

# Simultaneous Localization and Mapping (SLAM)

Stefan Holzer

Computer Aided Medical Procedures (CAMP)  
Technischer Universität München, Germany

# Introduction

## SLAM

### Problem of SLAM:

If placed in an unknown environment is it possible to:

- incrementally build a consistent map of this environment (= **mapping**) while
- simultaneously determining the own location within this map (= **localization**)?

### Tasks:

- Localization and mapping
- Loop detection and closure:
  - detect if we are visiting a place that we have visited (and mapped) before and ensure that map is consistent
- Re-initialization:
  - while localization is usually done by tracking features (i.e. making use of the previous system state) re-initialization cannot make use of tracking, i.e. it is a detection problem



# Introduction

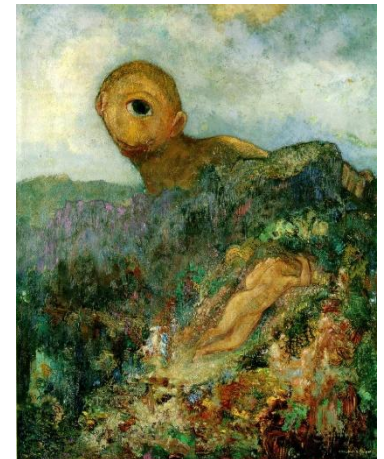
## MonoSLAM

### There are many different approaches depending on:

- the type of sensors (camera, odometry, laser range finder, ...) used and
- the methods used to solve the different tasks

### In this lecture we concentrate on the *MonoSLAM* approach:

- Visual SLAM with only one camera
- Extended Kalman Filter is used to maintain map and current location
- Addresses only the localization and mapping task



### This leads to the following setting:

- camera (e.g. mounted on a robot) is moving in an unknown environment that contains visually measurable landmarks (e.g. feature points)
- All measurements are relative to the camera pose

# MonoSLAM

## Basic Principle

### Basic principle of MonoSLAM:

- A state vector is used to represent the map and the current camera location
- This state vector is managed and updated using an **Extended Kalman Filter (EKF)**
- For each image frame:
  - the EKF is used to **predict the state vector** for this frame based on the previous state
  - based on this prediction the **expected measurements are computed** using the EKF
  - these expected measurements are used to **make the real measurements** by applying a local search based on the predicted uncertainty of the measurement
  - the differences between the expected and real measurements are used to **update the predicted state**

# MonoSLAM

## Requirements

**For the Kalman filter it is necessary to define**

- how we model the camera and the map (= **state vector**),
- what kind of dynamics we expect (= **process or motion model**) and
- how the sensors we use take measurements of the map (= **observation or measurement model**).

**This is what we get:**

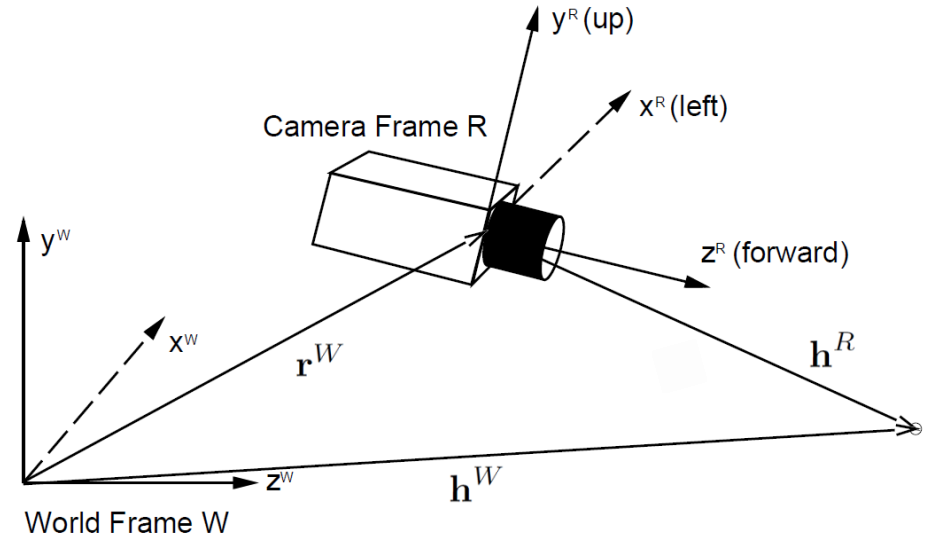


# Modeling the World

## Cameras and Points

Position and orientation of camera:

$$\mathbf{x}_P = \begin{pmatrix} \mathbf{r}^W \\ \mathbf{q}^{WR} \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ q_0 \\ q_x \\ q_y \\ q_z \end{pmatrix}$$



where

- $\mathbf{r}^W$  is the position of the camera and
- $\mathbf{q}^{WR}$  is the orientation represented using a quaternion

