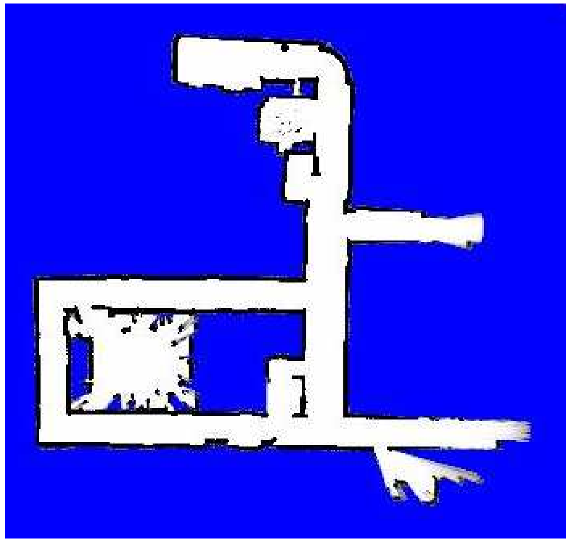
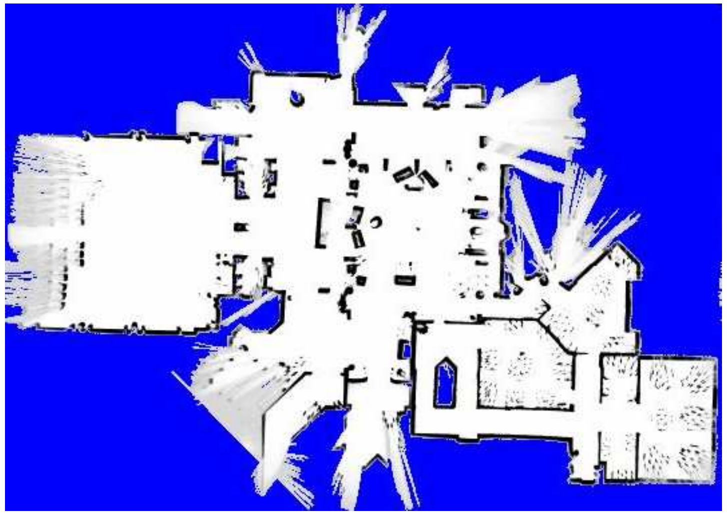
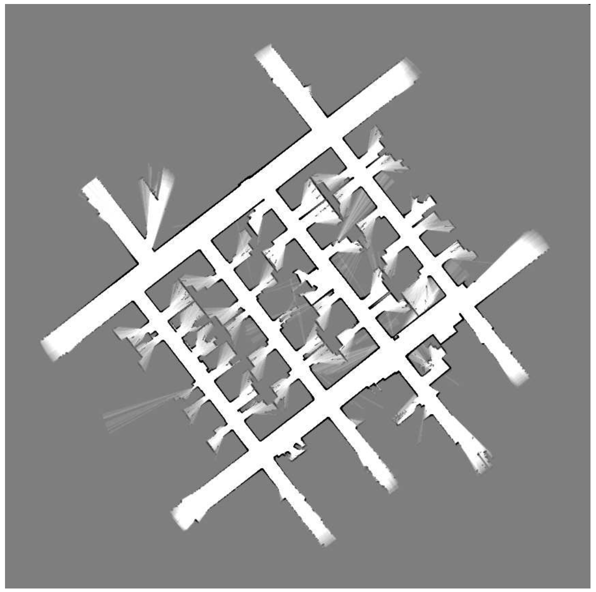
Introduction to Kalman Filters and SLAM

November 20, 2008







Acknowledgements:

Slides derived/borrowed from multiple sources:

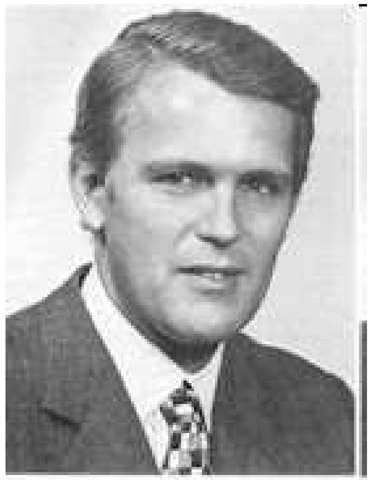
R. Siegwart and I. Nourbakhsh

P. Rybski

G. Welch and G. Bishop

Introduction to Kalman Filter

- Developed by Rudolf E. Kalman
  - *Born in 1930 in Hungary*
  - *Education: B.S., M.S. from MIT; Ph.D. (1957) from Columbia*
  - *Developed Kalman Filter in 1960-61*



- Filter: just a fancy word for an algorithm that takes an input (typically, a sensor signal) and calculates a function of that input

- Kalman Filter: an efficient, recursive filter that estimates the state of a dynamic system from a series of noisy measurements

2008

What is a Kalman Filter used for?

Broadly, it’s useful for any type of tracking application, such as:

- *Tracking missiles*
- *Estimating position of aircraft*
- *Surveillance of highway traffic*
- *GPS-based motion estimation*
- *Economics applications (e.g., estimating demand for international reserves)*
- *Mobile robot localization!*

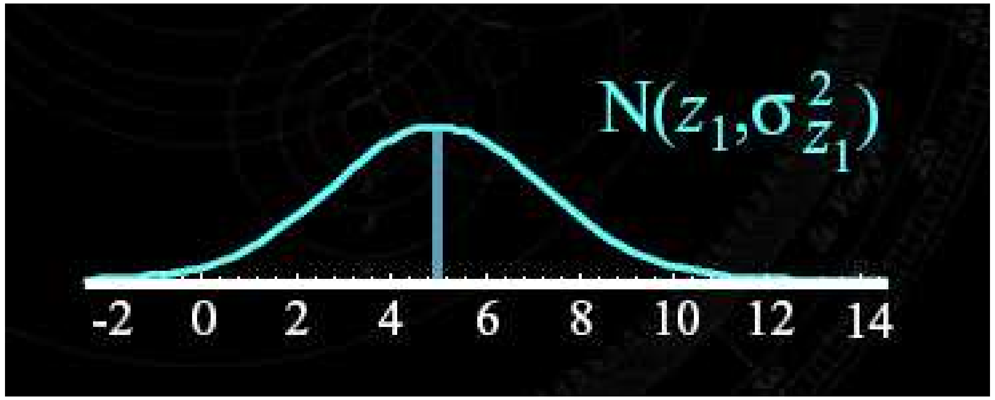
We’ll derive Kalman Filter update equations

- First, look at measurement updates without motion
- Then, see how motion affects estimation updates
- Then, put all into form of Kalman Filter update equations

Basic idea: Combining observation estimates (no motion for now)

- Let’s say we make an observation measurement. This measurement is assumed to obey a Gaussian (i.e., Normal) distribution, defined by 2 parameters (mean and variance):

$$z_1, \sigma_{z_1}^2$$



- So, our first estimate of the location of the measurement is given as:

$$\hat{x}_1 = z_1$$

$$\hat{\sigma}_1^2 = \sigma_{z_1}^2$$

Fusing in 2nd observation measurement

- Now, we make a 2nd measurement, which is described by a second Gaussian, with parameters:

