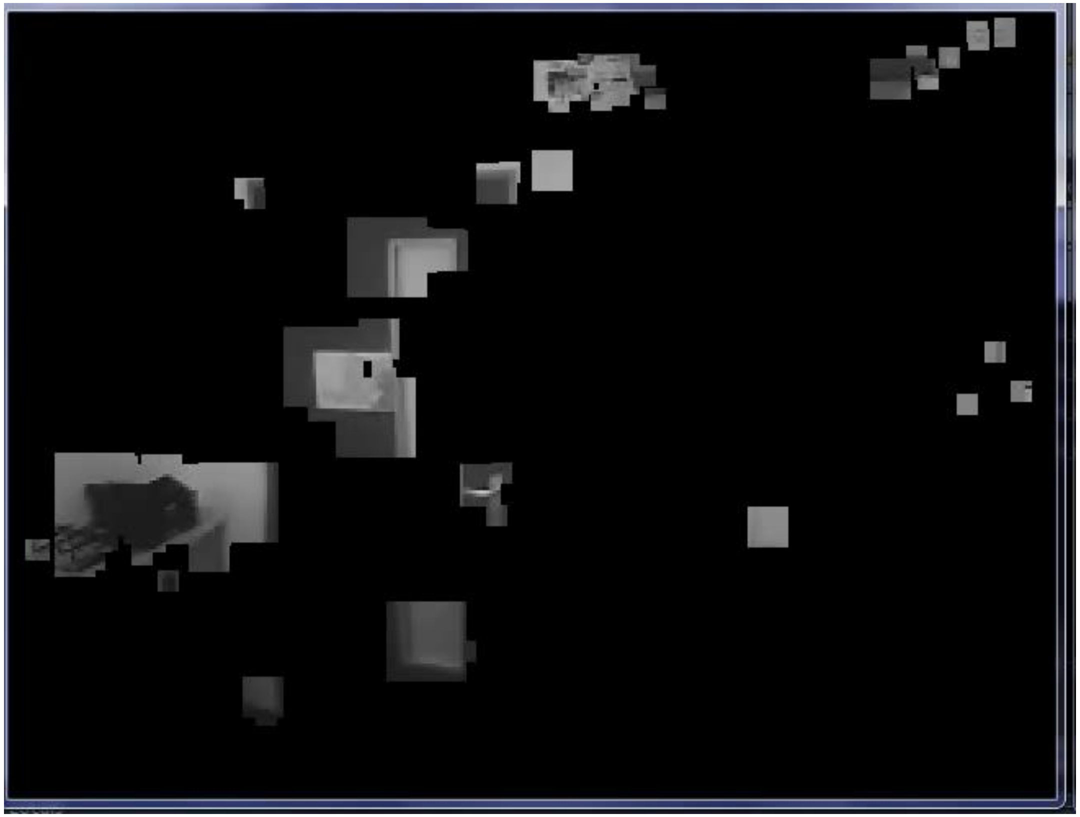
- Capabilities
  - Fast & accurate navigation (with up to 2 m/s)
  - Robust to temporary loss of visual tracking
  - No drift
  - Accurate scale estimation (2% RMSE)
  - Complete & working system (for only $300)
  - Open source
- Limitations
  - No obstacle recognition / path-planning
  - Requires sufficient keypoints in field of view

Feature-Based Visual SLAM

- Video feed from quadrocopter

Feature-Based Visual SLAM

- What PTAM actually sees



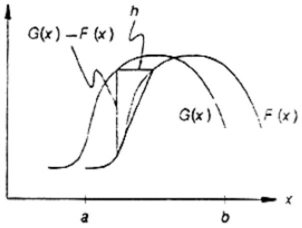
Dense Visual Odometry

- **Problem:** Keypoint-based approaches only use a small fraction of the available data
  - Keypoint detection
  - Visual features

- **Question:** How can we use most/all information to maximize the performance?

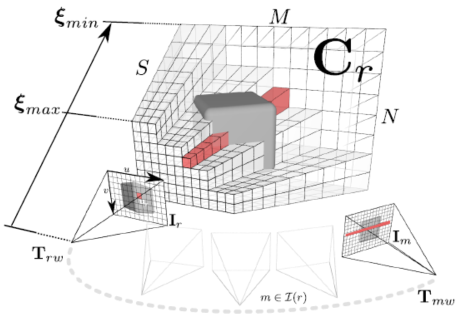
Related Work on Dense Tracking

- Lucas and Kanade (IJCAI'81)



- Lovegrove et al. (IV'11)

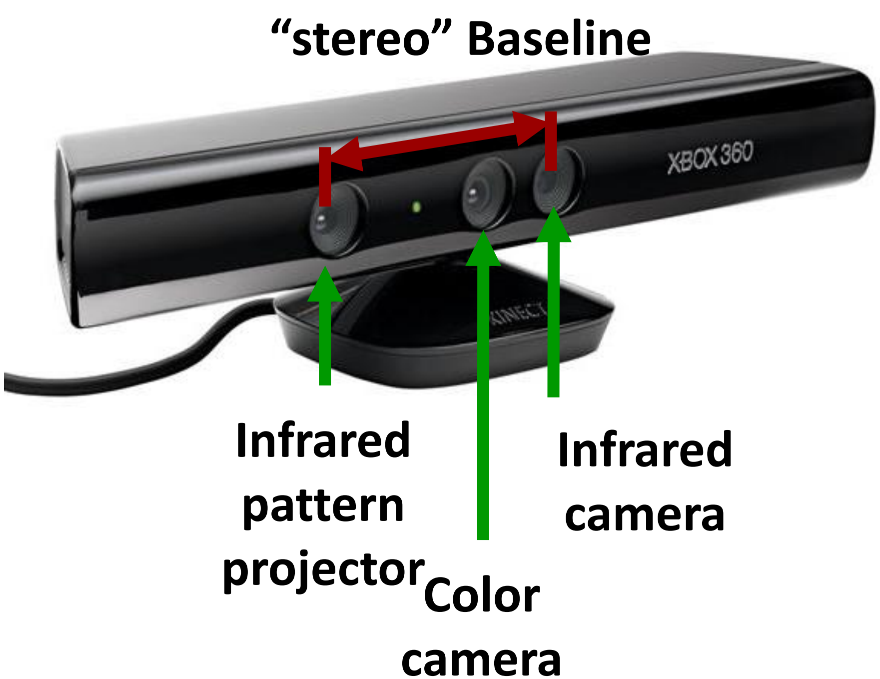
- Newcombe et al. (ICCV'11)



- Comport et al. (ICCV'11)

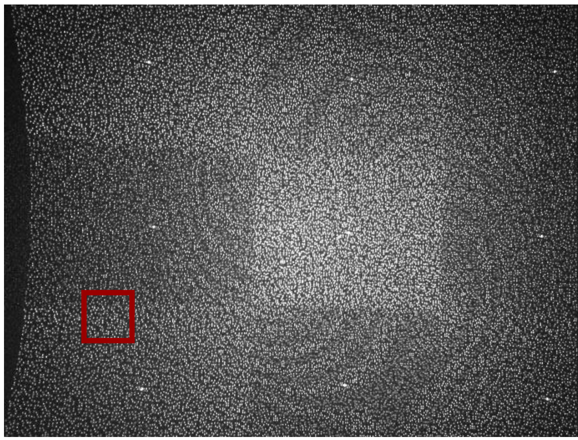
RGB-D Cameras

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



Sensor Principle of Kinect

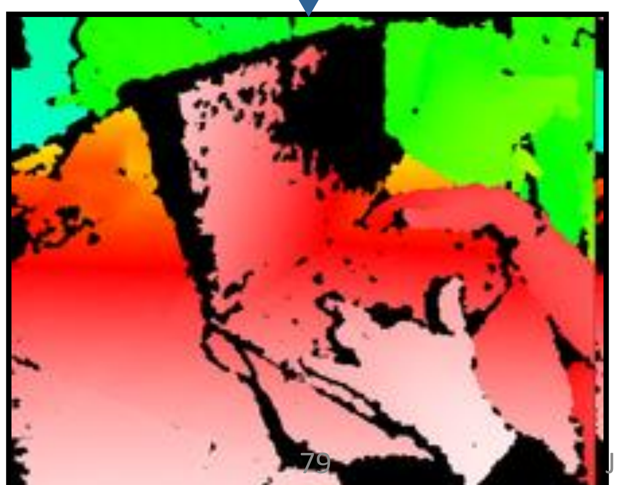
Infrared pattern
(known)



Infrared image
(with distorted pattern)

Standard block matcher (9x9)

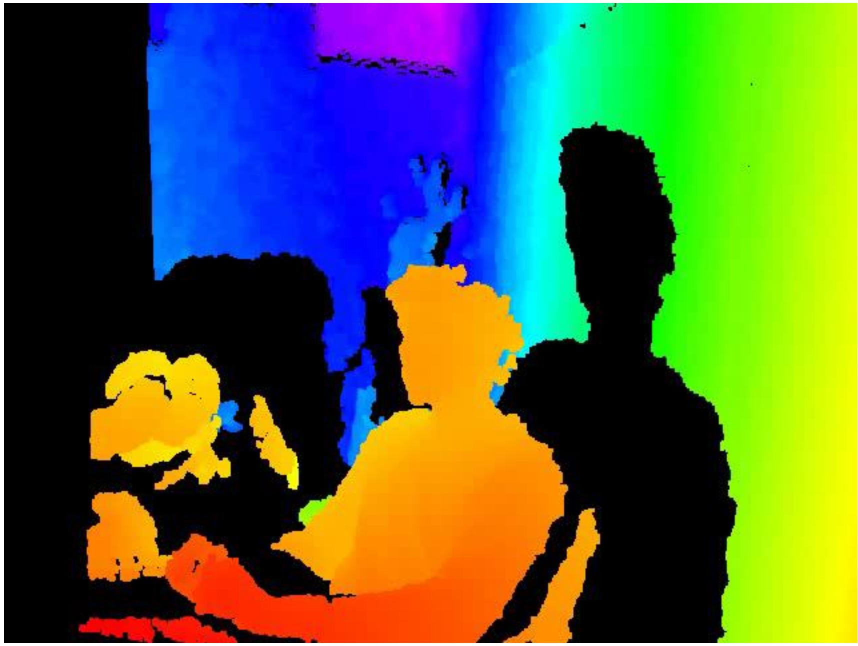
Disparity image

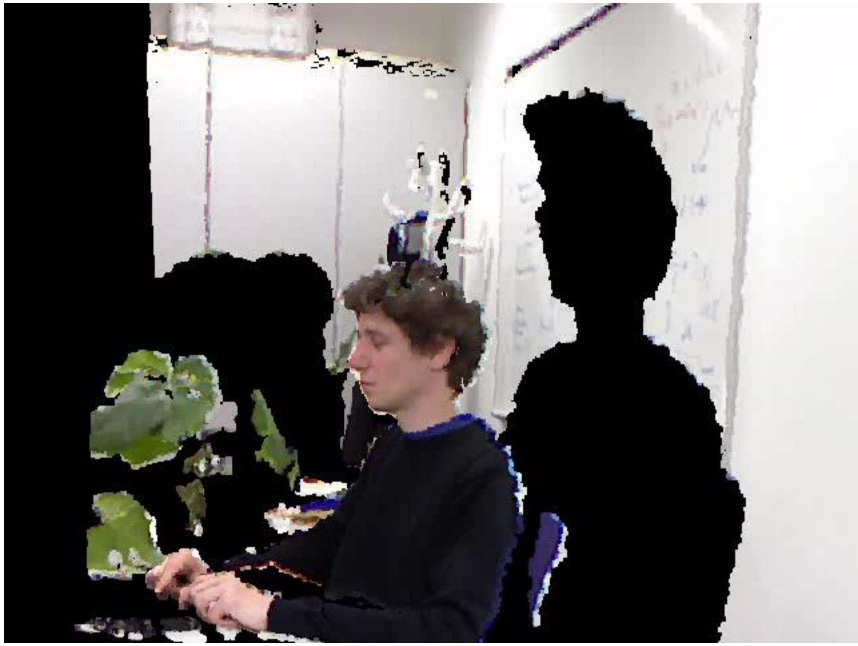


Depth image
(color encodes distance from camera)

Example Data

- Kinect provides color (RGB) and depth (D) video
- Dense depth video allows for completely novel approaches (will show two examples)

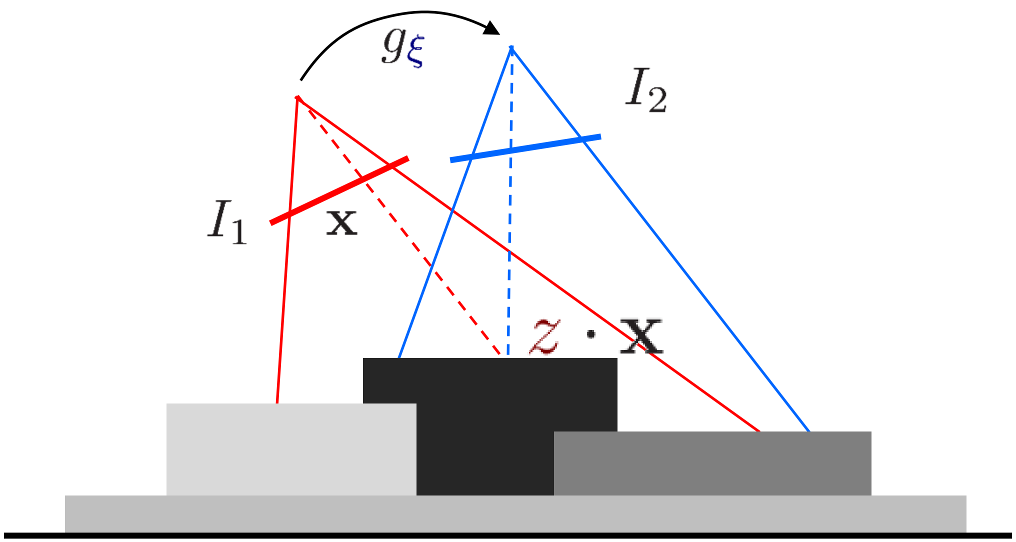




Dense Visual Odometry

[Steinbrücker et al., ICCV 2011; Kerl et al., ICRA 2013]

- How can we exploit all data of an RGB-D image?
- Idea



- Photo-consistency constraint

$$I_1(\mathbf{x}) = I_2\left(\pi(g_\xi(z \cdot \mathbf{x})\right) \text{ for all pixels } \mathbf{x}$$

How to deal with noise?