# Modeling the World

## Cameras and Points

### 3D Points relative to camera:

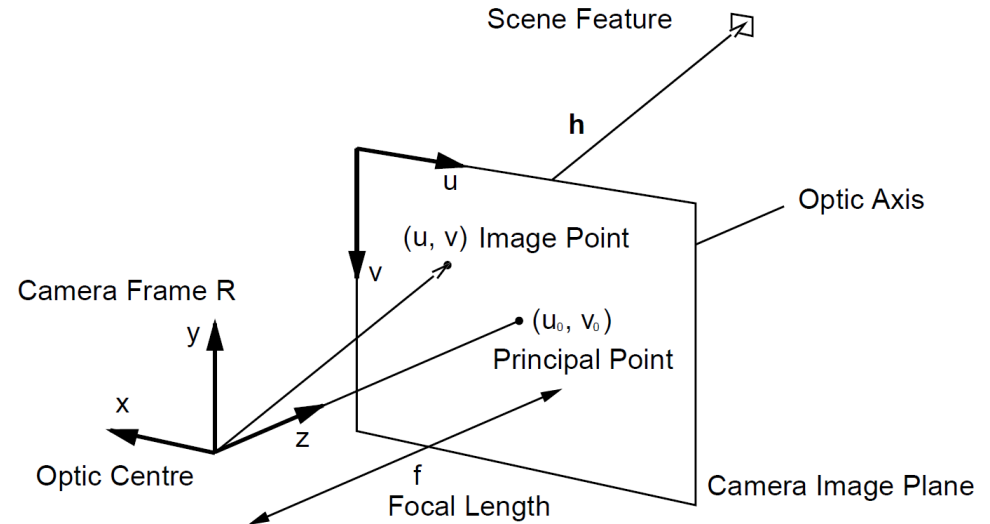
$$\mathbf{h}^R = \begin{pmatrix} h_x^R \\ h_y^R \\ h_z^R \end{pmatrix}$$

### Measurements in the image:

$$\mathbf{h} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 - f k_u \frac{h_x^R}{h_z^R} \\ v_0 - f k_v \frac{h_y^R}{h_z^R} \end{pmatrix}$$

where

- $f$  is the focal length
- $k_u$  is defined by the physical width of a image pixel
- $k_v$  is defined by the physical height of a image pixel



# Modeling the World

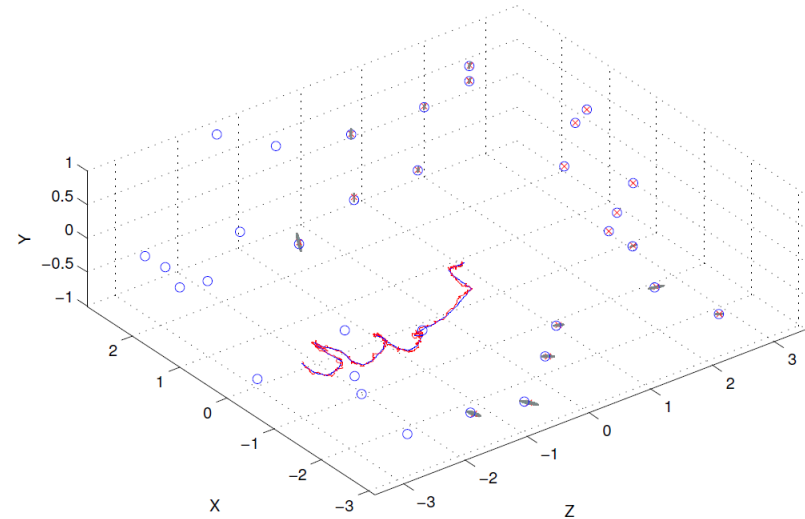
## Cameras and Points

The **state vector** models the whole world (camera and points):

$$\mathbf{x} = \begin{pmatrix} \mathbf{r}^W \\ \mathbf{q}^{WR} \\ \mathbf{h}_1^W \\ \dots \\ \mathbf{h}_n^W \end{pmatrix}$$

where

- $\mathbf{r}^W$  and  $\mathbf{q}^{WR}$  define the position and orientation of the camera and
- $\mathbf{h}_1^W \dots \mathbf{h}_n^W$  define the positions of the landmarks within the map relative to a common reference (world) coordinate frame



**Modeling uncertainty:** additional to the state vector we also store a corresponding covariance matrix that represents the uncertainty of the state.