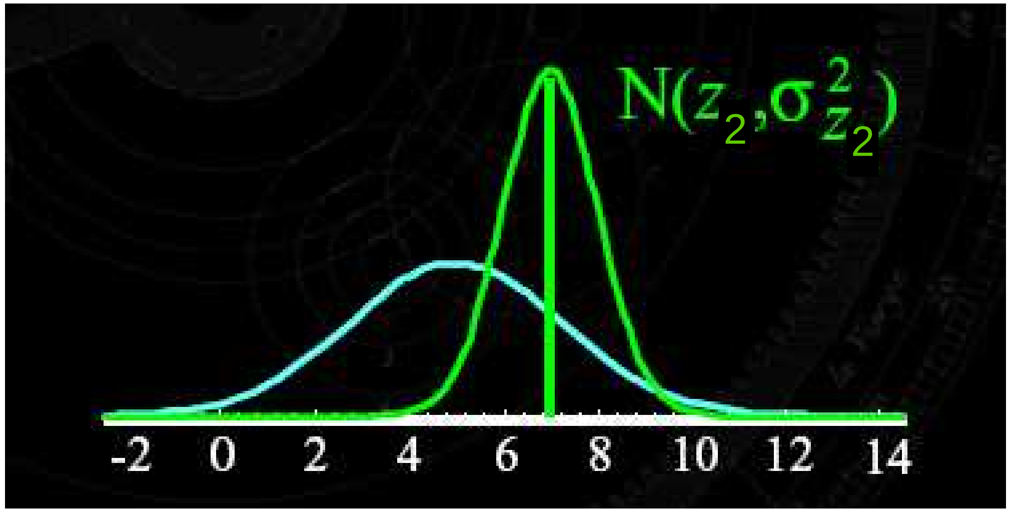
$$z_2, \sigma_{z_2}^2$$

- How do we generate our new position estimate?

$$\hat{x}_2 = \ldots?$$

$$\hat{\sigma}_2^2 = \ldots?$$

- We need to combine (i.e., *fuse*) this new information with our previous information



Combining estimates

- Remember, our first measurement was described by: $z_1, \sigma_{z_1}^2$

  which led to estimate:

  $$\hat{x}_1 = z_1$$

  $$\hat{\sigma}_1^2 = \sigma_{z_1}^2$$

- Our second measurement is described by:

  $$z_2, \sigma_{z_2}^2$$

- So, we combine new measurement with prior estimate to obtain updated estimate:

  $$\hat{x} = \hat{x}_2 = \hat{x}_1 + K_2(z_2 - \hat{x}_1)$$

  $$K_2 = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_{z_2}^2}$$

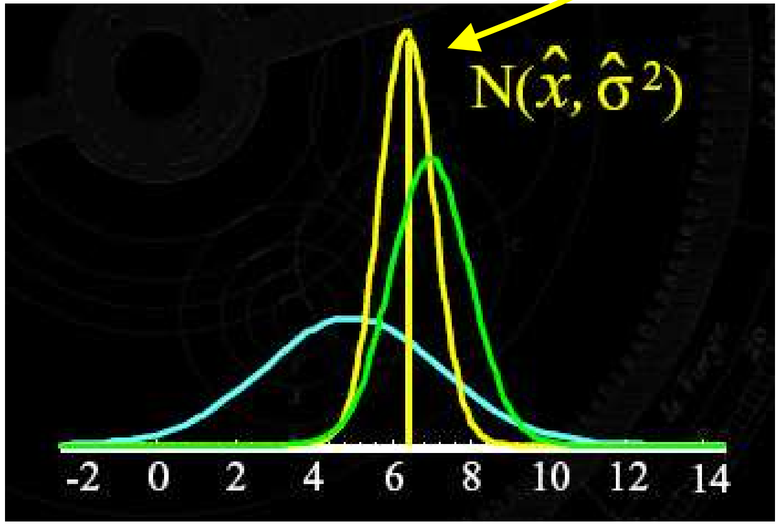
- Then we combine variances to get:

  $$\frac{1}{\sigma_2^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_{z_2}^2}$$

  which simplifies to:

  $$\hat{\sigma}^2 = \sigma_2^2 = \frac{\sigma_1^2 \sigma_{z_2}^2}{\sigma_1^2 + \sigma_{z_2}^2}$$

Combined estimate!



In more general notation for going from step *k* to *k*+1

$$\hat{x}_{k+1} = \hat{x}_k + K_{k+1}(z_{k+1} - \hat{x}_k)$$

$$\sigma^2_{k+1} = \sigma^2_k - K_{k+1}\sigma^2_k$$

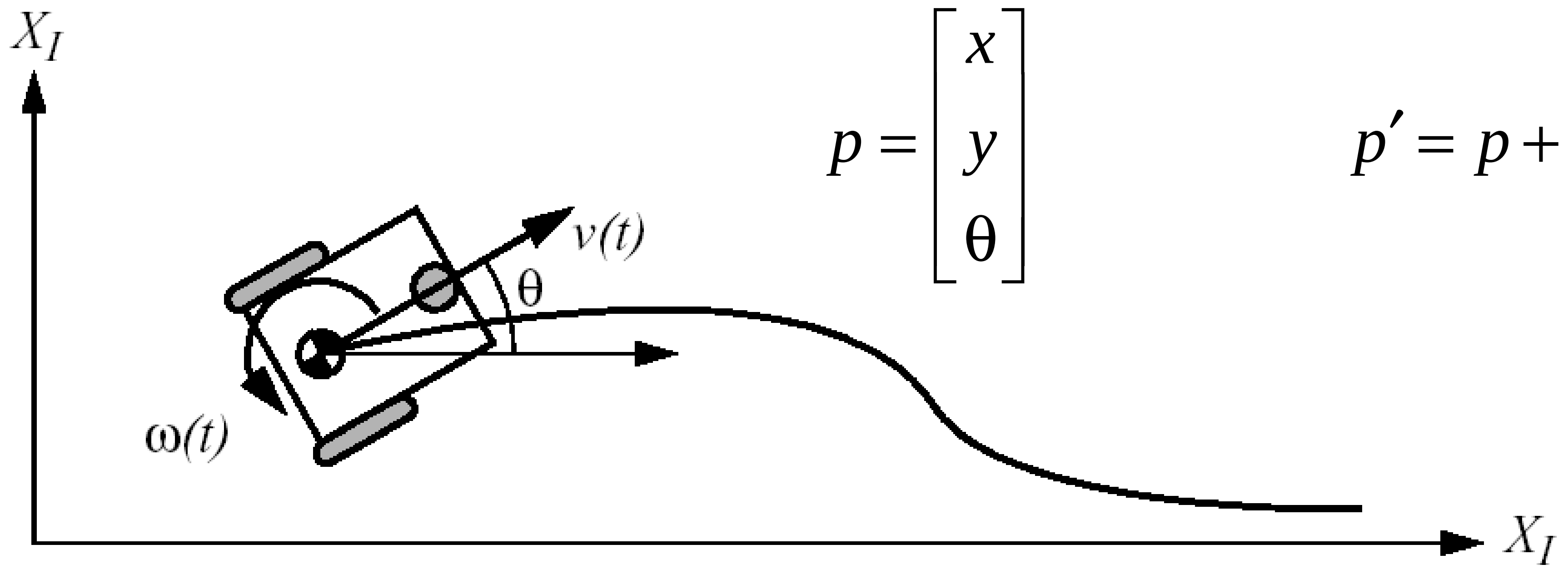
where:

$$K_{k+1} = \frac{\sigma_k^2}{\sigma_k^2 + \sigma_z^2}$$

Says that our best estimate of the new state at time *k*+1 equals the best prediction of the state before the new measurement (z_{k+1}) is taken, plus a correction term, which is an optimal weighting of the difference between the new measurement and the prior best estimate