[Steinbrücker et al., ICCV 2011; Kerl et al., ICRA 2013]

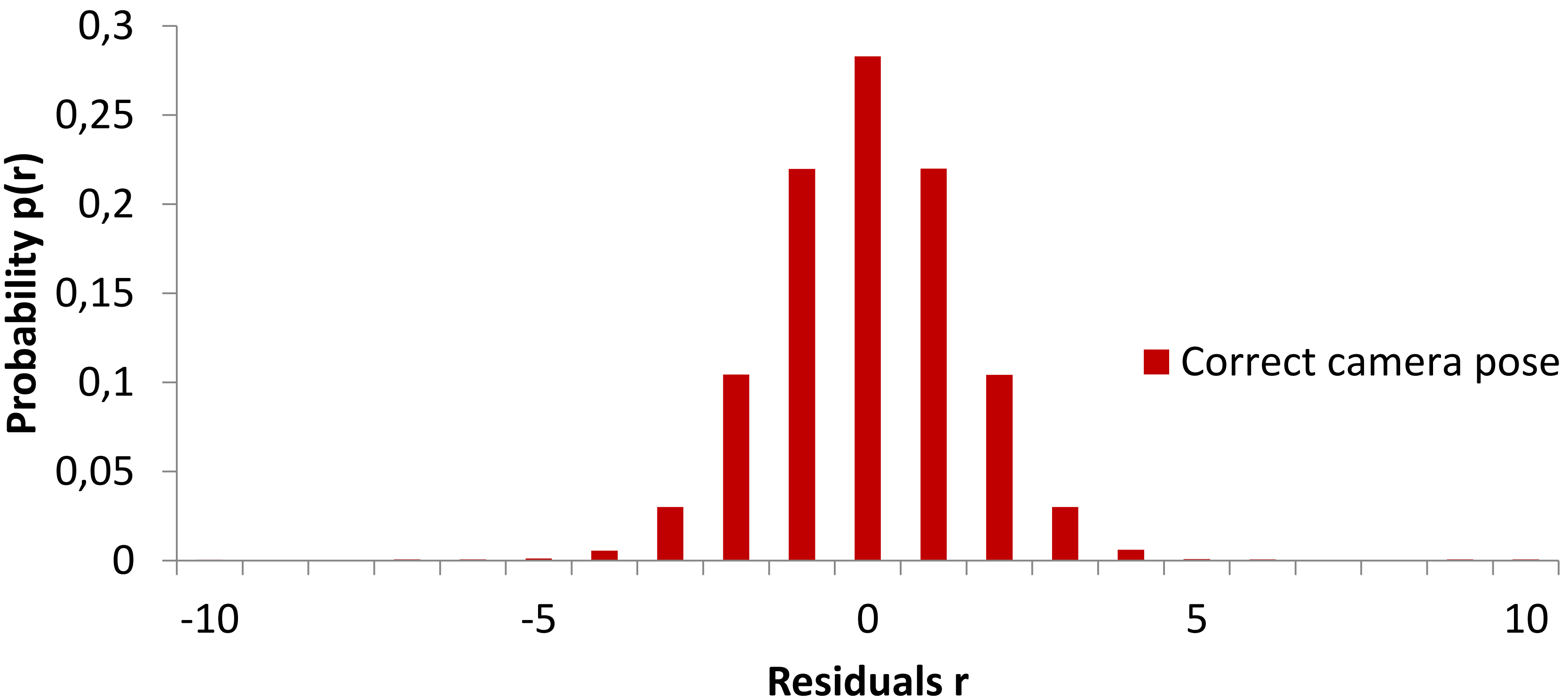
- Photo-consistency constraint will not perfectly hold
  - Sensor noise
  - Pose error
  - Reflections, specular surfaces
  - Dynamic objects (e.g., walking people)
- Residuals will be non-zero

$$r = I_1(\mathbf{x}) - I_2\left(\pi(g_\xi(z \cdot \mathbf{x}))\right)$$

- Residual distribution $p(r)$

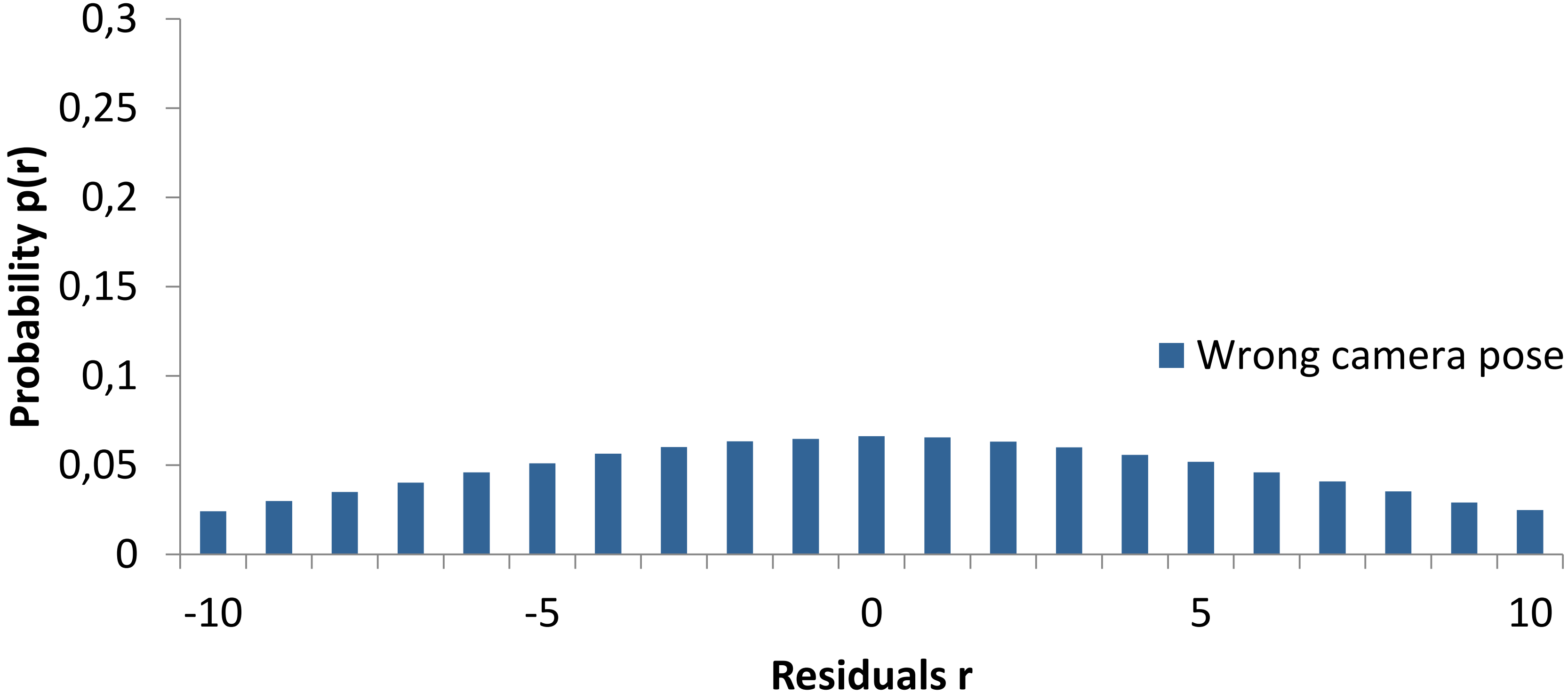
Residual Distribution
[Steinbrücker et al., ICCV 2011; Kerl et al., ICRA 2013]

- Zero-mean, peaked distribution
- Example: Correct camera pose



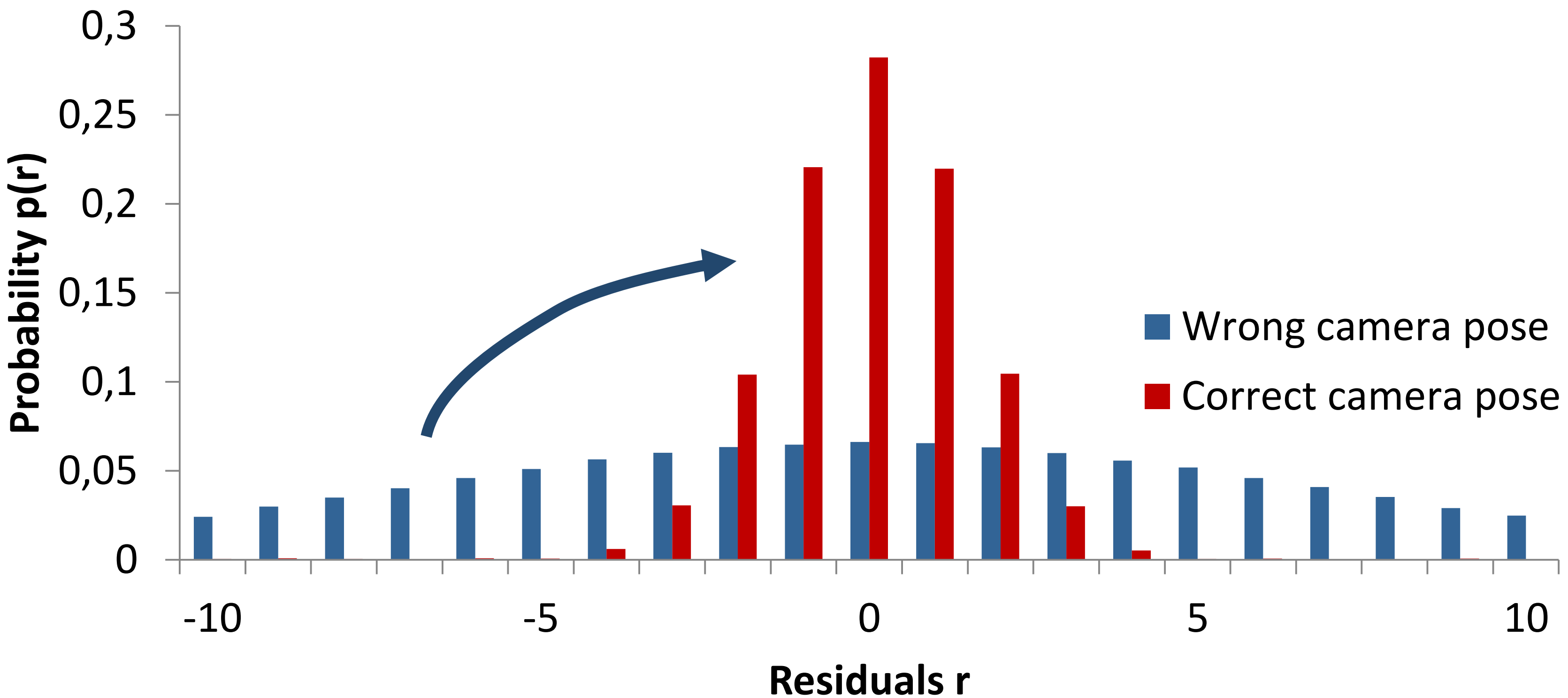
Residual Distribution
[Steinbrücker et al., ICCV 2011; Kerl et al., ICRA 2013]

- Zero-mean, peaked distribution
- Example: Wrong camera pose



Residual Distribution
[Steinbrücker et al., ICCV 2011; Kerl et al., ICRA 2013]

- **Goal:** Find the camera pose that maximizes the observation likelihood



Motion Estimation

[Steinbrücker et al., ICCV 2011; Kerl et al., ICRA 2013]

- **Goal:** Find the camera pose that maximizes the observation likelihood

$$\xi^* = \arg\max_{\xi} \prod_i p(r_i(\xi))$$

compute over all pixels

- Assume pixel-wise residuals are conditionally independent
- How can we solve this optimization problem?

Approach

[Steinbrücker et al., ICCV 2011; Kerl et al., ICRA 2013]

- Take negative logarithm

$$\xi^* = \arg\min_{\xi} \sum_i -\log p(r_i(\xi))$$

- Set derivative to zero

$$\sum_i \frac{\partial \log p(r_i(\xi))}{\partial \xi} = \sum_i \frac{\partial \log p(r_i)}{\partial r_i} \frac{\partial r_i(\xi)}{\partial \xi} \stackrel{!}{=} 0$$

Approach (cont.d)
[Steinbrücker et al., ICCV 2011; Kerl et al., ICRA 2013]

- This can be rewritten as a weighted least squares problem

$$\xi^* = \arg\min_{\xi} \sum_i w(r_i)(r_i(\xi))^2$$

with weights $w(r_i) = \frac{\partial \log p(r_i)}{\partial r_i} \frac{1}{r_i}$

- $r_i(\xi)$ is non-linear in ξ
- Need to linearize, solve, and iterate

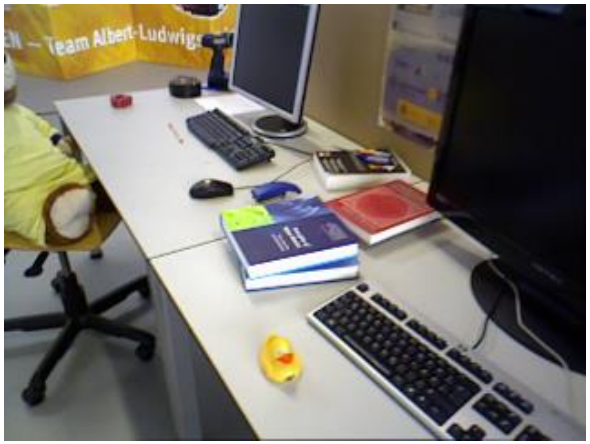
Iteratively Reweighted Least Squares

Problem: $$\xi^* = \arg\min_{\xi} \sum_i w(r_i)(r_i(\xi))^2$$

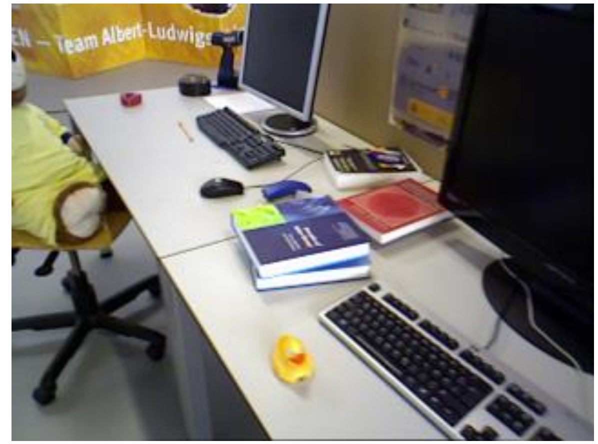
Algorithm:

1. Compute weights $w(r_i) = \frac{\partial \log p(r_i)}{\partial r_i} \frac{1}{r_i}$
2. Linearize in the camera motion ξ
$$r_{\text{lin}}(\xi_0 + \Delta\xi) = r(\xi_0) + J\Delta\xi$$
3. Build and solve normal equations
$$J^T W J \Delta\xi = -J^T W r(\xi_0)$$
4. Repeat until convergence