# Modeling the World

## Cameras and Points

The **state vector** models the whole world (camera and points):

$$\mathbf{x} = \begin{pmatrix} \mathbf{r}^W \\ \mathbf{q}^{WR} \\ \mathbf{h}_1^W \\ \dots \\ \mathbf{h}_n^W \end{pmatrix}$$



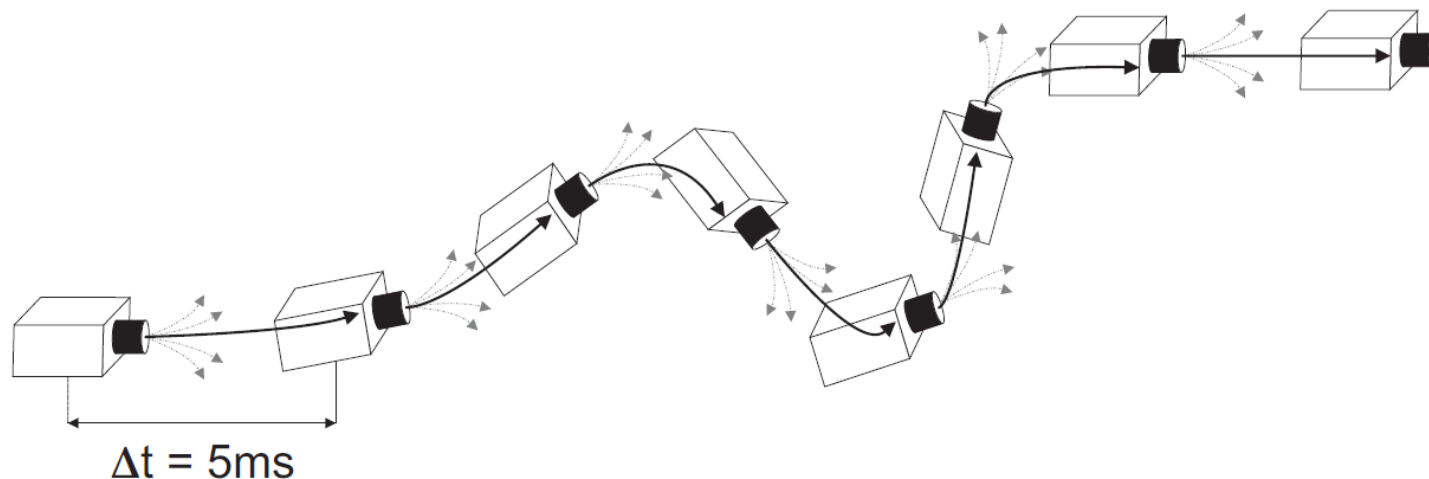
The state vector will be updated at each frame using the Kalman filter.

➔ For this we have to model the system dynamics and measurement process.

# Modeling the World

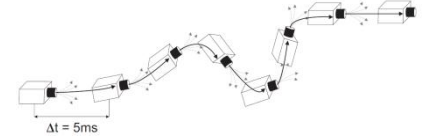
## System Dynamics

- Also known as **Motion** or **Process Model**
- Describes how the state variables of the process evolve in time, e.g. what types of camera movements we expect:



# Modeling the World

## System Dynamics



- If the last state is given as  $\hat{\mathbf{x}}_{k-1|k-1}$  then the process model is

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_{k-1} \mathbf{u}_{k-1} + \mathbf{w}_{k-1}$$

**Uncorrelated process noise**  
Zero mean Gaussian white noise  
with covariance  $\mathbf{Q}_{k-1}$

**Vector of control inputs**

In case we can control the movement in some way (e.g. if the camera is mounted on a robot)

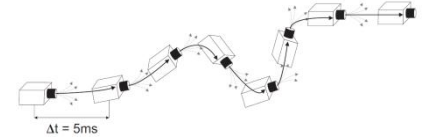
**Linear mapping between control inputs and state-space**

**State transition matrix**

Linear model of the state change over time

# Modeling the World

## System Dynamics



- If the last state is given as  $\hat{\mathbf{x}}_{k-1|k-1}$  then the process model is

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_{k-1} \mathbf{u}_{k-1} + \mathbf{w}_{k-1}$$

- The uncertainty of the new state is computed as

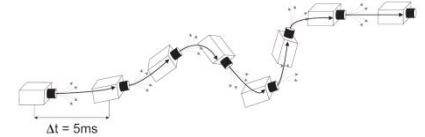
$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_{k-1}$$

where  $\mathbf{P}_{k-1|k-1}$  is the uncertainty of the last state

- **Remember:** The state vector consists of camera and landmarks  
 → We have to model the dynamics of both

# Modeling the World

## System Dynamics



- Example: **Constant Position Model**

Assume that the state vector does not change over time (useful for e.g. landmarks)

$$\mathbf{F} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} = \mathbf{I}_{n \times n}$$

- Example: **Constant Velocity Model**

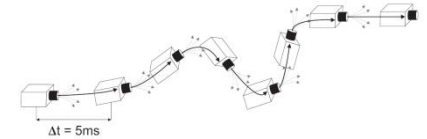
Assume that the object has a constant velocity (useful for e.g. cameras)

→ the state vector is augmented with the state change over time (= velocity)

$$\mathbf{F} = \begin{pmatrix} \mathbf{I}_{n \times n} & \mathbf{I}_{n \times n} \Delta t \\ \mathbf{0} & \mathbf{I}_{n \times n} \end{pmatrix}$$