But, for robot localization, the robot is typically moving

- Means that not all the difference is due to observation error
- Some of the difference is due to robot motion
- Thus, we make use of our motion model
- For a differential drive robot, we have:



$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \qquad p' = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}$$

Motion Model for Differential Drive Robot

- Kinematics

$\Delta s_r =$ travel distance of right wheel (measured by odometry)
$\Delta s_l =$ travel distance of left wheel (measured by odometry)
$b =$ distance between 2 wheels

$$\Delta x = \Delta s \cos(\theta + \Delta\theta / 2)$$

$$\Delta y = \Delta s \sin(\theta + \Delta\theta / 2)$$

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b}$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$$

$$p' = f(x, y, \theta, \Delta s_r, \Delta s_l) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2}\cos\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2}\sin\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix}$$

Eqn. (5.7)

(Note: The following slides use $\hat{x}$ instead of $\hat{p}$ (or p′) to represent estimated position)

What we know…
What we don’t know…

- We know what the control inputs of our process are
  - *We know what we’ve told the system to do and have a model for what the expected output should be if everything works right*

- We don’t know what the noise in the system truly is
  - *We can only estimate what the noise might be and try to put some sort of upper bound on it*

- When estimating the state of a system, we try to find a set of values that comes as close to the truth as possible
  - *There will always be some mismatch between our estimate of the system and the true state of the system itself. We just try to figure out how much mismatch there is and try to get the best estimate possible*

How to incorporate dynamic motion

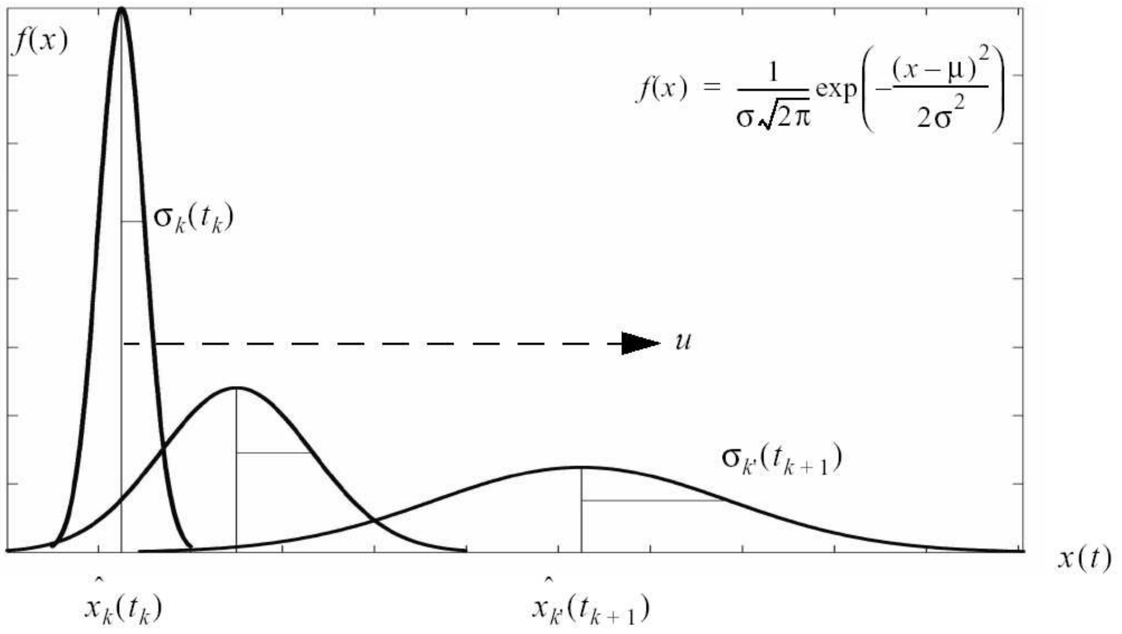
- Dynamic Prediction (robot moving): the amount of robot motion between times *k* and *k*+1 is described by:

$$\frac{dx}{dt} = u + w$$

u = velocity; w = noise

- Starting at time *k*, if we know the variance of robot position (σ_k^2) and of motion (σ_w^2), we get:

$$\hat{x}_{k'} = \hat{x}_k + u(t_{k+1} - t_k)$$
$$\sigma_{k'}^2 = \sigma_k^2 + \sigma_w^2[t_{k+1} - t_k]$$



This is the estimated position and variance just as the new measurement is taken at time *k*+1. It describes the growth of position error until a new measurement is taken.

Substituting into our previous equations…

- Previously, we had derived the following (without motion):

$$\boxed{\hat{x}_{k+1} = \hat{x}_k + K_{k+1}(z_{k+1} - \hat{x}_k)} \qquad \boxed{\sigma^2_{k+1} = \sigma^2_k - K_{k+1}\sigma^2_k} \qquad \boxed{K_{k+1} = \frac{\sigma^2_k}{\sigma^2_k + \sigma^2_z}}$$

- Now, adding in dynamic prediction of motion, we get:

$$\begin{aligned}\hat{x}_{k+1} &= \hat{x}_{k'} + K_{k+1}(z_{k+1} - \hat{x}_{k'}) \\ &= [\hat{x}_k + \underbrace{u(t_{k+1} - t_k)}_{}] + K_{k+1}[z_{k+1} - \hat{x}_k - \underbrace{u(t_{k+1} - t_k)}_{}]\end{aligned}$$

estimated change in position due to motion

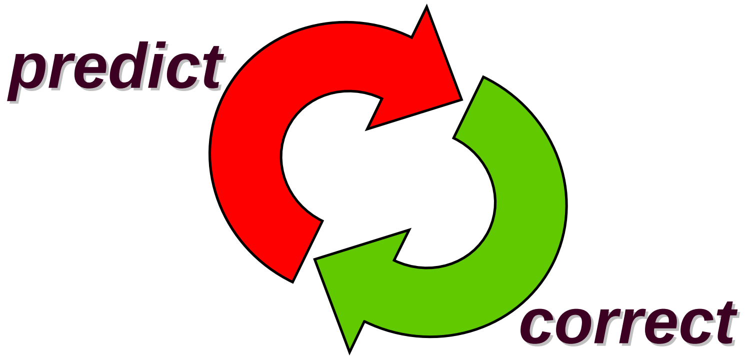
$$K_{k+1} = \frac{\sigma^2_{k'}}{\sigma^2_{k'} + \sigma^2_z} = \frac{\sigma^2_k + \underline{\sigma^2_w[t_{k+1} - t_k]}}{\sigma^2_k + \underline{\sigma^2_w[t_{k+1} - t_k]} + \sigma^2_z}$$

variance of the motion

Putting this all together into a Kalman Filter for robot localization!

- General Idea: Predict → Correct

- Kalman Filter operates by iteratively:
  - *Predicting the new state and its uncertainty (from the motion model)*
  - *Correcting with the new observation measurement*



Key differences between Kalman filter for localization and Markov localization

- Main difference: perception update process
- In Markov localization:
  - *Entire set of sensor measurements (at a point in time) is used to update each possible robot position, using Bayes formula*
- In Kalman filter:
  - *Perception update is a multi-step process.*
  - *Robot's total sensory input is not treated as a monolithic whole*
  - *Instead, given a set of features, the KF is used to fuse the distance estimate from each feature to a matching object in the map*
  - *Instead of carrying out this matching process for many possible locations (as in Markov approach), the KF does this for one belief state (which is represented by a Gaussian)*

Some more background: Minimum Mean Square Error

Reminder: *the expected value, or mean value, of a Continuous random variable* **x** *is defined as:*

$$E[x] = \int_{-\infty}^{\infty} x p(x) dx$$

Remember: Z represents our sensor measurements

Minimum Mean Square Error

What is the mean of this distribution? $P(x \mid Z)$

This is difficult to obtain exactly. With our approximations, we can get the estimate $\hat{x}$

…such that $E[(x-\hat{x})^2 \mid Z_t]$ is minimized.