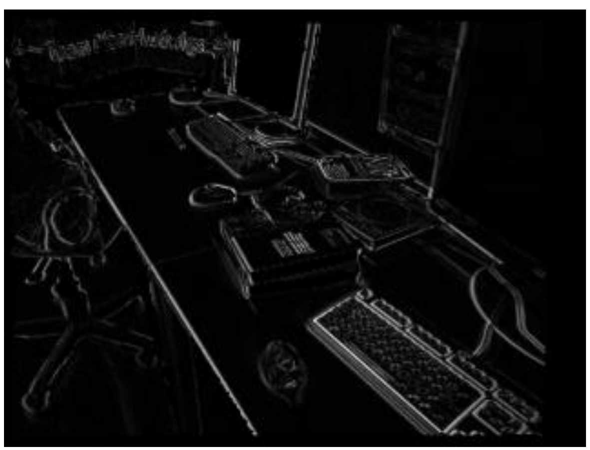
Example



First input image



Second input image



Residuals

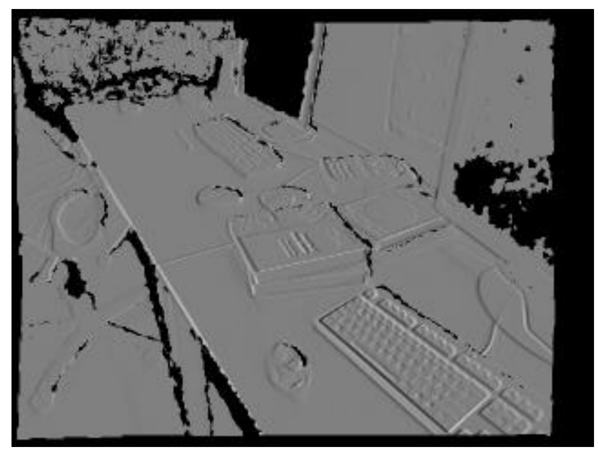
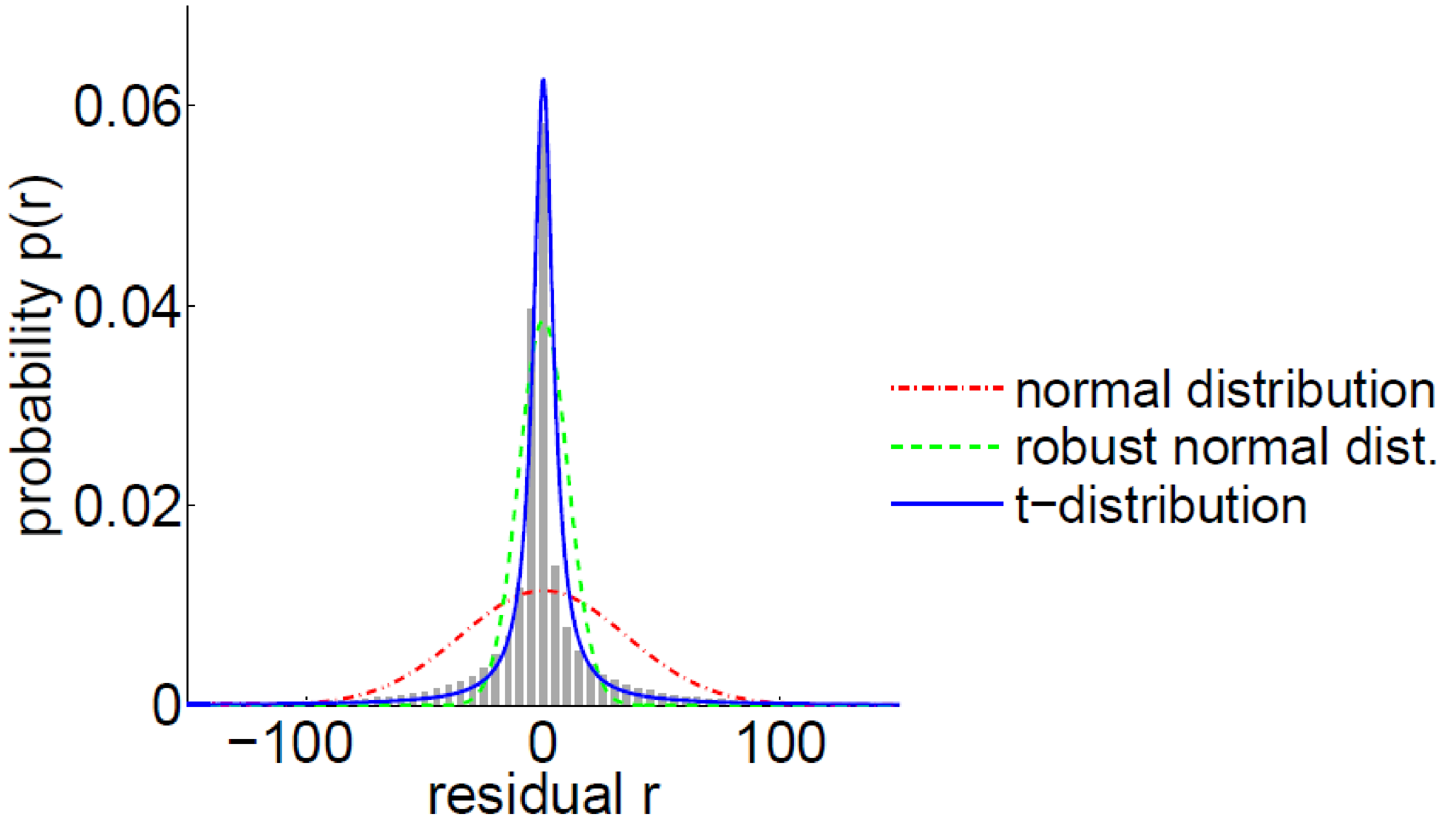
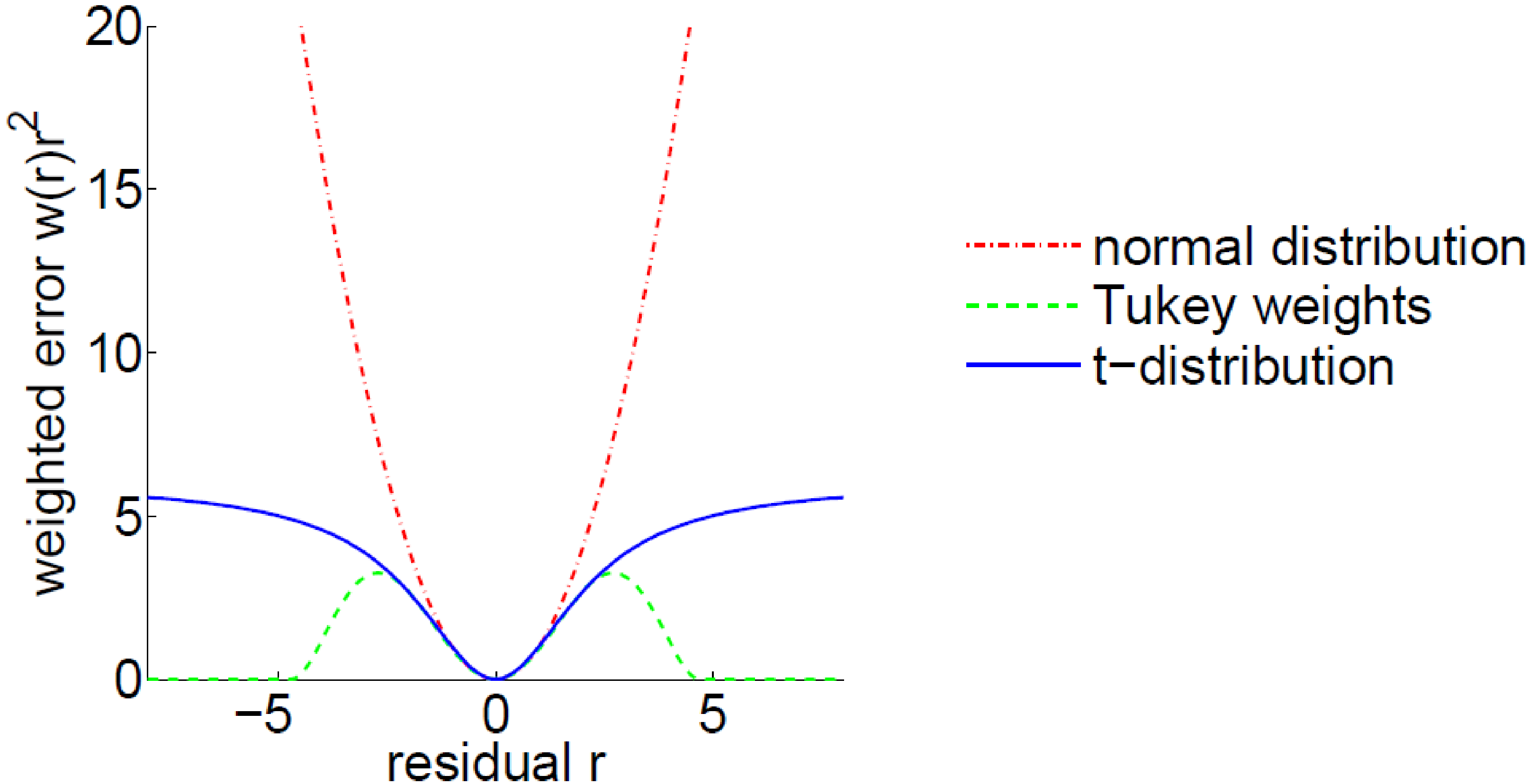


Image Jacobian for
camera motion along x axis

What is a Good Model for the Residual Distribution?



Weighted Error

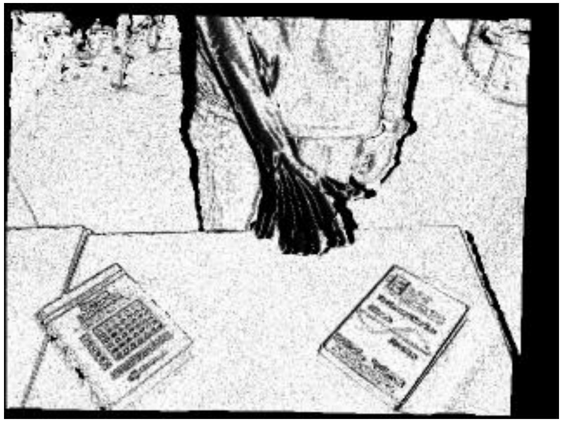


Example Weights

- Robust sensor model allows to down-weight outliers (dynamic objects, motion blur, reflections, …)

Scene

Residuals

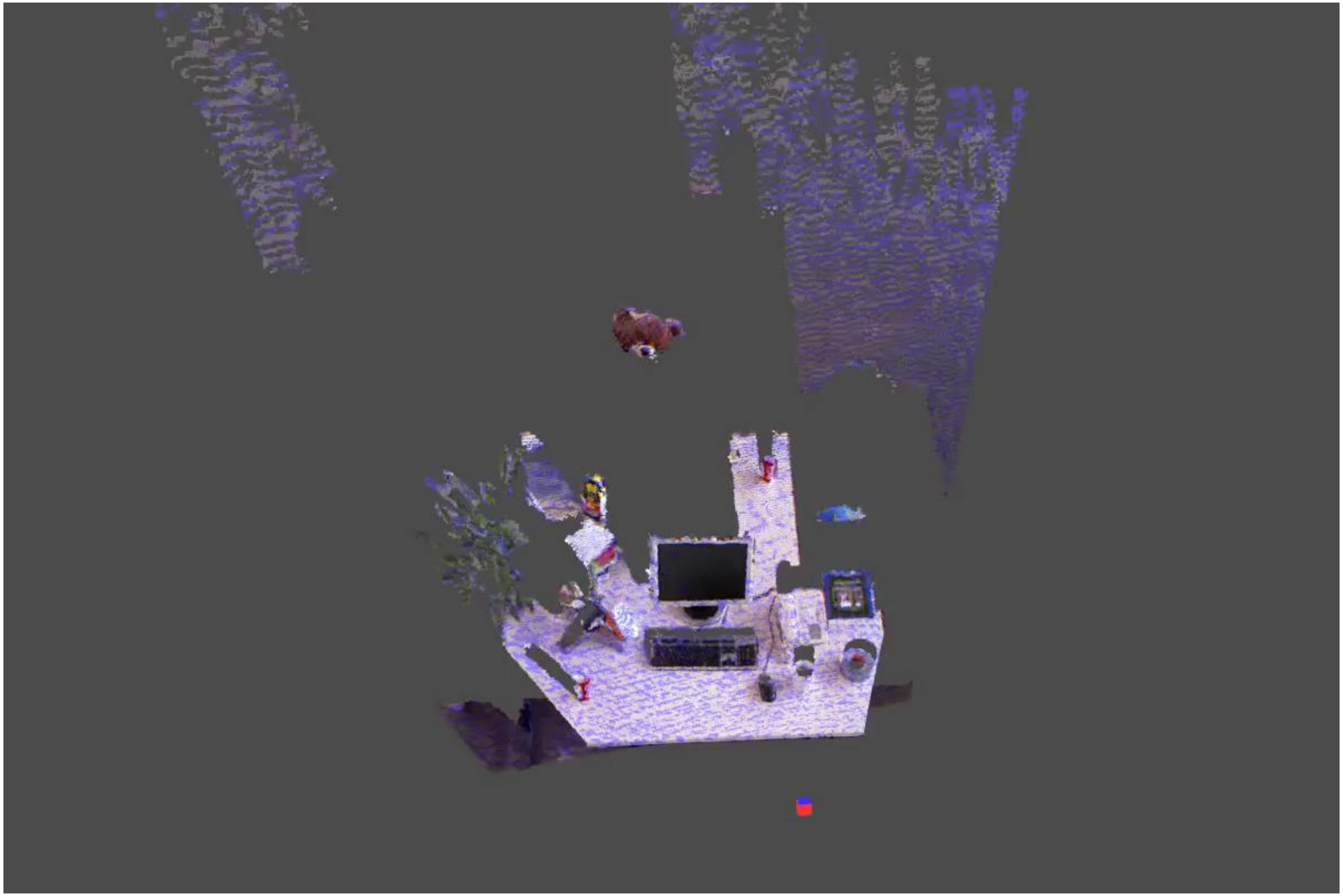


Weights

Coarse-to-Fine

- Linearization only holds for small motions
- Coarse-to-fine scheme
- Image pyramids

Dense Visual Odometry: Results



Dense Visual Odometry: Results

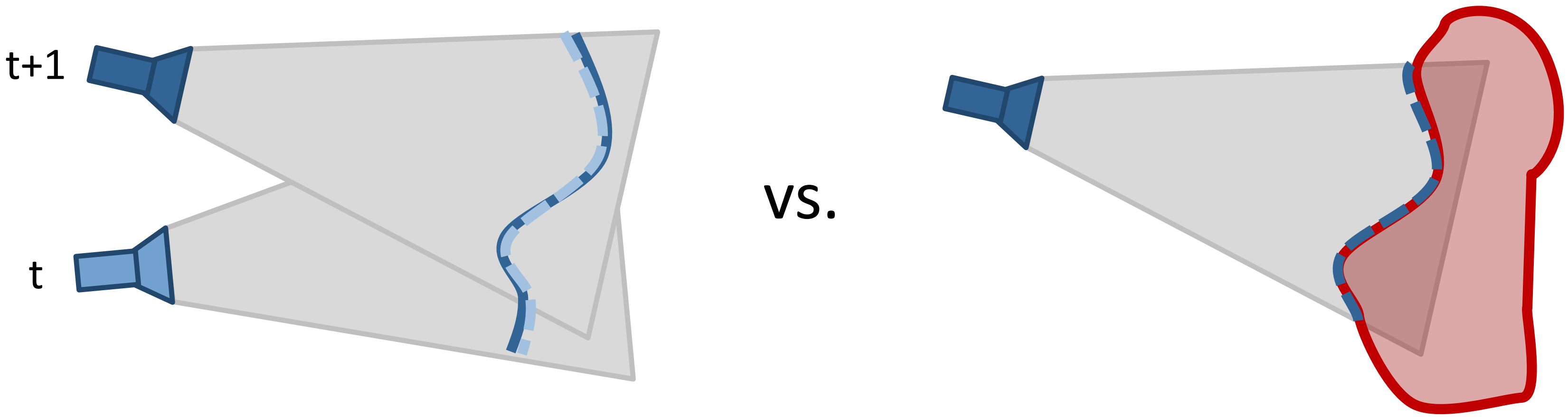
Wrap-Up: Dense Visual Odometry

[Steinbrücker et al., ICCV 2011; Kerl et al., ICRA 2013]

- Direct matching of consecutive RGB-D images
- Pro
  - Super fast, highly accurate (30 Hz on CPU)
  - Robust to outliers
  - Low, constant memory consumption
- Con
  - Accumulates drift over time, no fixed reference
- Available as open-source

Dense Tracking and Mapping
[Bylow et al., RSS 2013]

- Idea: Instead of tracking from **frame**-to-**frame**, track **frame**-to-**model** to reduce the drift



- **Question:** Where do we get the model from?