# Modeling the World

## System Dynamics



- We can also mix different models

$$\mathbf{F} = \begin{pmatrix} \mathbf{I}_{c \times c} & \mathbf{I}_{c \times c} \Delta t & 0 \\ 0 & \mathbf{I}_{c \times c} & 0 \\ 0 & 0 & \mathbf{I}_{3n \times 3n} \end{pmatrix}$$

— **Constant velocity model**  
 for camera

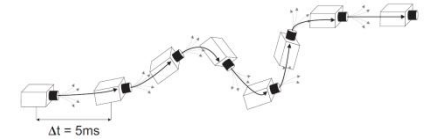
— **Constant position model**  
 for landmarks

- In case of the **Constant Velocity Model** the state vector is extended with the “velocity” of the parameters (change of parameters in time)
- This leads to the new extended camera state vector

$$\mathbf{x}_v = \begin{pmatrix} \mathbf{r}^W \\ \mathbf{q}^{WR} \\ \mathbf{v}^W \\ \boldsymbol{\omega}^W \end{pmatrix}$$

# Modeling the World

## System Dynamics



- This leads to the new extended camera state vector

$$\mathbf{x}_v = \begin{pmatrix} \mathbf{r}^W \\ \mathbf{q}^{WR} \\ \mathbf{v}^W \\ \omega^W \end{pmatrix}$$

- Problem:** Orientation is represented by an quaternion  
 → We cannot update the camera state linearly:

$\mathbf{q}((\omega^W + \Omega^W) \Delta t)$   
denotes the quaternion  
defined by the angle-axis  
vector  $(\omega^W + \Omega^W) \Delta t$

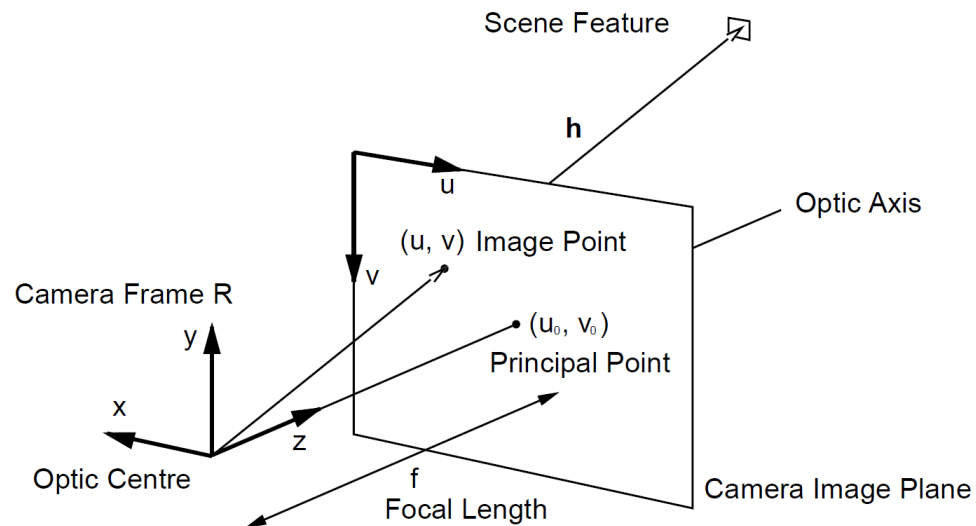
$$\mathbf{x}_{v,\text{new}} = \begin{pmatrix} \mathbf{r}_{\text{new}}^W \\ \mathbf{q}_{\text{new}}^{WR} \\ \mathbf{v}_{\text{new}}^W \\ \omega_{\text{new}}^W \end{pmatrix} = \begin{pmatrix} \mathbf{r}^W + (\mathbf{v}^W + \mathbf{V}^W) \Delta t \\ \mathbf{q}^{WR} \times \mathbf{q}((\omega^W + \Omega^W) \Delta t) \\ \mathbf{v}^W + \mathbf{V}^W \\ \omega^W + \Omega^W \end{pmatrix}$$

- We will see later how to solve this using the *Extended Kalman Filter*

# Modeling the World

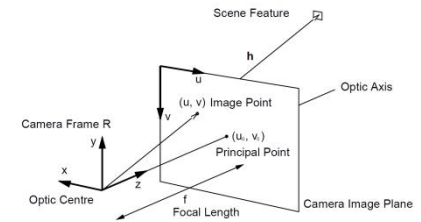
## Observation Model

- The observation model describes how the sensor measures the current state



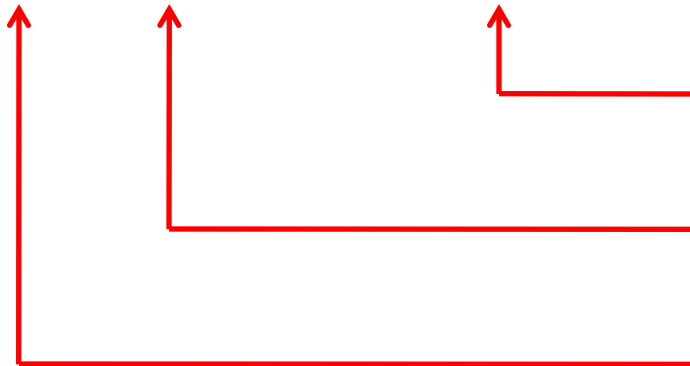
# Modeling the World

## Observation Model



- The observation model describes how the sensor measures the current state
- Given a state  $\hat{\mathbf{x}}_{k|k-1}$  the measurement model is

$$\hat{\mathbf{z}}_k = \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} + \mathbf{v}_{k-1}$$