According to the **Fundamental Theorem of Estimation Theory** this estimate is:

$$\hat{x}^{MMSE} = E[x \mid Z] = \int_{-\infty}^{\infty} x p(x \mid Z) dx$$

Fundamental Theorem of Estimation Theory

- The minimum mean square error estimator equals the expected (mean) value of *x* conditioned on the observations *Z*

- The minimum mean square error term is quadratic:

$$E[(x-\hat{x})^2 \mid Z_t]$$

  - *Its minimum can be found by taking the derivative of the function w.r.t.* x *and setting that value to 0.*

$$\nabla_x (E[(x-\hat{x})^2 \mid Z]) = 0$$

Kalman Filter Components

(also known as: Way Too Many Variables…)

Motion model (Linear discrete time dynamic system):

$$\overset{\text{State}}{x_{t+1}} = \underset{\text{State transition function}}{F_t}\,\overset{\text{State}}{x_t} + \underset{\text{Control input function}}{B_t}\,\overset{\text{Control input}}{u_t} + \underset{\text{Noise input function with covariance Q}}{G_t}\,\overset{\text{Process noise}}{w_t}$$

Sensor model (Measurement equation):

$$\overset{\text{Sensor reading}}{z_{t+1}} = \underset{\text{Sensor function}}{H_{t+1}}\,\overset{\text{State}}{x_{t+1}} + \overset{\text{Sensor noise with covariance R}}{n_{t+1}}$$

Computing the MMSE Estimate of the State and Covariance

Given a set of measurements: $Z_{t+1} = \{z_i, i \le t+1\}$

According to the Fundamental Theorem of Estimation, the state and covariance will be:

$$\hat{x}^{MMSE} = E[x \mid Z_{t+1}]$$

$$P^{MMSE} = E[(x - \hat{x})^2 \mid Z_{t+1}]$$

We will now use the following notation:

$$\hat{x}_{t+1|t+1} = E[x_{t+1} \mid Z_{t+1}]$$

$$\hat{x}_{t|t} = E[x_t \mid Z_t]$$

$$\hat{x}_{t+1|t} = E[x_{t+1} \mid Z_t]$$

Computing the MMSE Estimate of the State and Covariance

Remember:

$$x_{t+1} = F_t x_t + B_t u_t + G_t w_t$$

$$z_{t+1} = H_{t+1} x_{t+1} + n_{t+1}$$

$$\hat{x}_{t+1|t+1} = E[x_{t+1} \mid Z_{t+1}]$$

$$\hat{x}_{t|t} = E[x_t \mid Z_t]$$

$$\hat{x}_{t+1|t} = E[x_{t+1} \mid Z_t]$$

What is the **minimum mean square error** estimate of the system state and covariance?

$$\hat{x}_{t+1|t} = F_t \hat{x}_{t|t} + B_t u_t$$ Estimate of the state variables

$$\hat{z}_{t+1|t} = H_{t+1} \hat{x}_{t+1|t}$$ Estimate of the sensor reading

$$P_{t+1|t} = F_t P_{t|t} F_t^T + G_t Q_t G_t^T$$ Covariance matrix for the state

$$S_{t+1|t} = H_{t+1} P_{t+1|t} H_{t+1}^T + R_{t+1}$$ Covariance matrix for the sensors

At last! The Kalman Filter…

$$\hat{x}_{t+1|t+1} = E[x_{t+1} \mid Z_{t+1}]$$
$$\hat{x}_{t|t} = E[x_t \mid Z_t]$$
$$\hat{x}_{t+1|t} = E[x_{t+1} \mid Z_t]$$

Propagation (motion model):

$$\hat{x}_{t+1/t} = F_t \hat{x}_{t/t} + B_t u_t$$

- (1) Robot position prediction: State estimate is updated from system dynamics

$$P_{t+1/t} = F_t P_{t/t} F_t^{\ T} + G_t Q_t G_t^{\ T}$$

- Uncertainty associated with this prediction (note that the estimate *GROWS)*

Update (sensor model):

- (2) Observe (no equation here!)

$$\hat{z}_{t+1} = H_{t+1} \hat{x}_{t+1/t}$$

- (3) Compute expected value of sensor reading, using map (H transforms from world frame to sensor frame)

$$r_{t+1} = z_{t+1} - \hat{z}_{t+1}$$

- (4) Match: Compute the difference between expected and “true” observations

$$S_{t+1} = H_{t+1} P_{t+1/t} H_{t+1}^{\ T} + R_{t+1}$$

- Compute covariance (noise) of sensor reading

$$K_{t+1} = P_{t+1/t} H_{t+1}^{\ T} S_{t+1}^{\ -1}$$

- (5) Estimation: apply Kalman filter; Compute the Kalman Gain (how much to correct estimate)

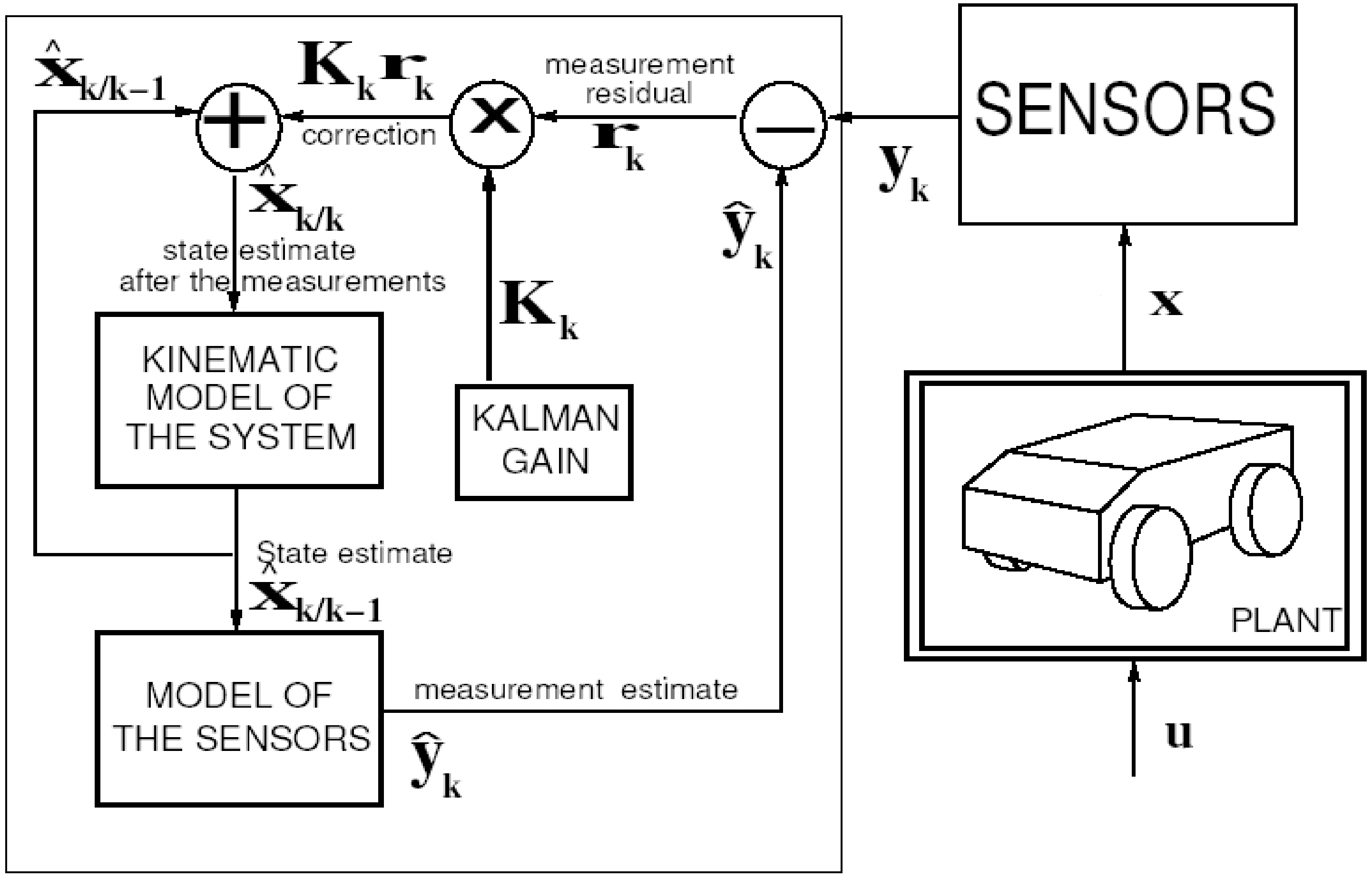
$$\hat{x}_{t+1/t+1} = \hat{x}_{t+1/t} + K_{t+1} r_{t+1}$$