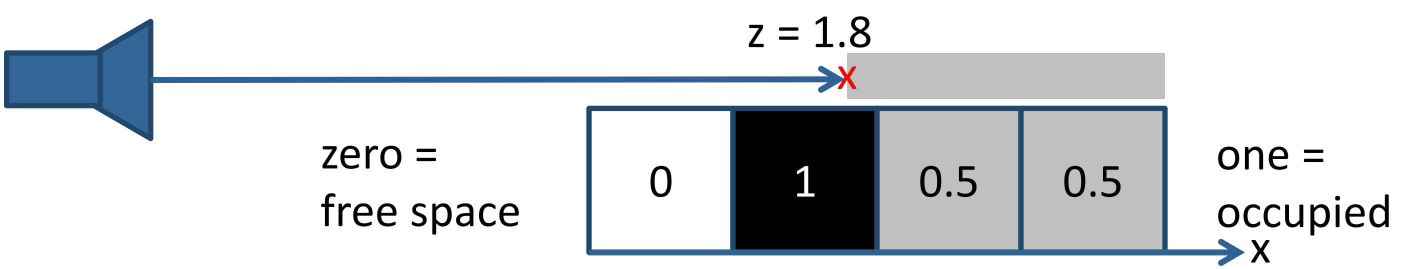
Dense Tracking and Mapping
[Bylow et al., RSS 2013]

- **Idea:** Compute an iterative solution
  1. Reconstruct model with known poses
  2. Track camera with respect to known model

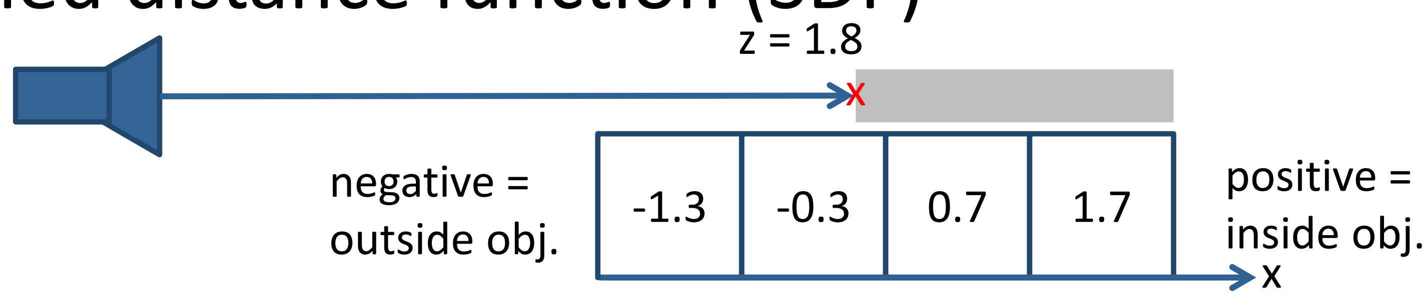
- **Next question:** How to represent the model?

Representation of the 3D Model

- **Idea:** Instead of representing the cell occupancy, represent the distance of each cell to the surface
- Occupancy grid maps



- Signed distance function (SDF)



Wrap-Up: Dense Mapping

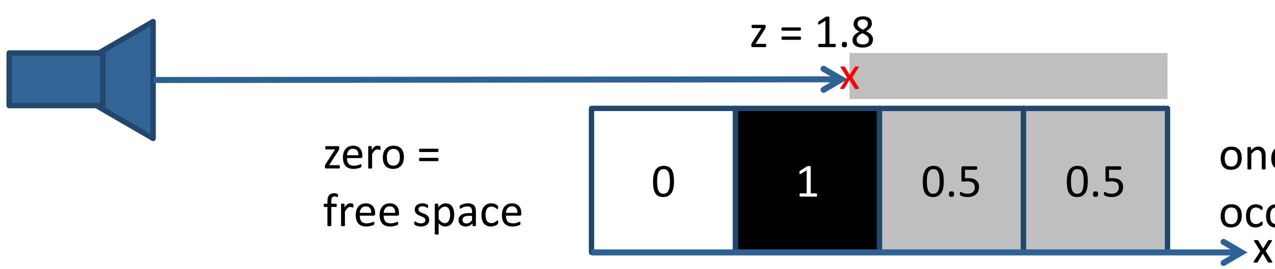
[Bylow et al., RSS 2013]

- Pro
  - Real-time
  - Accuracy similar to RGB-D SLAM on small indoor scenes
  - Nice models
- Con
  - Needs GPU
  - Still drifts (although less)
  - High memory consumption
- How to eliminate the drift?

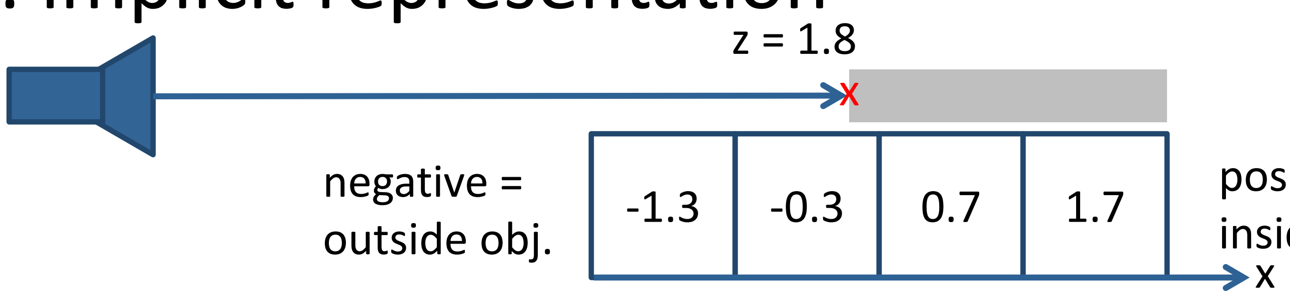
Signed Distance Field (SDF)

[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

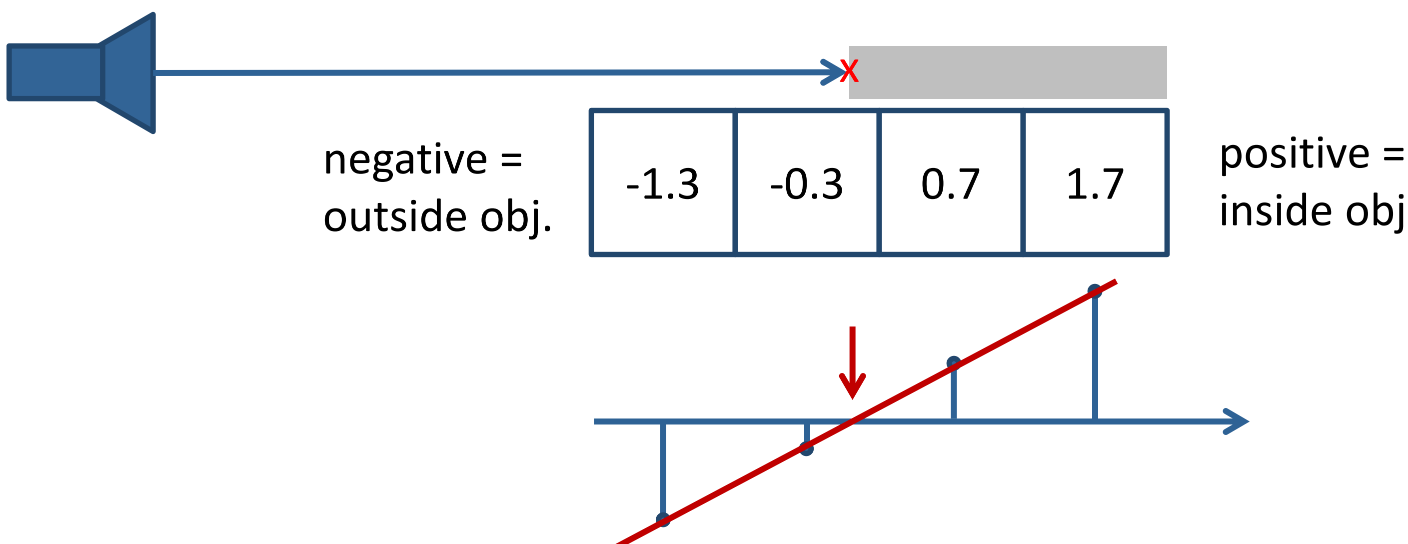


Signed Distance Field (SDF)

[Curless and Levoy, 1996]

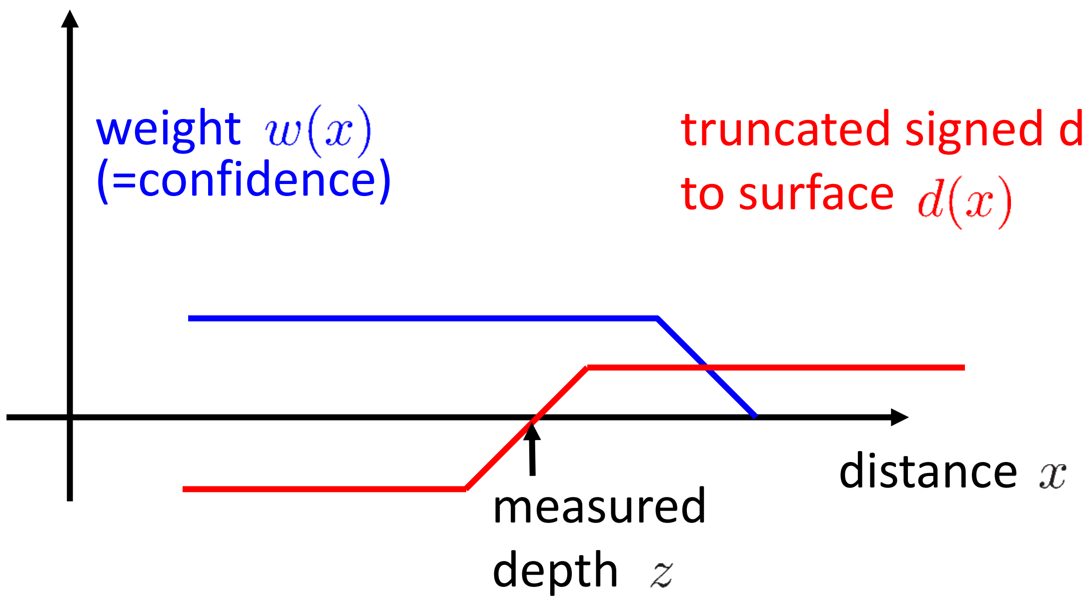
Algorithm:

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



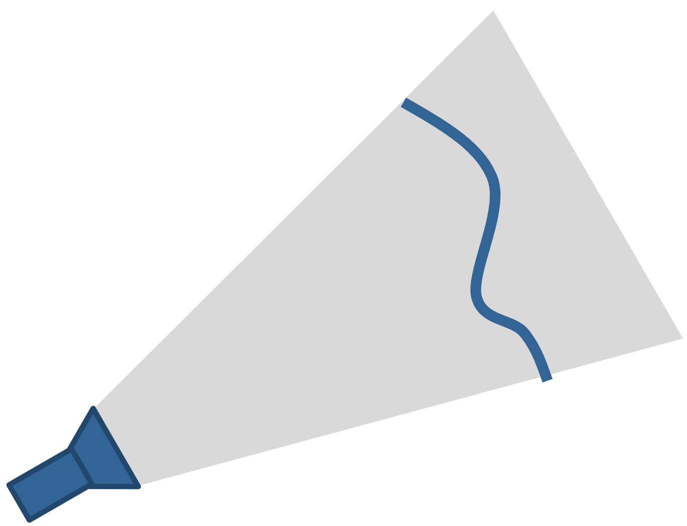
Distance and Weighting Functions

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



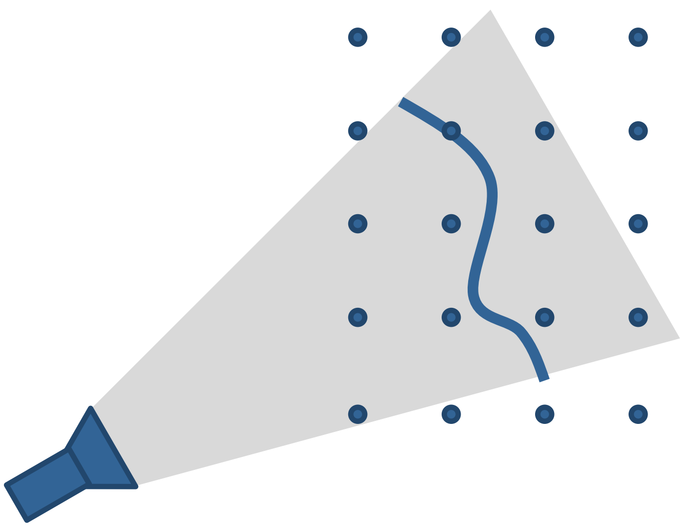
Dense Mapping: 2D Example

- Camera with known pose



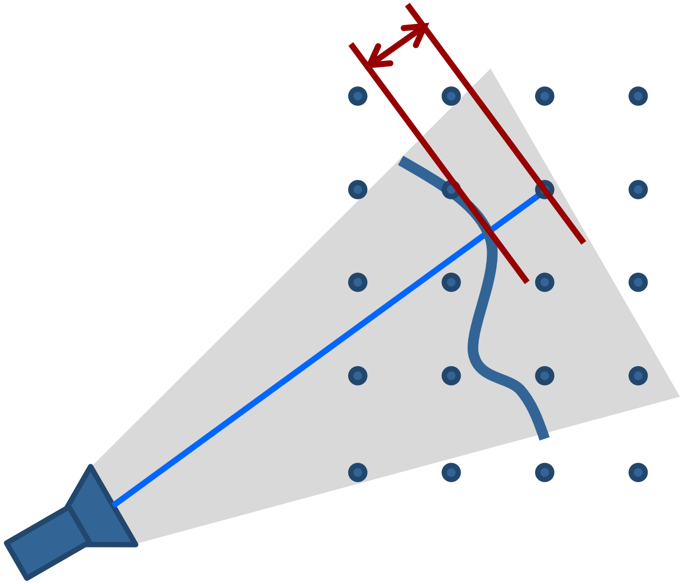
Dense Mapping: 2D Example

- Camera with known pose
- Grid with signed distance function



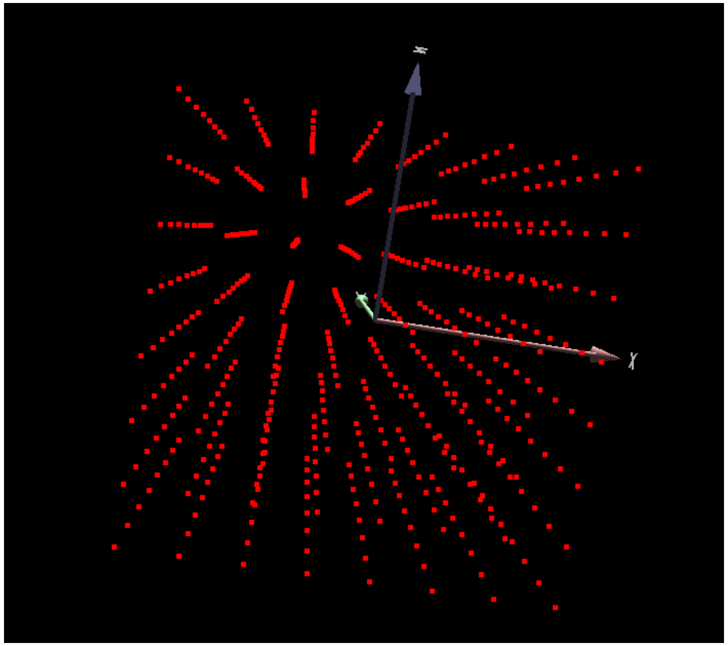
Dense Mapping: 2D Example

- 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



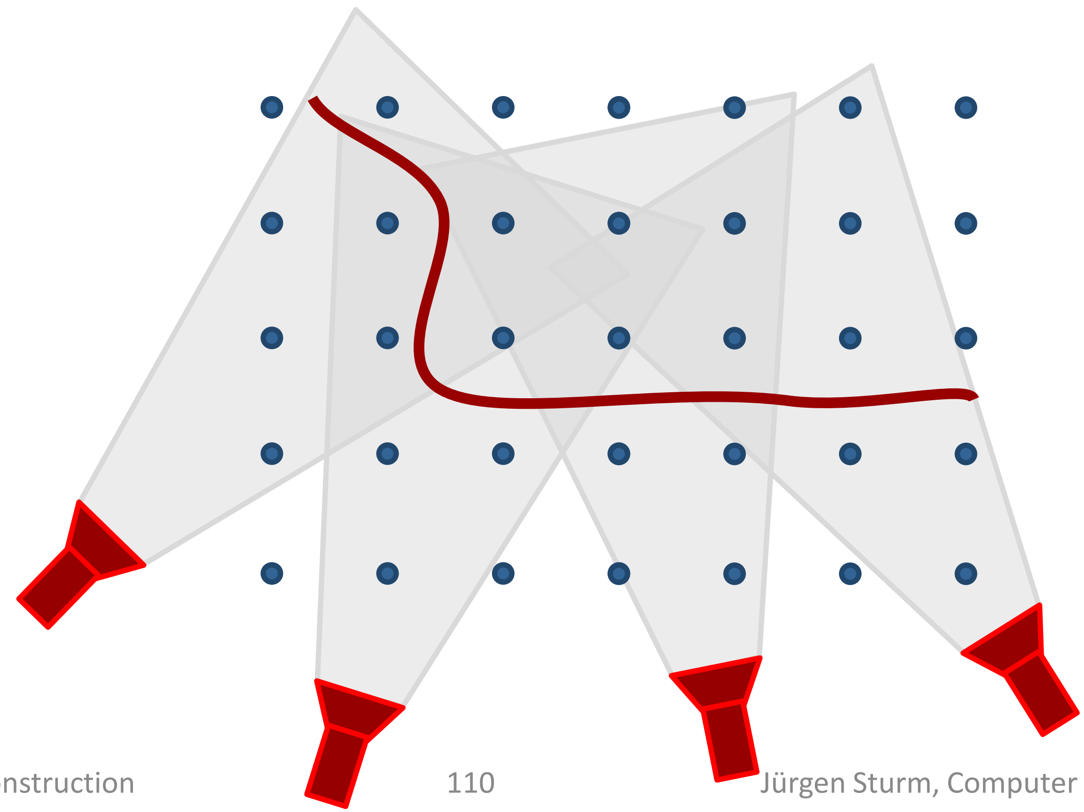
Data Fusion

- **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

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

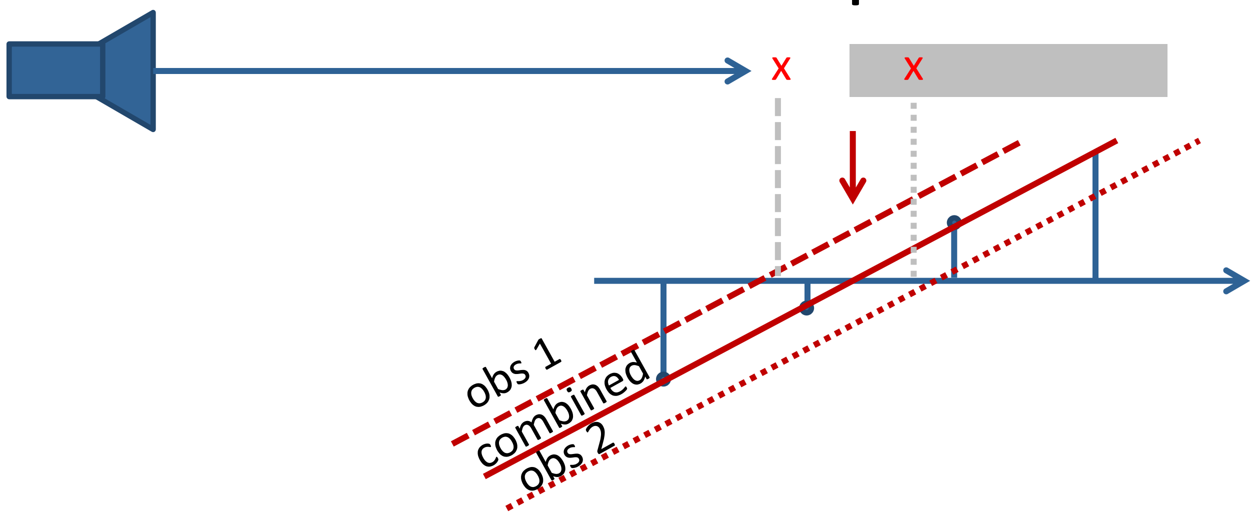
Data Fusion

- 3D model built from the first k frames



Two Nice Properties

- Noise cancels out over multiple measurements

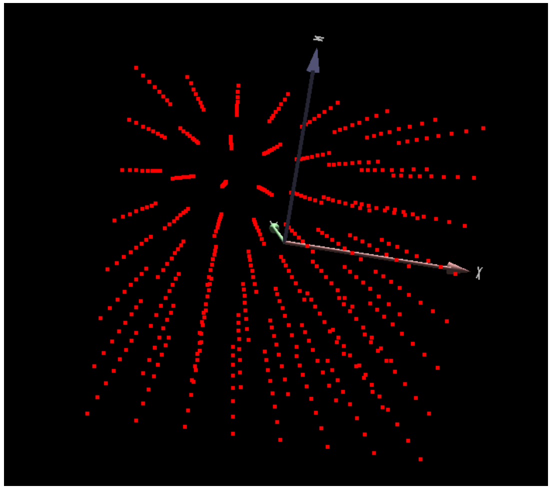


- Zero-crossing can be extracted at sub-voxel accuracy (least squares estimate)

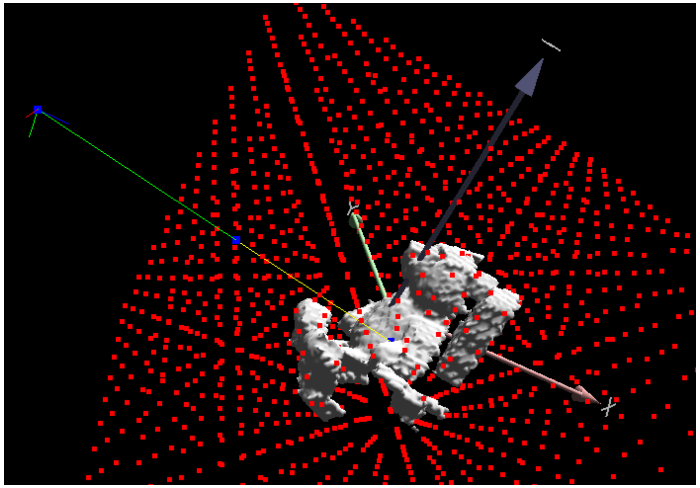
1D Example: $$x^* = \frac{\sum D_t(x)x}{\sum W_t(x)x}$$

Surface Reconstruction

- **We have:** 3D signed distance field
- **We want:** Triangle mesh for rendering
- How can we extract a 3D triangle mesh from the SDF?



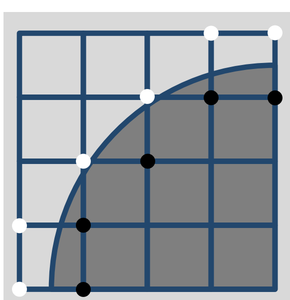
?