**Uncorrelated measurement noise**

Zero mean Gaussian white noise with covariance  $\mathbf{R}_k$

**Observation matrix**

Describes the linear measurement process

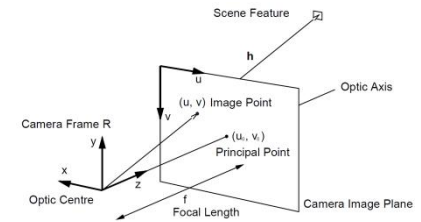
**Expected measurement**

- The uncertainty of the expected measurement is

$$\mathbf{Z}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k$$

# Modeling the World

## Observation Model



- **Problem:** the considered measurement process is non-linear  
 → we will see later how to solve this using the *Extended Kalman Filter*
- If we consider an orthogonal projection with the principal point in the origin (which is linear)

$$\mathbf{h} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -fk_u h_x^R \\ -fk_v h_y^R \end{pmatrix}$$

- we get the following observation matrix

$$\mathbf{H} = \begin{pmatrix} -fk_u & 0 & 0 \\ 0 & -fk_v & 0 \end{pmatrix}$$

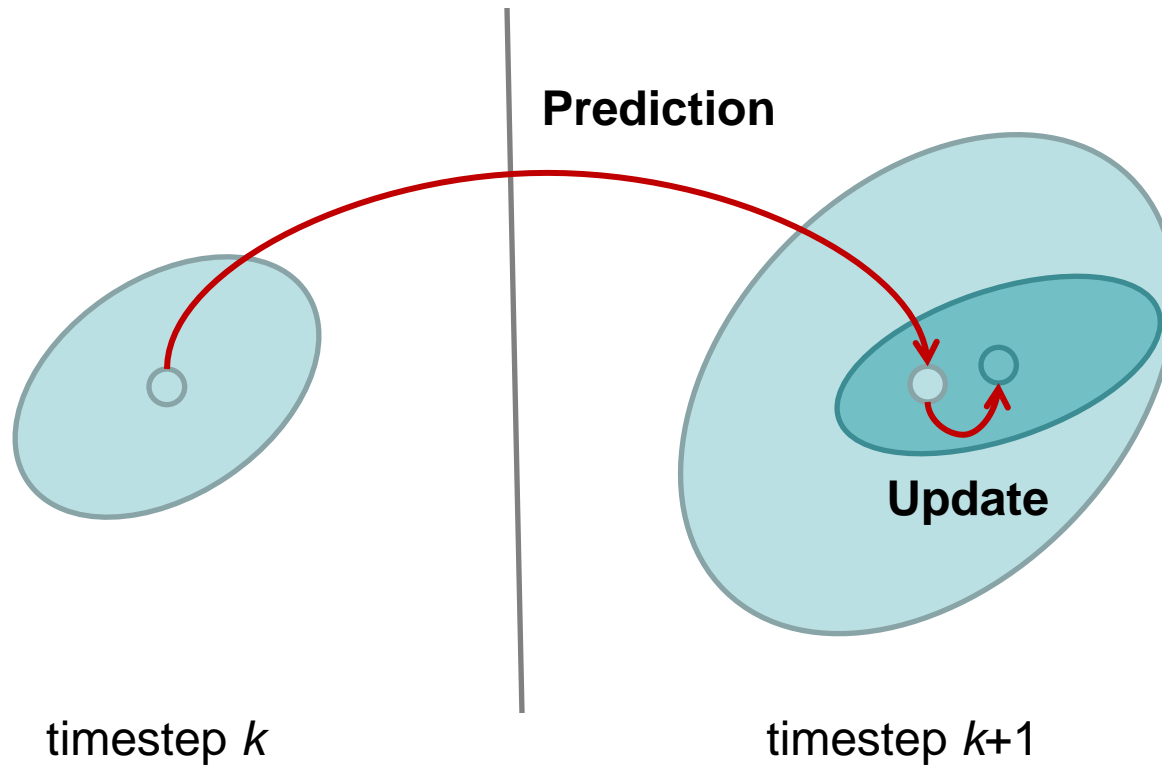
# Kalman Filter

## Overview

- Provides efficient recursive equations to estimate the state of a process
  - only the estimated state from the previous time step and the current measurement are needed to compute the estimate of the current state
- State of the filter is represented by
  - Estimate of the state at time  $k$
  - Error covariance matrix (measure for the accuracy of the state estimate)
- **Assumption:** process can be described using linear models
- Non-linear models can be handled using the Extended Kalman Filter (see later)
- Given an initial state estimate the following steps are iteratively applied:
  - Predict
  - Observe
  - Update