- Multiply residual times gain to correct state estimate

$$P_{t+1/t+1} = P_{t+1/t} - P_{t+1/t} H_{t+1}^{\ T} S_{t+1}^{\ -1} H_{t+1} P_{t+1/t}$$

- Uncertainty estimate SHRINKS

Kalman Filter Block Diagram



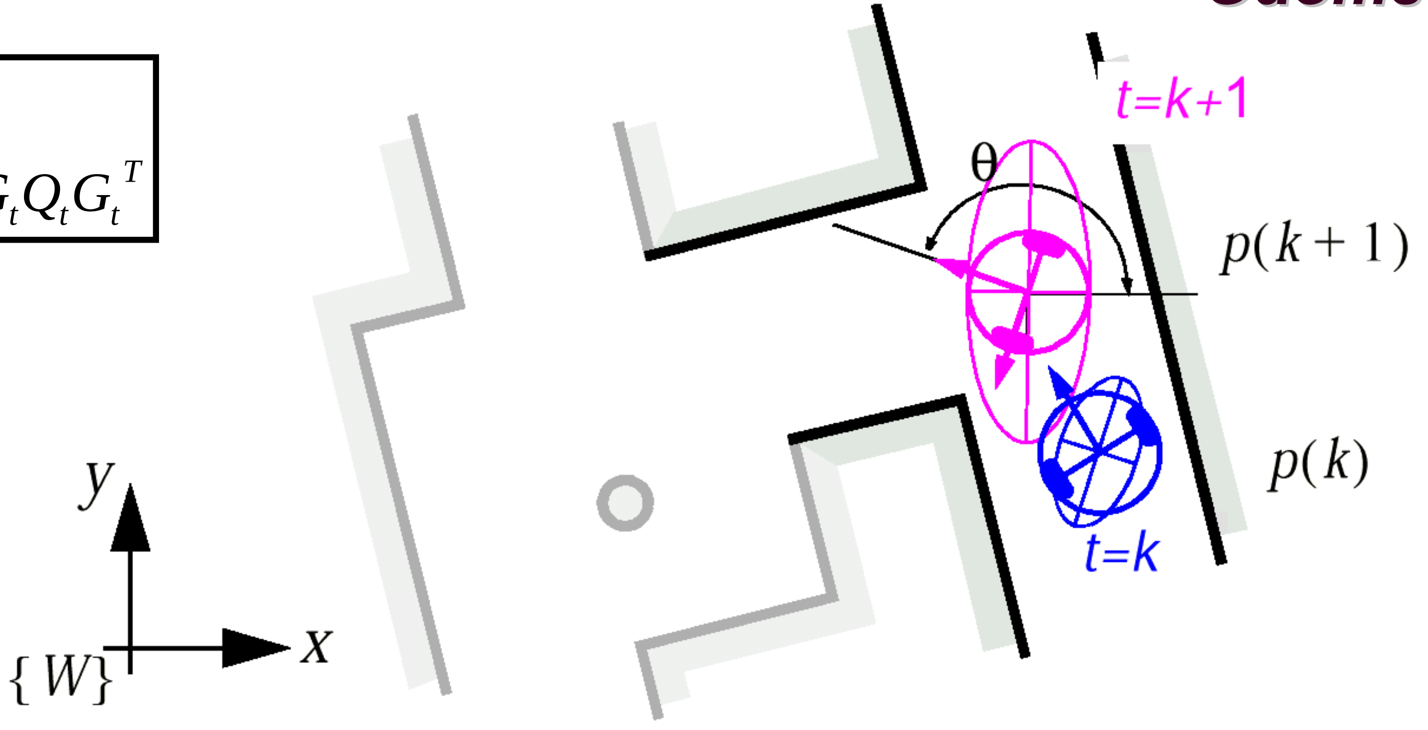
Kalman Filter for Mobile Robot Localization

1. Robot Position Prediction

- In a first step, the robot's position at time step *t*+1 is predicted based on its old location (time step *t*) and its movement due to the control input u_t:

$$\boxed{\begin{aligned}\hat{x}_{t+1/t} &= F_t \hat{x}_{t/t} + B_t u_t \\ P_{t+1/t} &= F_t P_{t/t} F_t^T + G_t Q_t G_t^T\end{aligned}}$$

Odometry



Kalman Filter for Mobile Robot Localization

2. Observation

- The second step is to obtain the observation Z_{t+1} (measurements) from the robot's sensors at the new location at time $t+1$

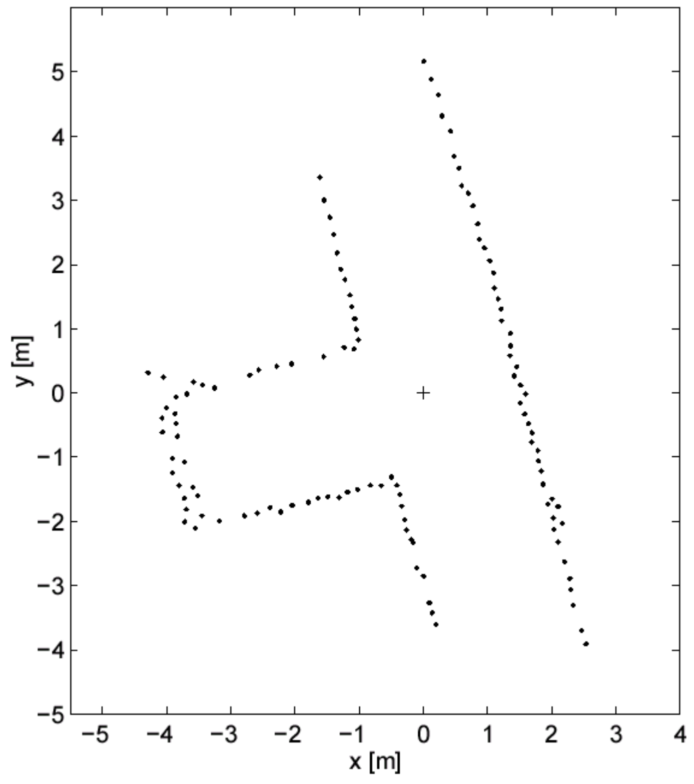
- The observation usually consists of a set of single observations extracted from the different sensor's signals. It can represent *raw data scans* as well as *features* like *lines*, *doors* or *any kind of landmarks*.