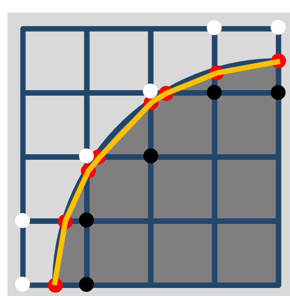


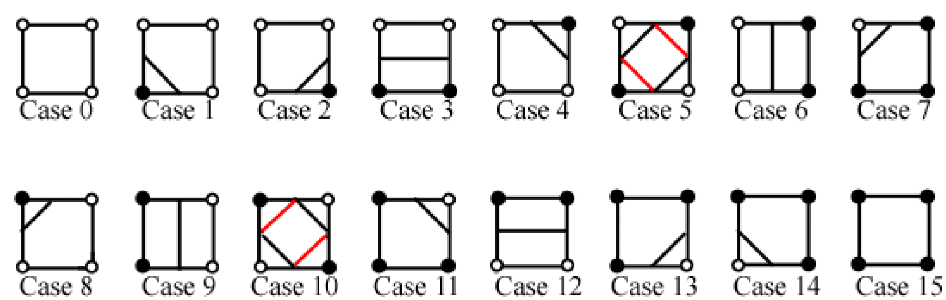
Marching Cubes

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

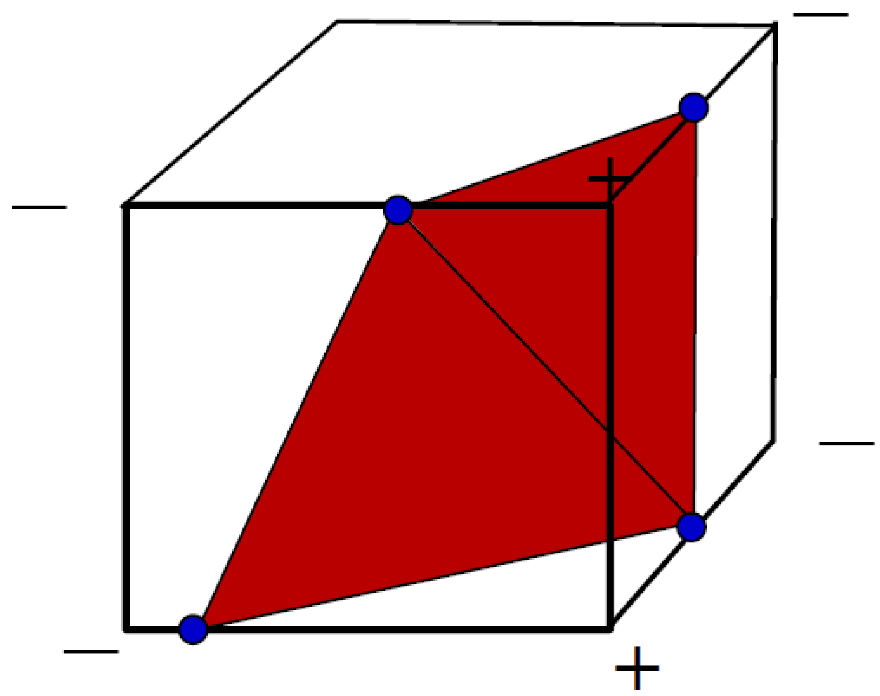




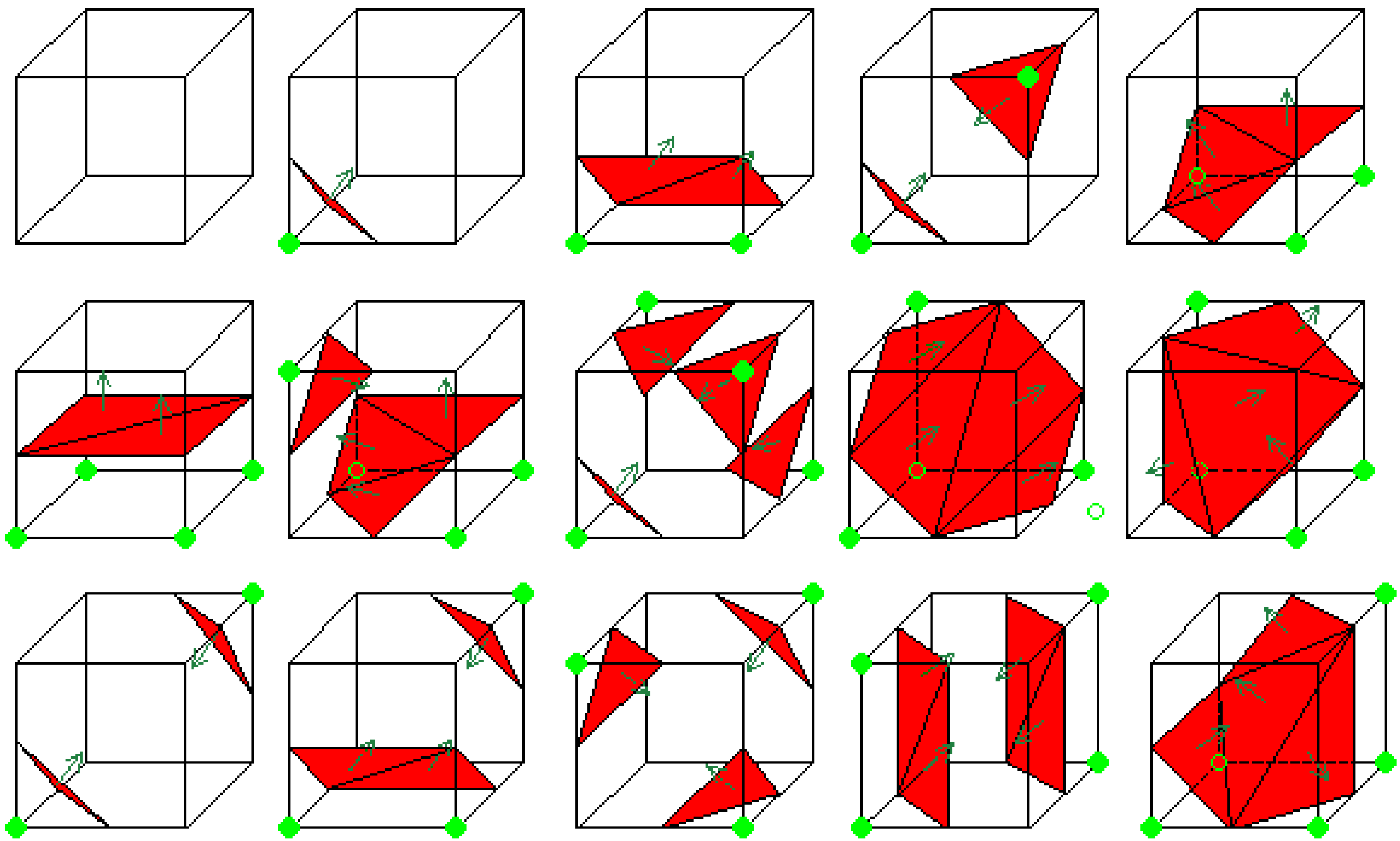


Marching Cubes

- In 3D, the principle is the same
- Generate triangles instead of lines



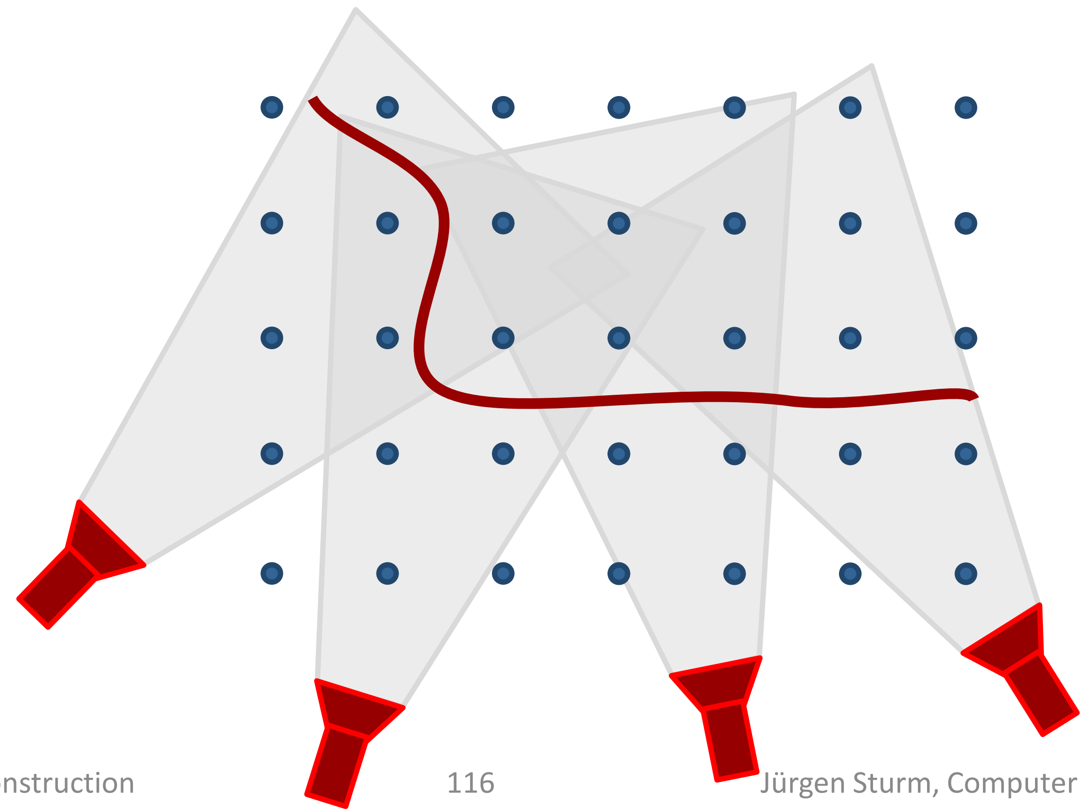
Marching Cubes



Dense Tracking: 2D Example

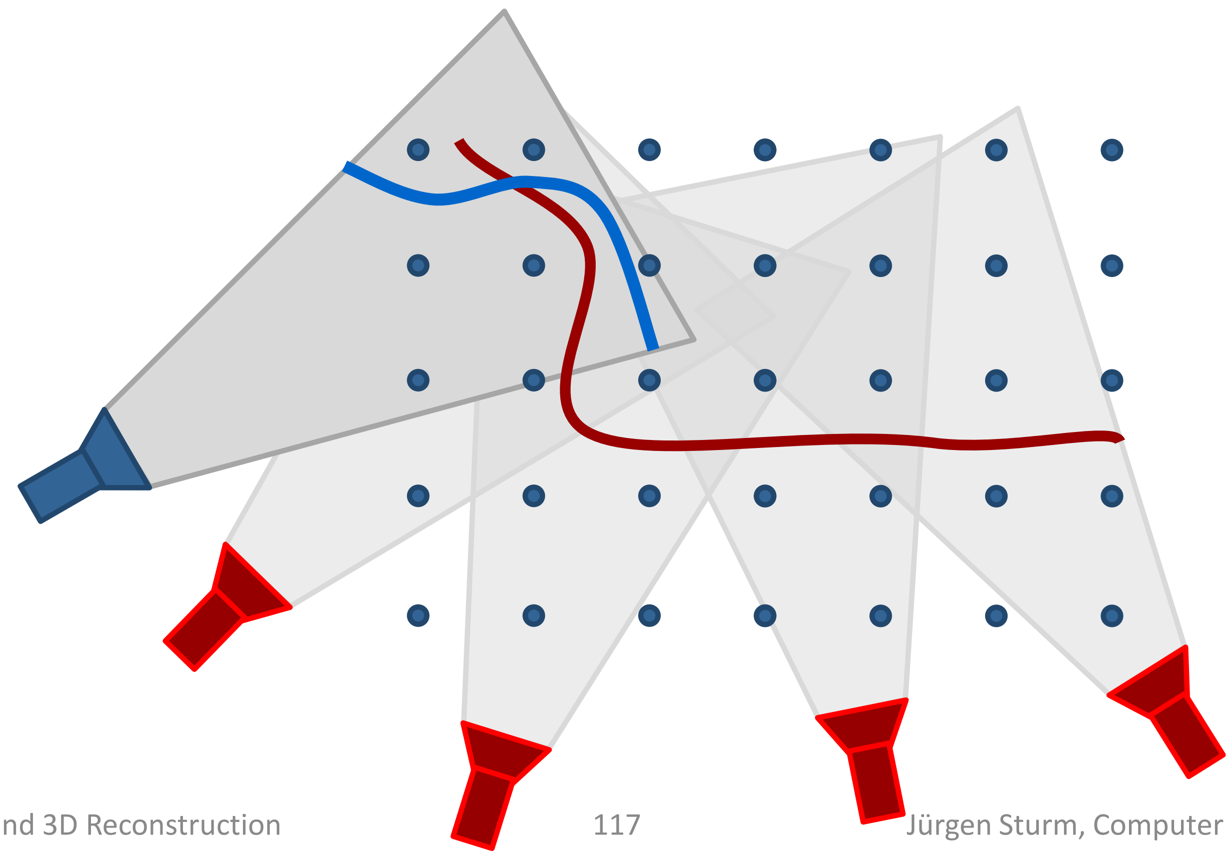
[Bylow et al., RSS 2013]

- 3D model built from the first k frames



Dense Tracking: 2D Example
[Bylow et al., RSS 2013]

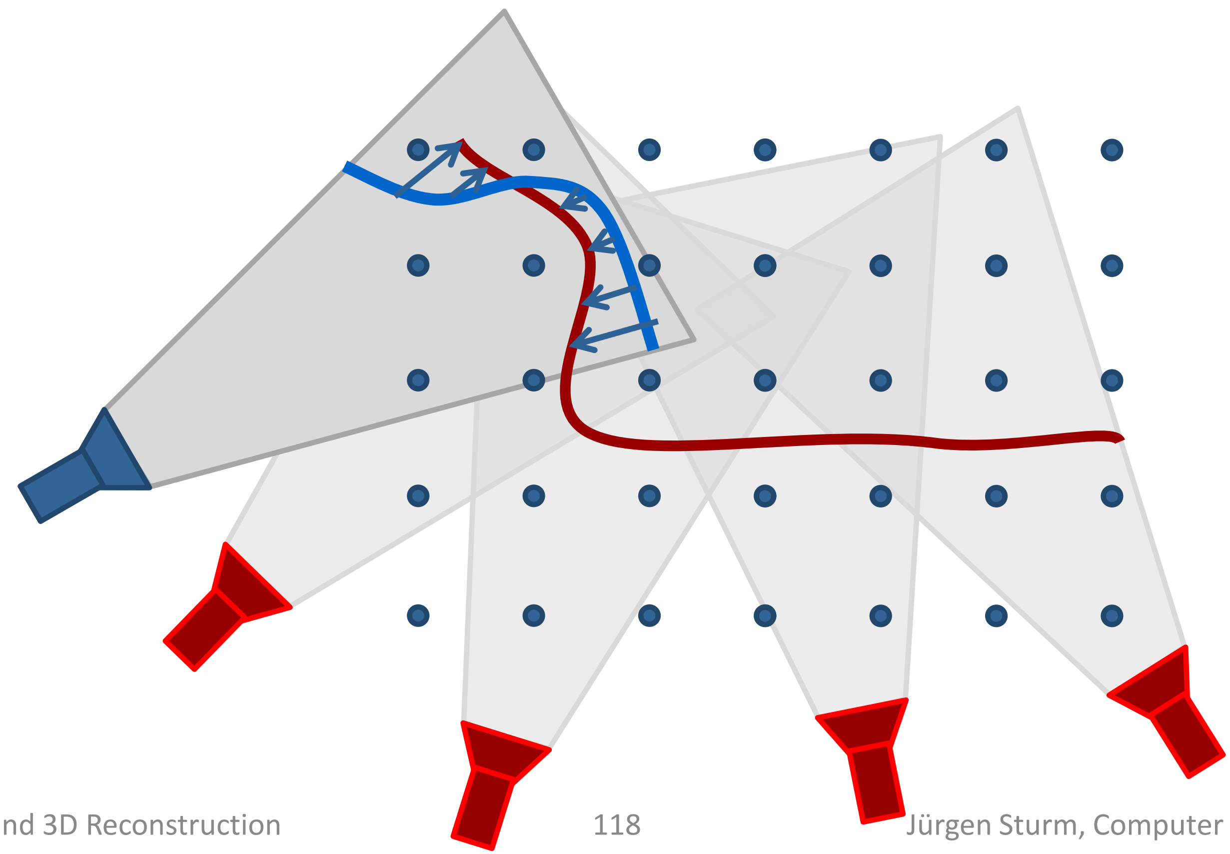
- Minimize distance between depth image and SDF



Dense Tracking: 2D Example

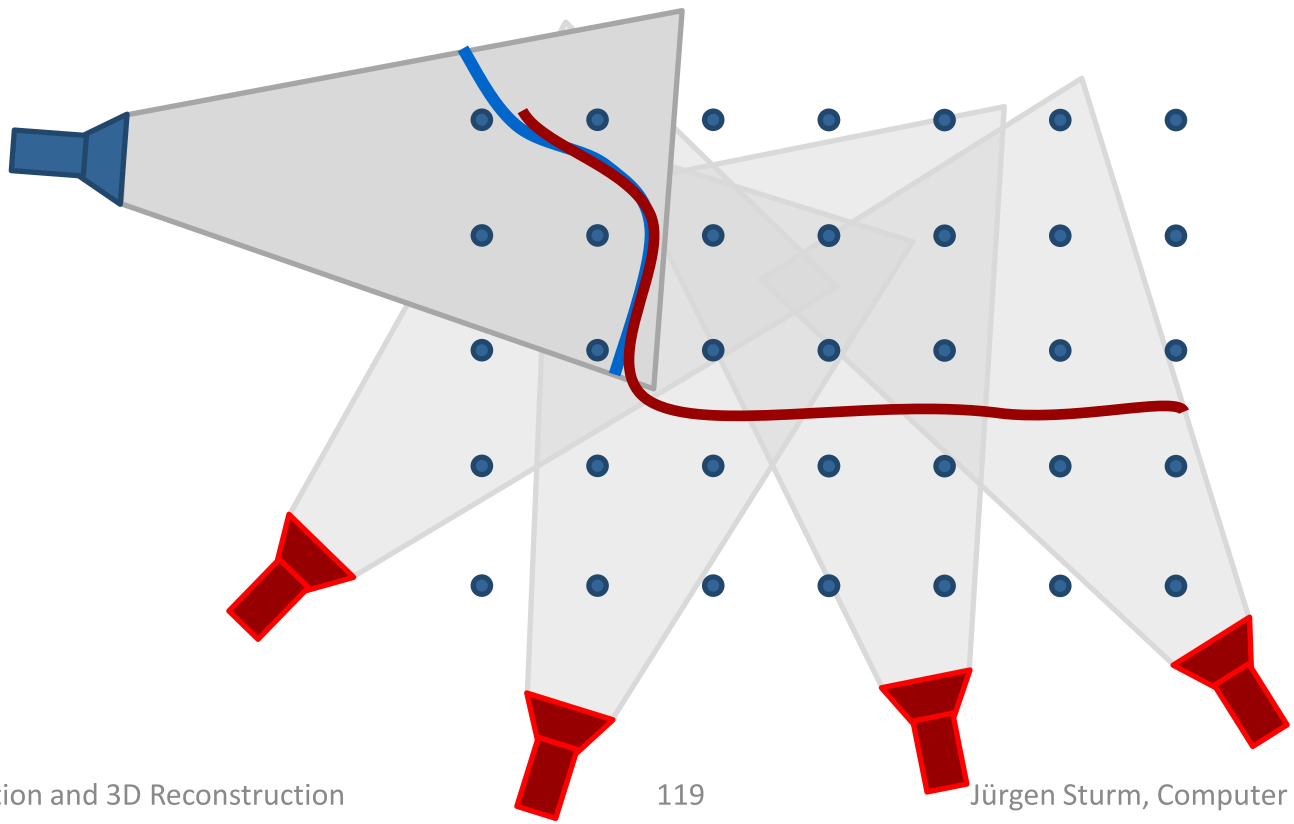
[Bylow et al., RSS 2013]

- Minimize distance between depth image and SDF



Dense Tracking: 2D Example
[Bylow et al., RSS 2013]

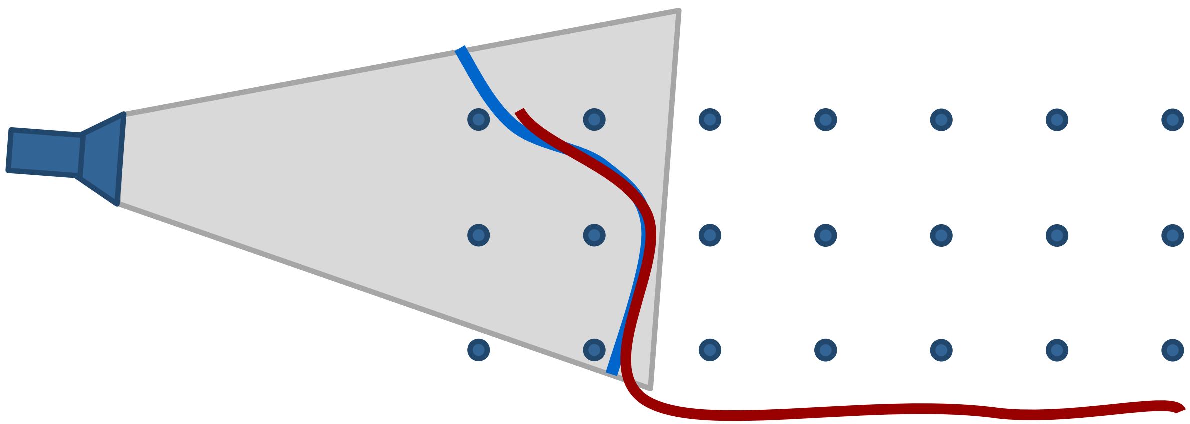
- Minimize distance between depth image and SDF



Dense Tracking: 2D Example

[Bylow et al., RSS 2013]

- Minimize distance between depth image and SDF



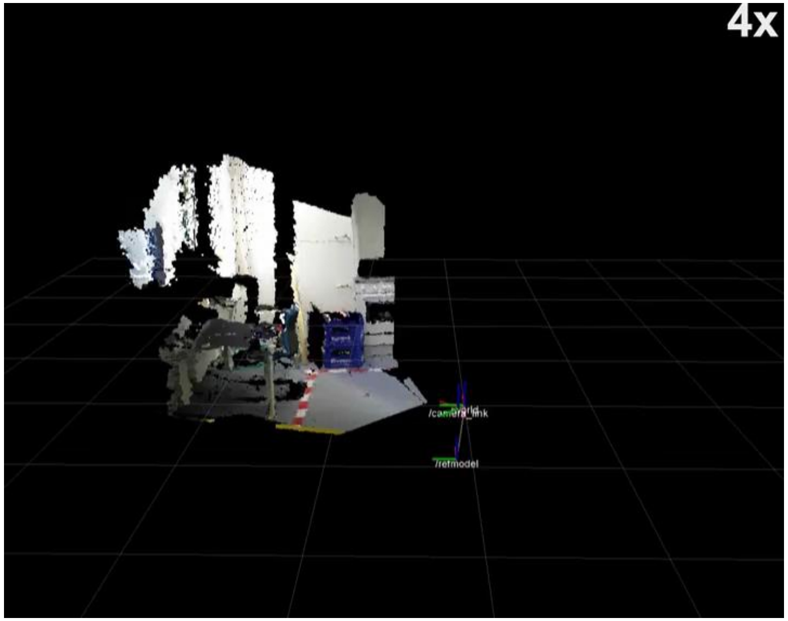
$$\arg\min_{\xi} E(\xi) = \arg\min_{\xi} \frac{1}{M} \sum_{ij} V(X(\xi, (i, j), I_d))^2$$