# Kalman Filter

## Prediction-Update Scheme



# Kalman Filter

## Prediction

- After setting initial values for the state  $\hat{\mathbf{x}}_{k-1|k-1}$  and its uncertainty  $\mathbf{P}_{k-1|k-1}$  a prediction is generated for

- The a priori state estimate

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_{k-1} \mathbf{u}_{k-1}$$

- The a priori observation

$$\hat{\mathbf{z}}_k = \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}$$

- The a priori state covariance (describes how uncertain the obtained states are)

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_{k-1}$$

# Kalman Filter

## Observation

- Using the predicted observation a real observation is made
- Discrepancy between actual measurement and predicted measurement is called innovation
- The innovation and innovation covariance matrix are computed as

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k$$

# Kalman Filter

## Update

- The state estimate and the corresponding covariance are updated using

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^T + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^T$$

where  $\mathbf{K}_k$  is the Kalman gain and is computed as

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$$

# Kalman Filter

## Summary

- **Predict new state:**

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_{k-1} \mathbf{u}_{k-1}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_{k-1}$$

- **Predict measurement and compute innovation:**

$$\hat{\mathbf{z}}_k = \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}$$

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \hat{\mathbf{z}}_k$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k$$

- **Update:**

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^T + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^T$$

# Kalman Filter

## Influence of measurement noise

- Very good measurement  $\mathbf{R}_k \rightarrow \mathbf{0}$

$$\mathbf{S}_k \approx \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T$$

$$\mathbf{K}_k \approx \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$$

$$= \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{H}_k^{-T} \mathbf{P}_{k|k-1}^{-1} \mathbf{H}_k^{-1}$$

$$= \mathbf{H}_k^{-1}$$

$$\hat{\mathbf{x}}_{k|k} \approx \hat{\mathbf{x}}_{k|k-1} + \mathbf{H}_k^{-1} \tilde{\mathbf{y}}_k$$

$$= \hat{\mathbf{x}}_{k|k-1} + \mathbf{H}_k^{-1} (\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1})$$

$$= \hat{\mathbf{x}}_{k|k-1} + \mathbf{H}_k^{-1} \mathbf{z}_k - \hat{\mathbf{x}}_{k|k-1}$$

$$= \mathbf{H}_k^{-1} \mathbf{z}_k$$

New state depends only on current measurement

# Kalman Filter

## Influence of measurement noise

- Very bad measurement  $\mathbf{R}_k \rightarrow \infty$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k$$

$$\mathbf{S}_k \rightarrow \infty$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$$

$$\mathbf{S}_k^{-1} \rightarrow \mathbf{0} \quad \mathbf{K}_k \rightarrow \mathbf{0}$$

$$\hat{\mathbf{x}}_{k|k} \approx \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$$

$$\approx \hat{\mathbf{x}}_{k|k-1}$$

New state depends  
only on prediction

# Extended Kalman Filter

## Overview

- Extended Kalman Filter is non-linear extension of Kalman Filter
- **Idea:** local linear approximation of the non-linearity
- This way the prediction of the process state and measurement can be modeled using non-linear *differentiable* functions  $f$  and  $h$

$$\hat{\mathbf{x}}_{k|k-1} = f\left(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}\right) + \mathbf{w}_{k-1}$$

$$\hat{\mathbf{z}}_k = h\left(\hat{\mathbf{x}}_{k|k-1}\right) + \mathbf{v}_{k-1}$$

remind:

- $\mathbf{u}_{k-1}$  is a vector of control inputs
- $\mathbf{v}_{k-1}$  is a vector of uncorrelated observation errors with zero means and covariance  $\mathbf{R}_k$
- $\mathbf{w}_{k-1}$  is a vector of uncorrelated process noise errors with zero means and covariance  $\mathbf{Q}_{k-1}$

# Extended Kalman Filter

## Prediction and Update

- The functions  $f$  and  $h$  can be used to compute the predicted state and measurement

$$\hat{\mathbf{x}}_{k|k-1} = f \left( \hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1} \right)$$

$$\hat{\mathbf{z}}_k = h \left( \hat{\mathbf{x}}_{k|k-1} \right)$$

- *But:* they cannot be used to compute the corresponding covariances
- *Solution:* the Jacobians of the functions are used

$$\mathbf{F}_k = \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k} \quad \mathbf{H}_k = \left. \frac{\partial h}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}}$$

- All other equations stay the same

# Extended Kalman Filter

## Prediction and Update

- Based on this linearization we can model our original observation model

$$\mathbf{h} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 - f k_u \frac{h_x^R}{h_z^R} \\ v_0 - f k_v \frac{h_y^R}{h_z^R} \end{pmatrix}$$