- The parameters of the targets are usually observed in the sensor frame {S}.
  - *Therefore the observations have to be transformed to the world frame {W} or*
  - *The measurement prediction has to be transformed to the sensor frame {S}.*
  - *This transformation is specified in the update equation*

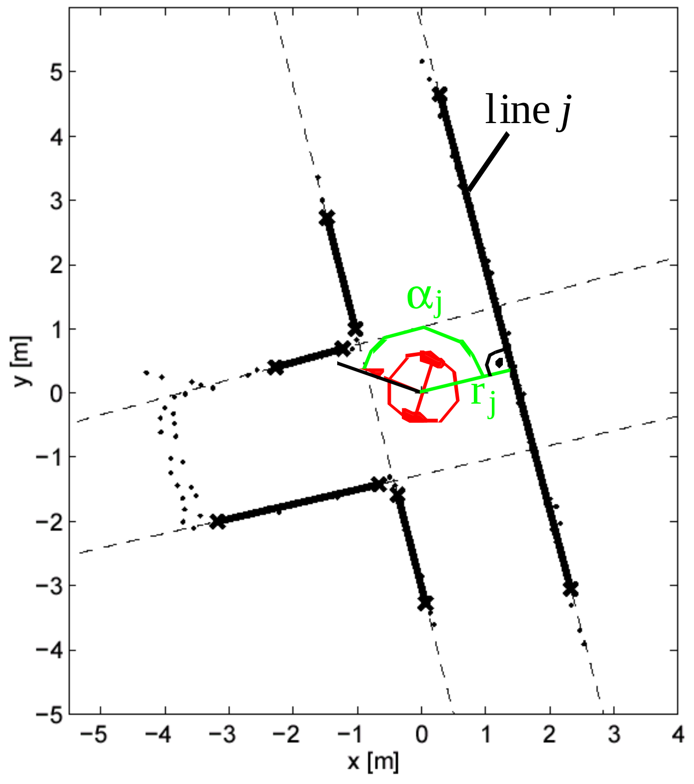
Kalman Filter for Mobile Robot Localization

2. Observation: *Example*

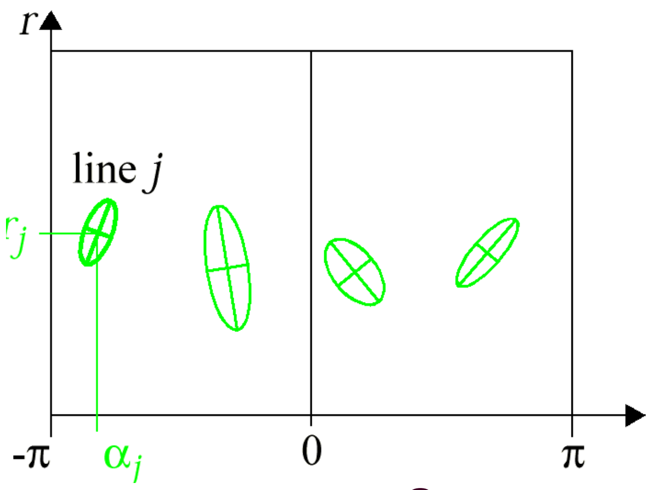
Raw Data of Laser Scanner



Extracted Lines



Extracted Lines in Model Space



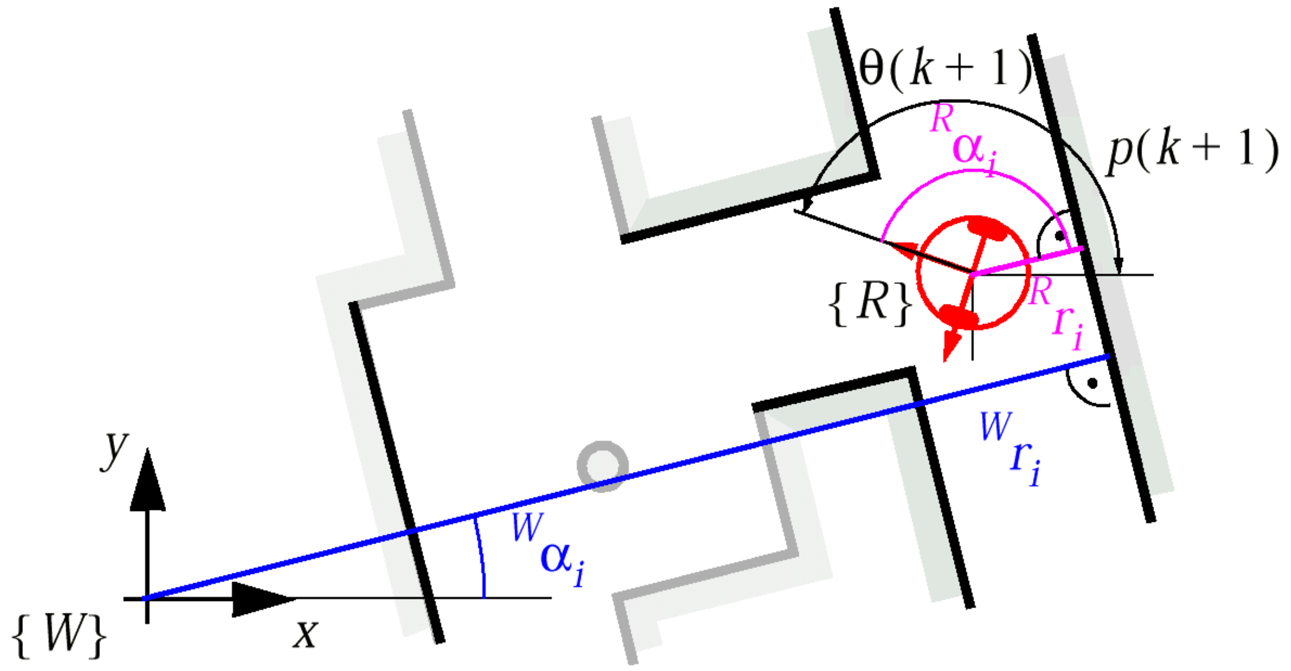
Sensor (robot) frame

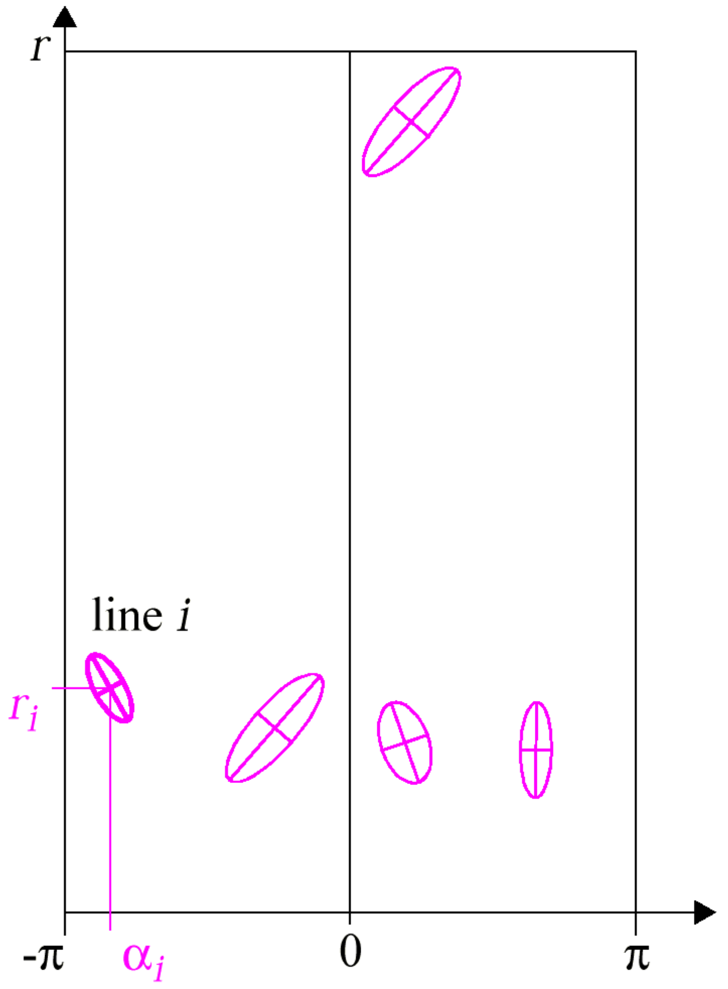
Kalman Filter for Mobile Robot Localization