3D Reconstruction from a Quadrocopter

[Bylow et al., RSS 2013]

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

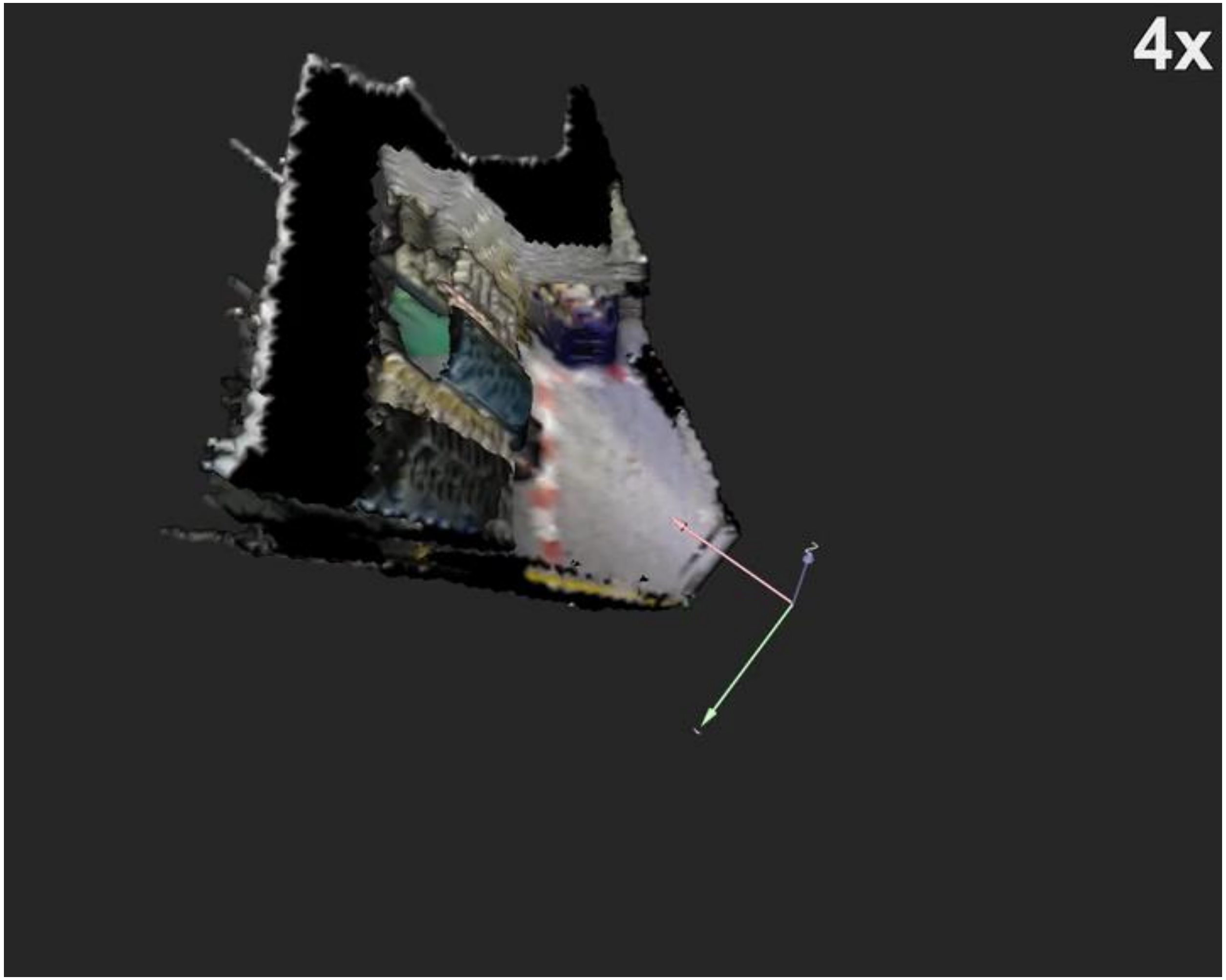
external view



estimated pose

Resulting 3D Model

[Bylow et al., RSS 2013]

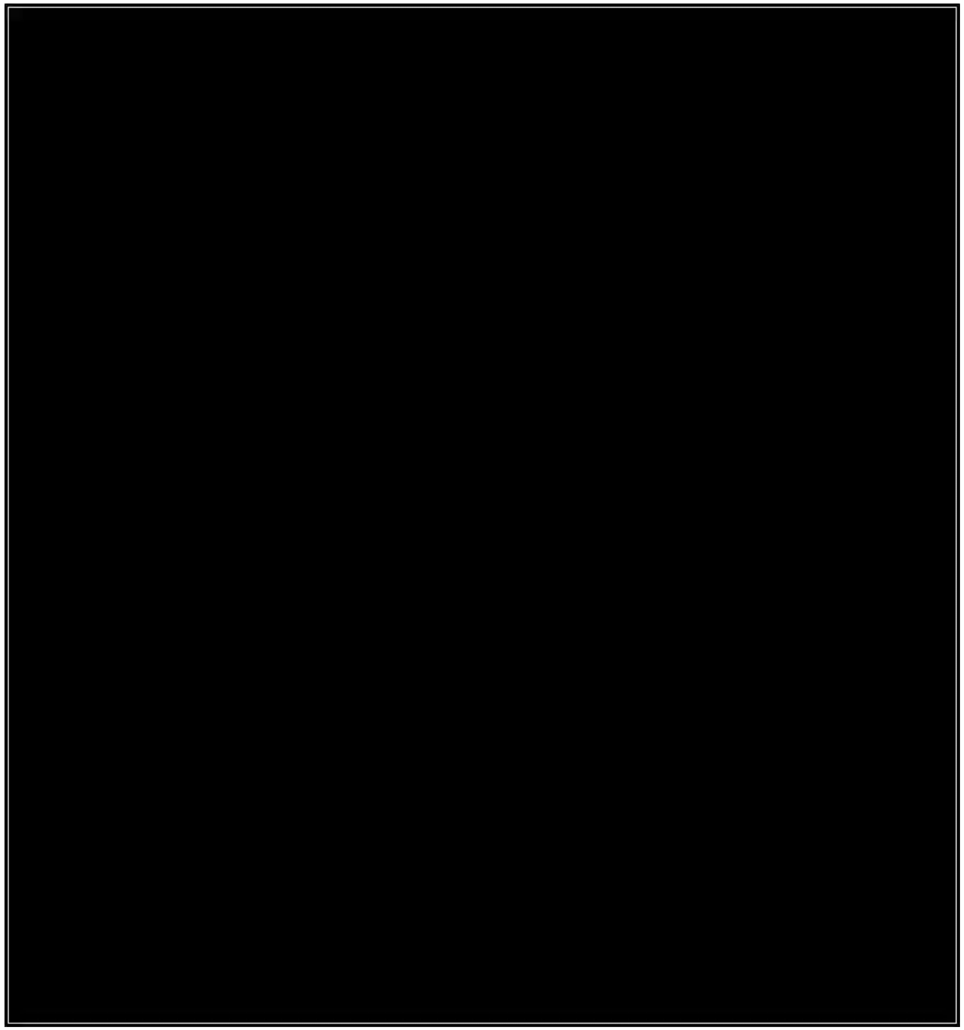


Dense SLAM

[under review]

- Dense Visual Odometry
  - Input: Two RGB-D frames
  - Output: Relative pose
- Use this in pose graph SLAM
  - Select keyframes
  - Detect loop-closures
  - Build and optimize pose graph

Results: 3D Pose Graph



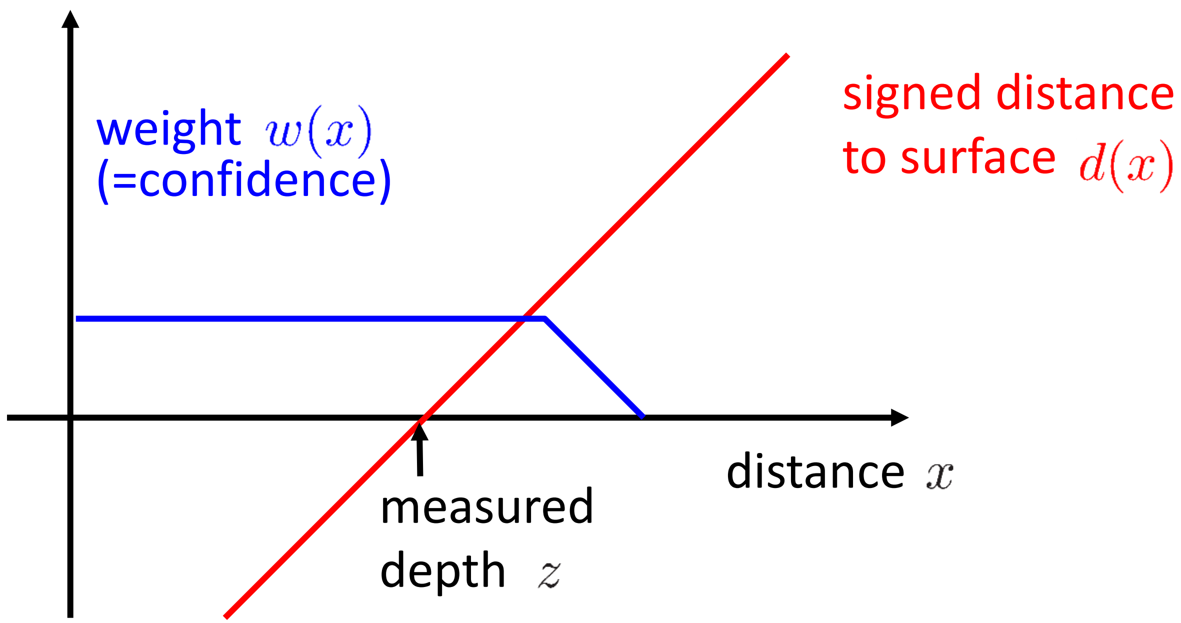
High-Quality 3D Reconstruction

- **We have:** Optimized pose graph
- **We want:** High-resolution 3D map

- **Problem:** High-resolution voxel grids consume much memory (grows cubically)
  - 512^3 voxels, 24 byte per voxel → 3.2 GB
  - 1024^3 voxels, 24 byte per voxel → 24 GB
  - …

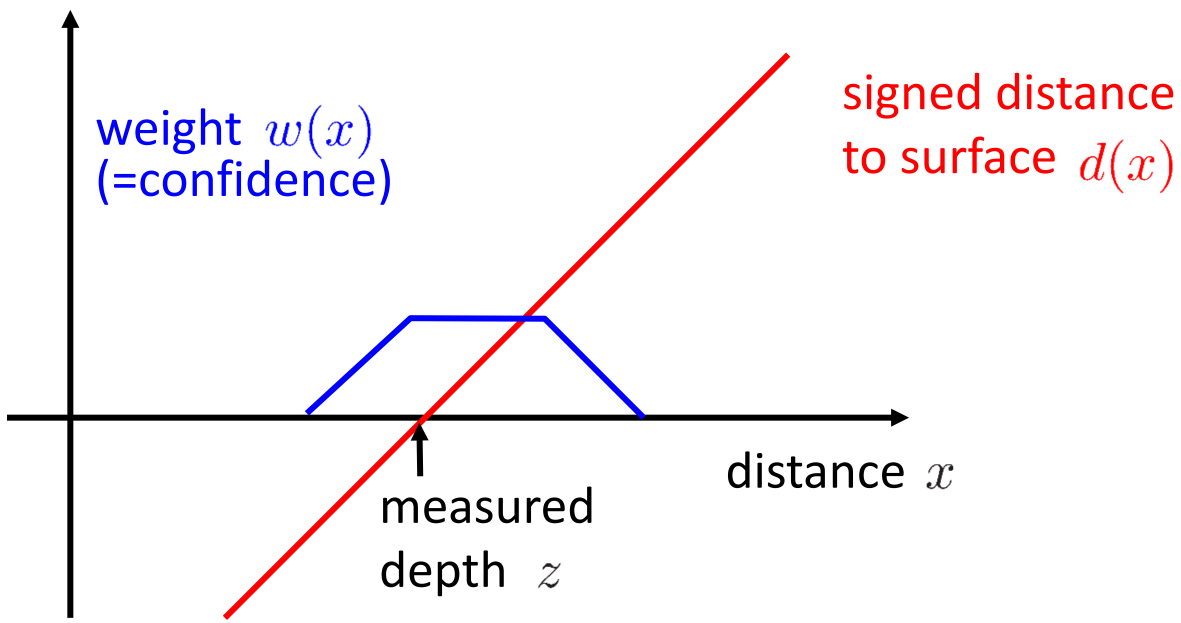
High-Resolution 3D Reconstruction

- **Idea:** Only allocate voxels that are close to the surface (narrow band)
- Before:



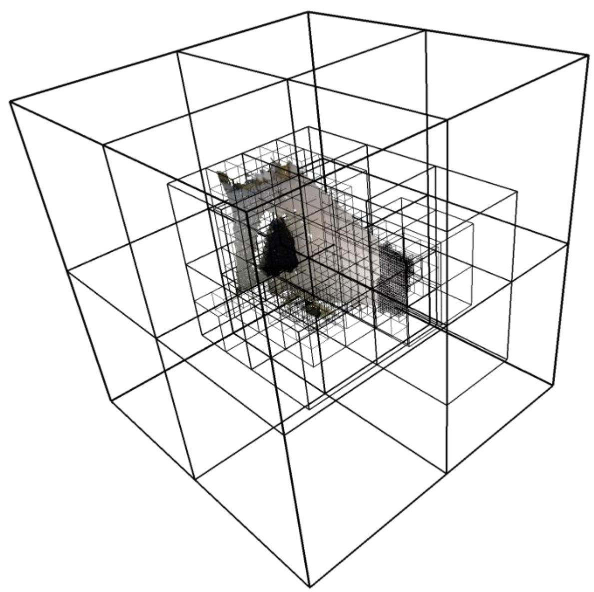
High-Resolution 3D Reconstruction

- **Idea:** Only allocate voxels that are close to the surface (narrow band)
- After:



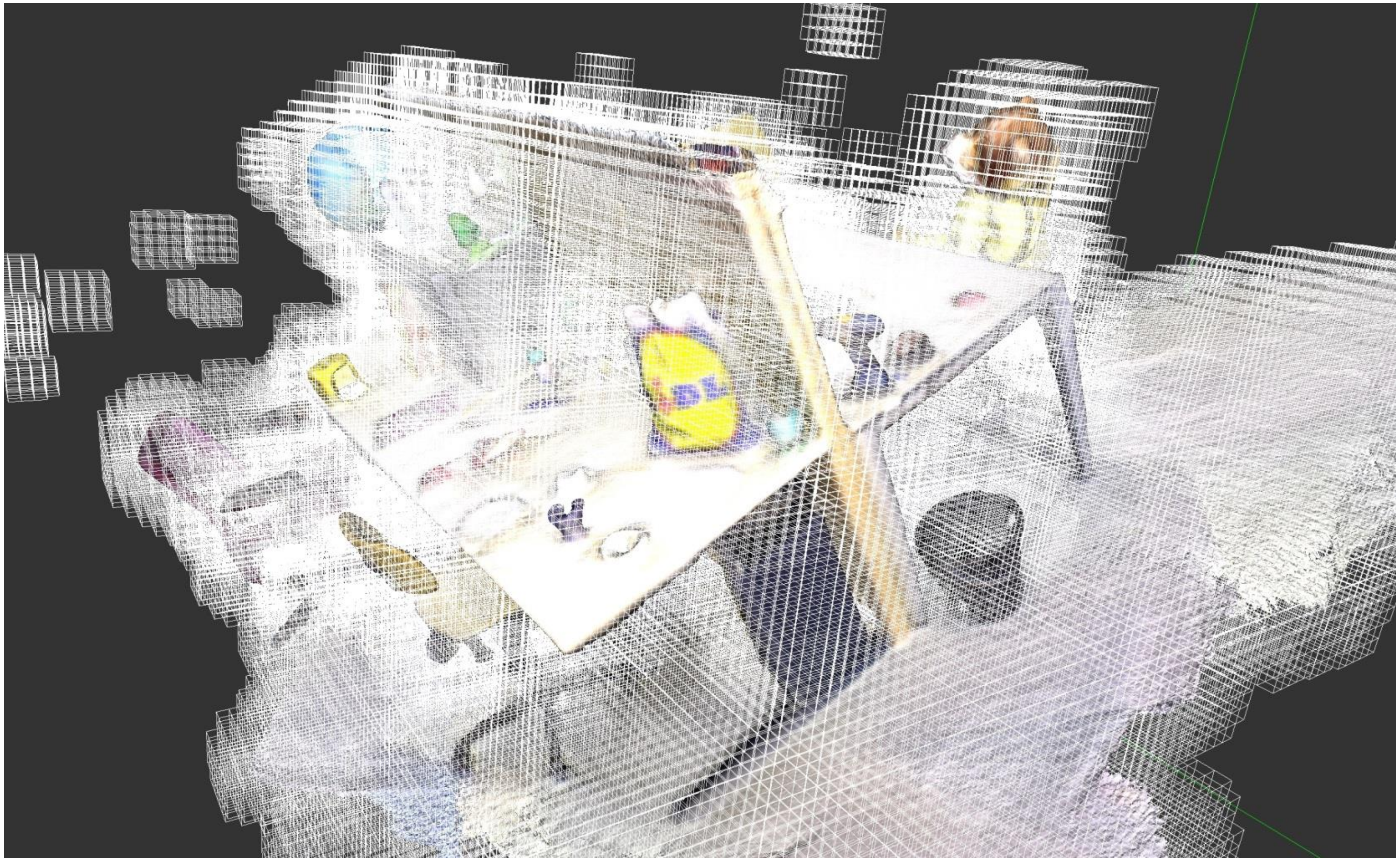
High-Resolution 3D Reconstruction

- Save data in oct-tree data structure
- Leafs are only allocated when needed
- Tree can grow dynamically (no fixed size)

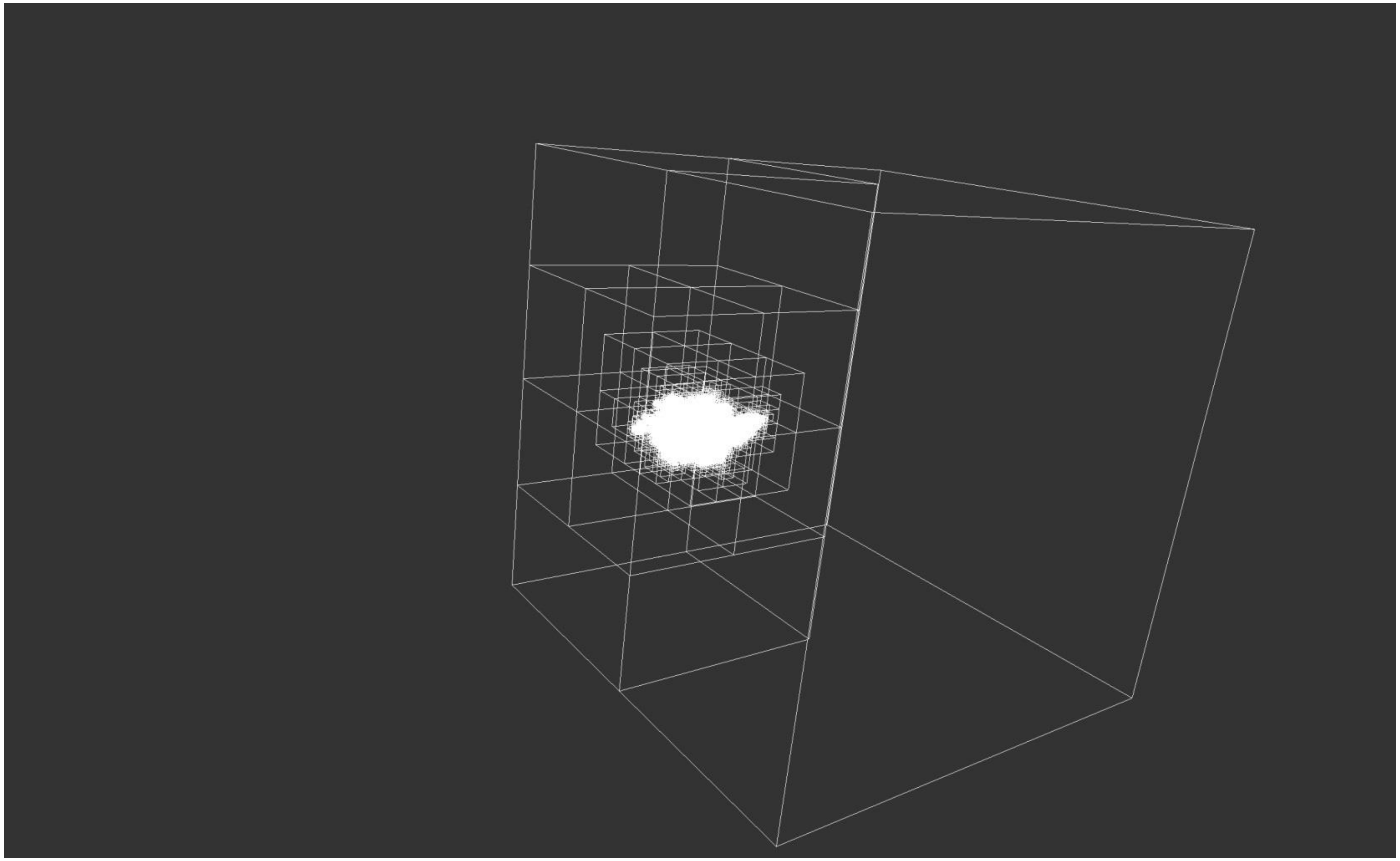


Example: Triangle Mesh

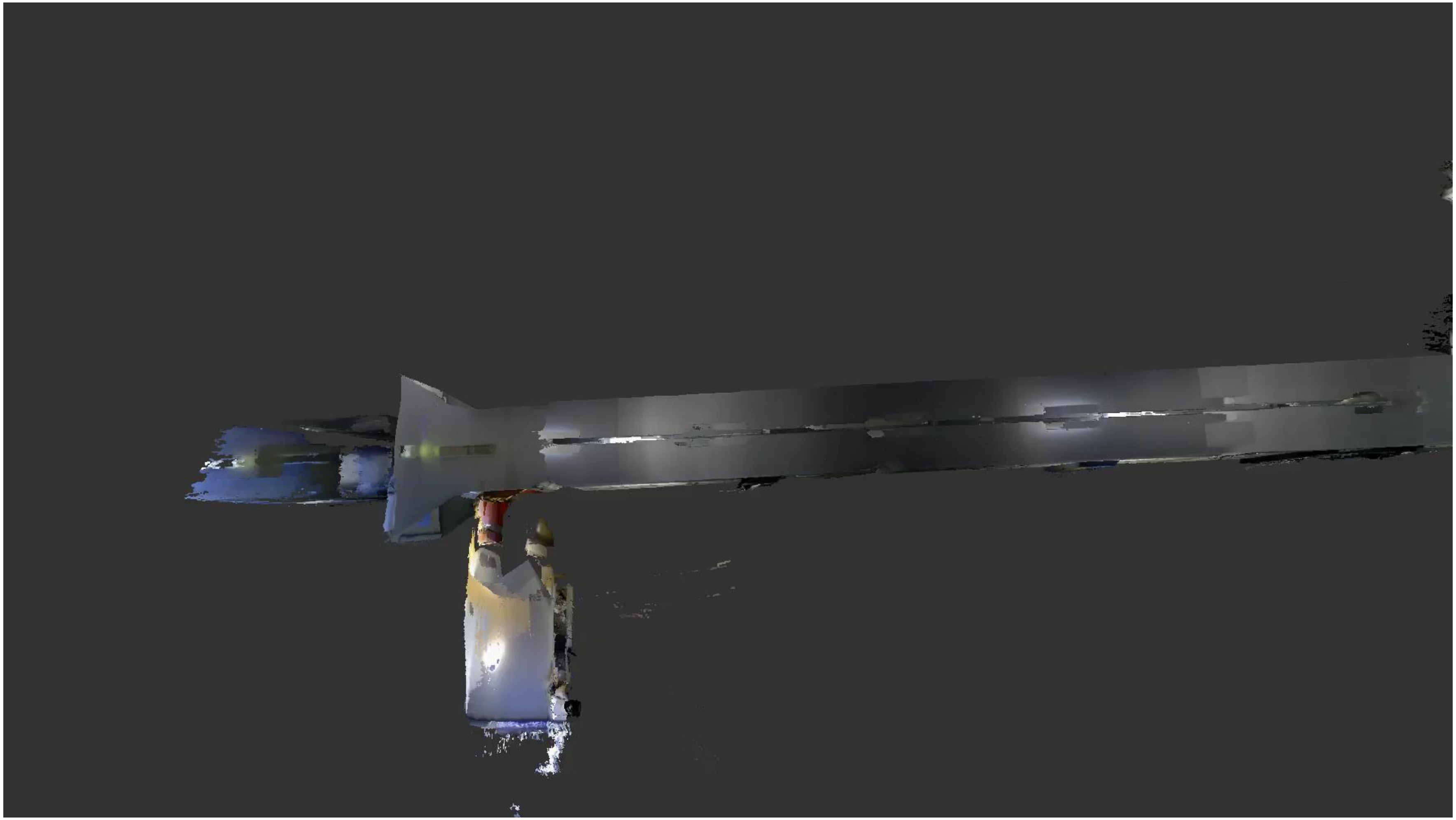
Example: Allocated Leafs



Example: Tree

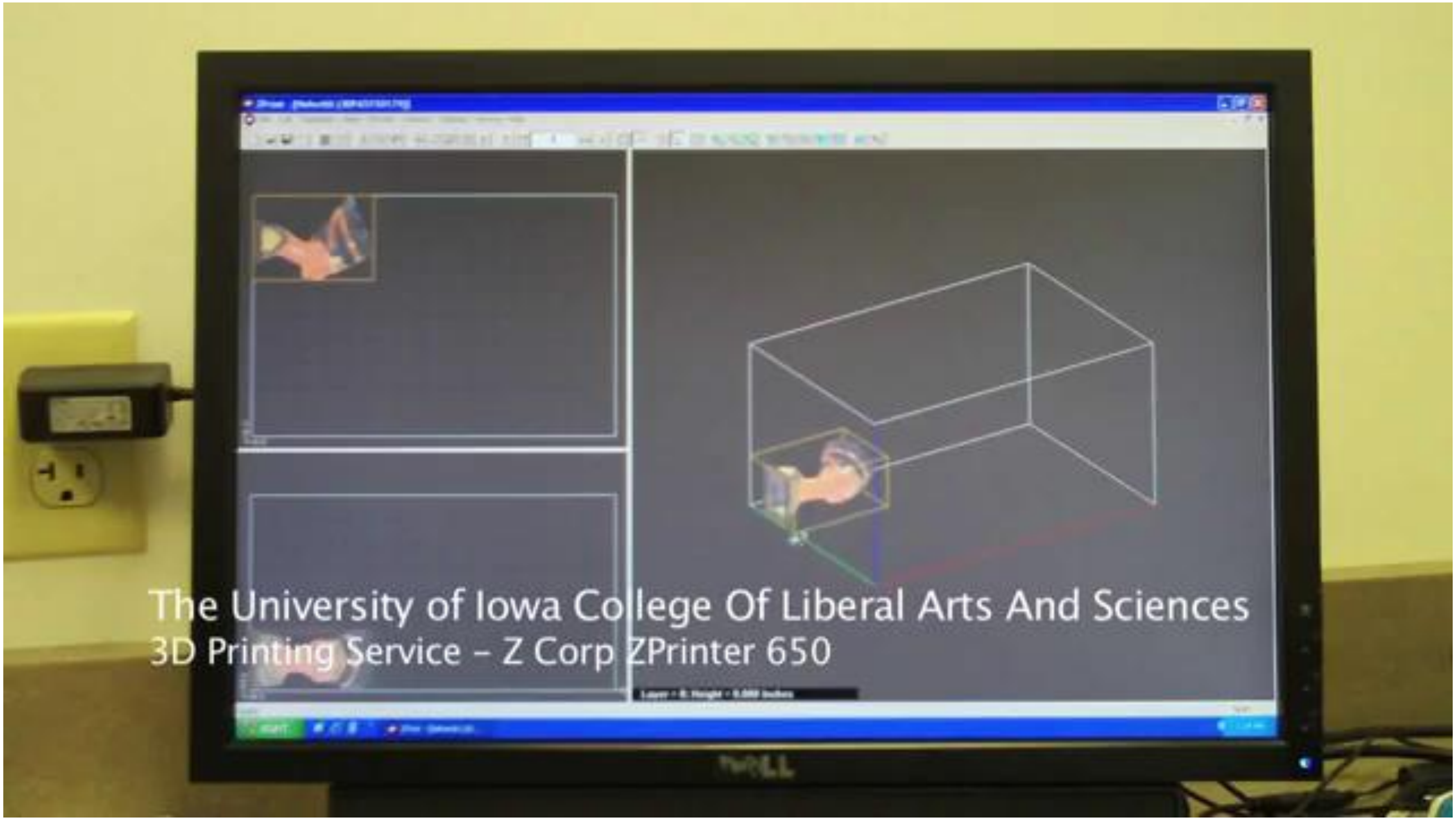


Resulting 3D Model

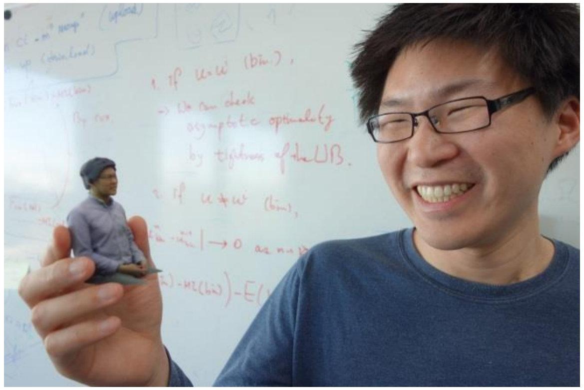


Let's Scan a Person!
[Sturm et al., GCPR 2013]

3D Color Printing



Can We Print These Models in 3D?



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

Hands-On: Afternoon Session

- Team up (2-3 persons in each team)
- Goal for the afternoon: **Autonomous Flight**
- Two options, full code available for both
  1. Marker-based flight
     - Kalman filter, PID controller
     - Easy to understand and to extend
  2. Marker-less flight
     - Based on PTAM
     - Really nice demo

Conclusion

- Visual navigation for quadrocopters
- Much open-source software → easy entry
- Dense methods bear a large potential
  - Dense camera tracking
  - Dense 3D reconstruction
  - Dense SLAM
- Many directions for future research
- Contact us if you are interested in collaboration