- The corresponding Jacobian is

$$\mathbf{H}_k = \left. \frac{\partial \mathbf{h}}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}} = \begin{pmatrix} -f \frac{k_u}{h_z^R} & 0 & f k_u \frac{h_x^R}{(h_z^R)^2} \\ 0 & -f \frac{k_v}{h_z^R} & f k_v \frac{h_y^R}{(h_z^R)^2} \end{pmatrix}$$

# Extended Kalman Filter

## Prediction and Update

- The Jacobian can be easily computed using Matlab (if Symbol-Toolbox is available), Maple, ....:

```
>> syms f ku kv hx hy hz u0 v0;
```

```
>> u = u0 - f*ku*(hx/hz);
>> v = v0 - f*kv*(hy/hz);
```

}

Define measurement function

```
>> H(1,1) = diff(u, hx);
>> H(1,2) = diff(u, hy);
>> H(1,3) = diff(u, hz);
>> H(2,1) = diff(v, hx);
>> H(2,2) = diff(v, hy);
>> H(2,3) = diff(v, hz);
```

}

Compute partial derivatives

```
>> H
```

```
H =
```

```
[ -(f*ku)/hz,          0, (f*hx*ku)/hz^2]
[           0, -(f*kv)/hz, (f*hy*kv)/hz^2]
```

# Extended Kalman Filter

## Prediction and Update

- We can do the same for the prediction of the state vector

$$f(\mathbf{x}_v) = \begin{pmatrix} \mathbf{r}^W + \mathbf{v}^W \Delta t \\ \mathbf{q}^{WR} \times \mathbf{q}(\omega^W \Delta t) \\ \mathbf{v}^W \\ \omega^W \end{pmatrix}$$

- The corresponding Jacobian is

$$\mathbf{F}_k = \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k-1|k-1}} = \begin{pmatrix} \mathbf{I}_{3 \times 3} & 0 & \mathbf{I}_{3 \times 3} \Delta t & 0 \\ 0 & \frac{\partial(\mathbf{q}^{WR} \times \mathbf{q}(\omega^W \Delta t))}{\partial \mathbf{q}^{WR}} & 0 & \frac{\partial(\mathbf{q}^{WR} \times \mathbf{q}(\omega^W \Delta t))}{\partial \omega^W} \\ 0 & 0 & \mathbf{I}_{3 \times 3} & 0 \\ 0 & 0 & 0 & \mathbf{I}_{3 \times 3} \end{pmatrix}$$

# Mapping and Localization

## Short Summary

- Now we know how to maintain an estimate of the map and of our current location:
  - Predict process state at next time step
  - Compute expected observation
  - Use expected observation to measure real observation
  - Update state using discrepancy between expected and real observation
- But, there are still open questions:
  - How to add new landmarks to the map?  
Two cases:
    - Map is empty
      - We cannot make observations
      - We cannot use the filter to obtain camera pose estimates
    - Map is not empty
      - We get camera pose estimates from the filter

# Mapping and Localization

## How to add new landmarks to the map?

- If the map is empty we have to estimate the camera pose of each frame
  - Track feature points
  - Compute poses relative to first camera poses using epipolar geometry
    - Basically the same as “*Structure from Motion*”
- Having the camera poses the problem is the same for both cases:  
Compute 3D location of new landmarks:
  - Position of landmarks is estimated in multiple frames with known camera pose
  - 3D Position is obtained from these 2D Positions:
    - By triangulating the point
    - By using a own Extended Kalman Filter for each new landmark
  - If approximate 3D Position is estimated the landmark is added to the map

# Mapping and Localization

## How to make a measurement?

- Basic Approach:
  - Store local image patch for each feature
  - Warp patch according to predicted pose
  - Apply a local search around the expected position of the observation and according to the expected uncertainty
  - Select the location with the lowest difference (e.g. with the highest NCC value)

Prewarp patches:



Search for patch matches:



# Mapping and Localization

## How to make a measurement?

- Active Matching:
  - Store and warp local image patch as for the basic approach
  - Measure the landmark which is expected to give most information per search area size
  - Use this measurement to update the other predictions and their uncertainties
  - Iterate until all landmarks are measured or enough information is gathered
  - *Disadvantage:* It is expensive to iteratively update the state

Measure first patch:



Update and match next patch:



# Mapping and Localization

## Active Matching

Imperial College  
London

### **ACTIVE MATCHING**

Margarita Chli and Andrew J. Davison  
ECCV 2008

## Re-initialization / Re-localization

- Tracking using the EKF can fail due to
  - Camera motions that do not fit to the motion model (e.g. constant velocity model)
  - Occlusions
  - Insufficient landmarks (e.g. if a white wall is imaged)
  - Camera is “kidnapped”
- The problem becomes a detection and pose estimation problem:
  - We have given a set of features with known 3D positions
  - Find the corresponding projected features in the current image
  - Many possible solutions:
    - Bag of words
    - SIFT, SURF
    - ....
  - Restriction: Feature-descriptors have to be computed online during the SLAM process

# Re-initialization / Re-localization



## Visual SLAM applications of loop-closure detection: recovering from kidnapping in EKF SLAM

Adrien Angeli, David Filliat, Stéphane Doncieux  
and Jean-Arcady Meyer



EKF-based visual SLAM [Davison et al., IAV2004]:

- provides estimation of 3D/6DoF camera pose
- builds a 3D metric map of visual keypoints
- 30Hz processing

Incremental vision-based loop-closure detection  
[Angeli et al., ICRA2008]:

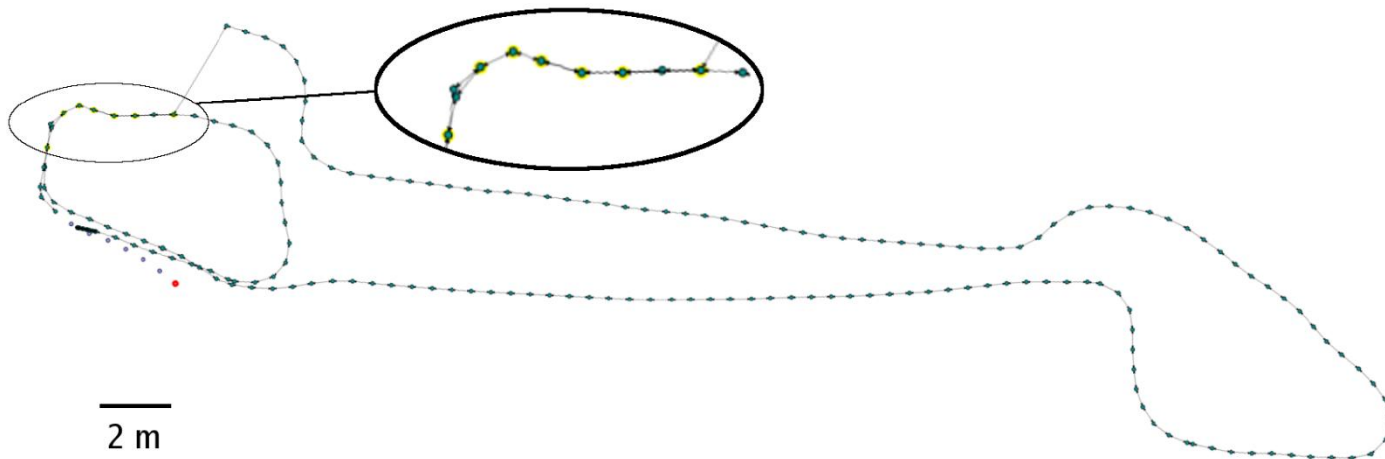
- computes the probability that the current image comes from a known location
- encodes images using visual bags of words [Filliat, ICRA2007]
- 1Hz processing

When the camera is kidnapped:

- EKF estimation is automatically stalled
- loop-closure detection makes it possible to recover a consistent camera pose

## Loop Detection and Loop Closure

- Small errors can sum over time
  - ➔ leads to errors in landmark locations within the map
  - ➔ same real location can have two instances within the map after a large loop



- Task is similar to re-initialization, but without failed tracking
  - ➔ continuously look for correspondences between current image and mapped features
- Map should be corrected

# Loop Detection and Loop Closure



AnimatLab



## Real-Time Visual Loop-Closure Detection

Adrien Angeli,  
Stéphane Doncieux,  
Jean-Arcady Meyer

Université Pierre et Marie Curie - Paris 6  
FRE 2507, ISIR, 4 place Jussieu, F-75005  
Paris, France.  
firstname.lastname@upmc.fr

David Filliat

ENSTA  
32, bvd Victor, F-75015  
Paris, France  
david.filliat@ensta.fr

# Loop Detection and Loop Closure



AnimatLab



## Real-Time Visual Loop-Closure Detection

Adrien Angeli,  
Stéphane Doncieux,  
Jean-Arcady Meyer

Université Pierre et Marie Curie - Paris 6  
FRE 2507, ISIR, 4 place Jussieu, F-75005  
Paris, France.  
firstname.lastname@upmc.fr

David Filliat

ENSTA  
32, bvd Victor, F-75015  
Paris, France  
david.filliat@ensta.fr

# EKF-based SLAM

## Properties

### Advantages:

- Other sensors can be easily added

### Limitations:

- EKF can only handle a very limited number of features in real time
- Process can only be approximately modeled

## Other Approaches

### Parallel Tracking and Mapping (PTAM)

- Two parallel threads:
  - Tracking feature points
  - Bundle adjustment on key frames
- Can handle more features than EKF, but the number of key frames is critical
- More accurate than *MonoSLAM* approach when considering the same amount of computation time



## Other Approaches

### Parallel Tracking and Mapping (PTAM)

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