3. Measurement Prediction

- In the next step we use the predicted robot position and the map to generate the predicted observations, which are transformed into the sensor frame

$$\boxed{\hat{z}_{t+1} = H_{t+1}\hat{x}_{t+1/t}}$$



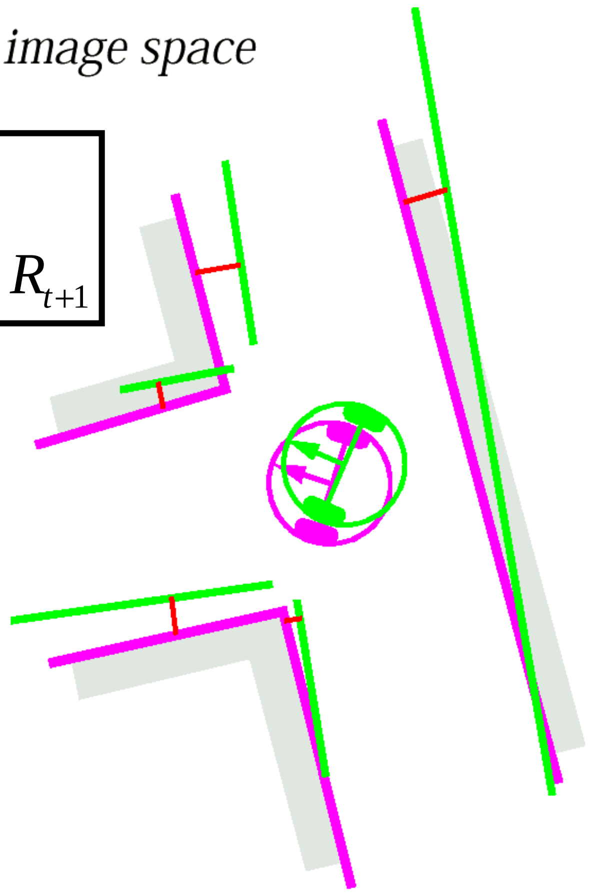


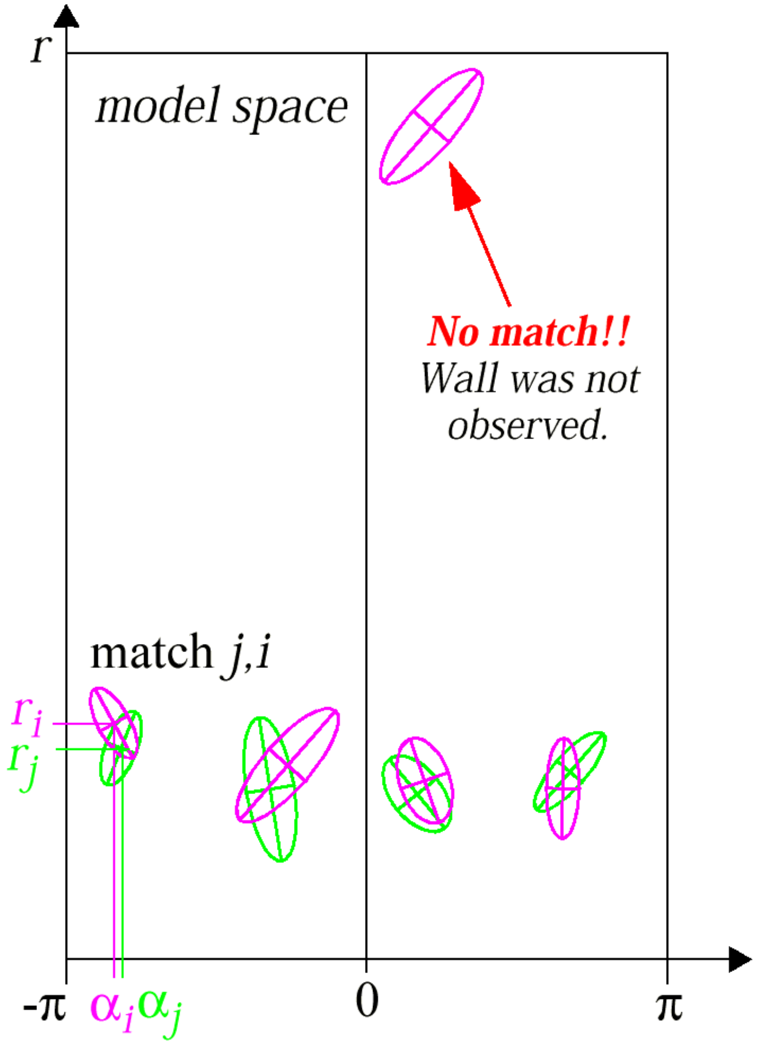
Kalman Filter for Mobile Robot Localization

4. Matching: *Example*

$$r_{t+1} = z_{t+1} - \hat{z}_{t+1}$$

$$S_{t+1} = H_{t+1} P_{t+1/t} H_{t+1}^{T} + R_{t+1}$$





To find correspondence between predicted and observed features, use a distance metric (such as Mahalanobis distance)

Kalman Filter for Mobile Robot Localization

5. Estimation: Applying the Kalman Filter

- Calculate Kalman filter gain:

$$K_{t+1} = P_{t+1/t} H_{t+1}{}^{T} S_{t+1}{}^{-1}$$

- Update robot's position estimate:

$$\hat{x}_{t+1/t+1} = \hat{x}_{t+1/t} + K_{t+1} r_{t+1}$$

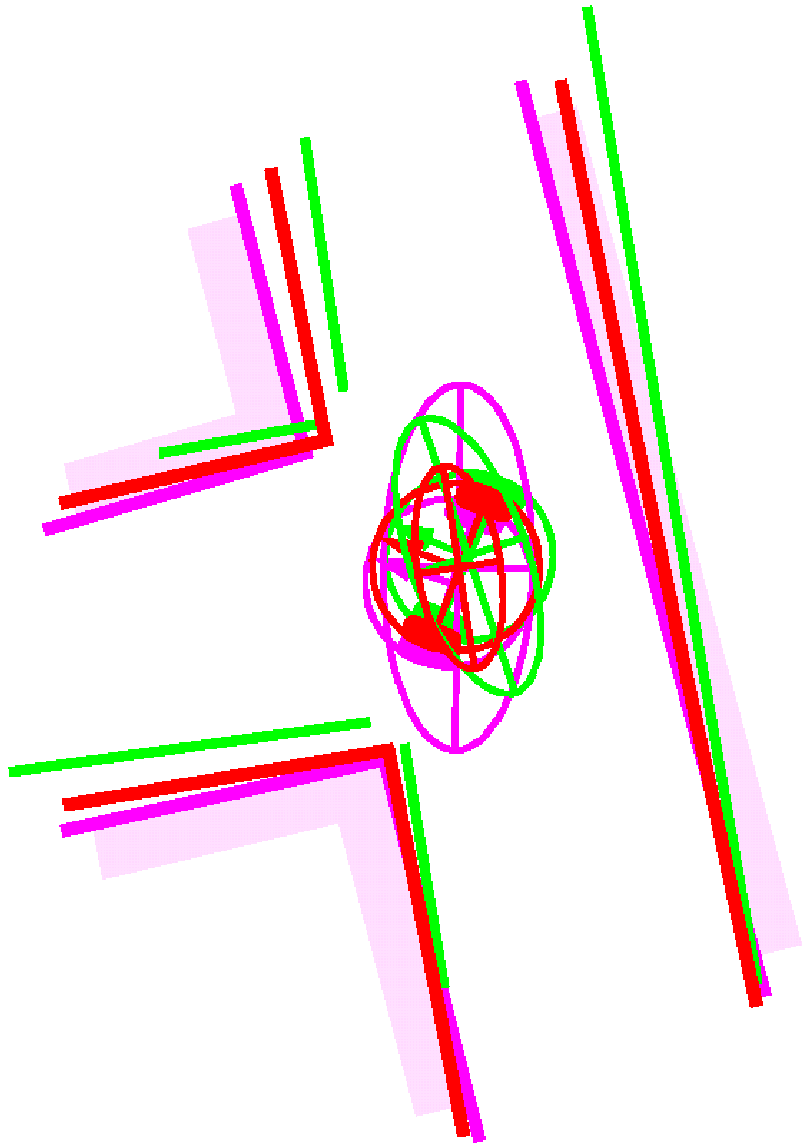
- Calculate the associated variance

$$P_{t+1/t+1} = P_{t+1/t} - P_{t+1/t} H_{t+1}{}^{T} S_{t+1}{}^{-1} H_{t+1} P_{t+1/t}$$

Kalman Filter for Mobile Robot Localization

Estimation: *Example*

- Kalman filter estimation of the new robot position:
  - *By fusing the prediction of robot position (magenta) with the innovation gained by the measurements (green) we get the updated estimate of the robot position (red)*



Example 1: Simple 1D Linear System

Given: F=G=H=1, u=0
Initial state estimate = 0
Linear system:

$$x_{t+1} = x_t + w_t$$

$$z_{t+1} = x_{t+1} + n_{t+1}$$

Unknown noise parameters

Propagation:

$$\hat{x}_{t+1/t} = \hat{x}_{t/t}$$

$$P_{t+1/t} = P_{t/t} + Q_t$$

Update:

$$\hat{z}_{t+1} = \hat{x}_{t+1/t}$$

$$r_{t+1} = z_{t+1} - \hat{x}_{t+1/t}$$

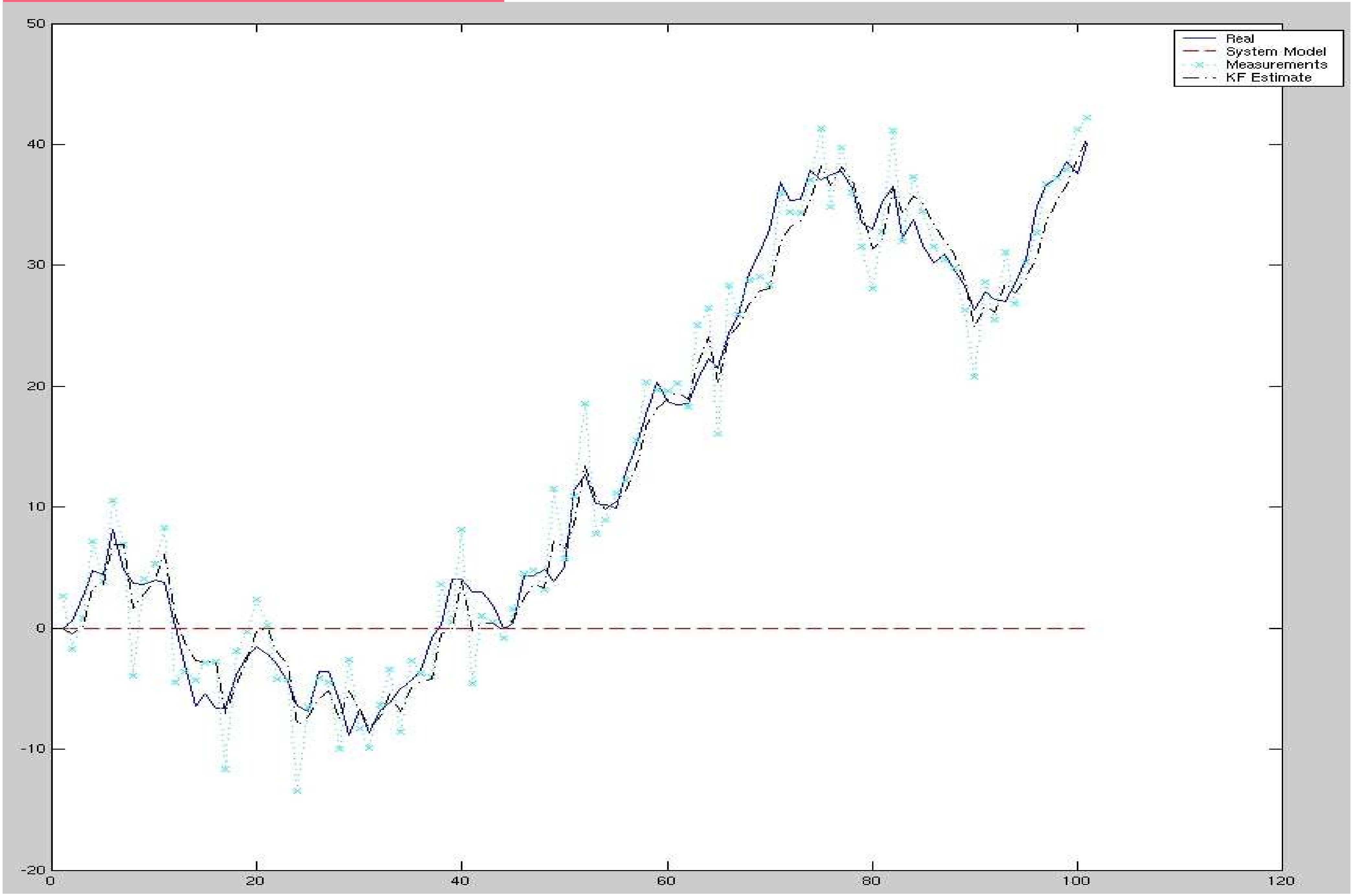
$$S_{t+1} = P_{t+1/t} + R_{t+1}$$

$$K_{t+1} = P_{t+1/t} S_{t+1}^{-1}$$

$$\hat{x}_{t+1/t+1} = \hat{x}_{t+1/t} + K_{t+1} r_{t+1}$$

$$P_{t+1/t+1} = P_{t+1} - P_{t+1/t} S_{t+1}^{-1} P_{t+1/t}$$

State Estimate



Autonomous Map Building

Starting from an arbitrary initial point,

a mobile robot should be able to autonomously explore the environment with its on board sensors,

gain knowledge about it,

interpret the scene,

build an appropriate map

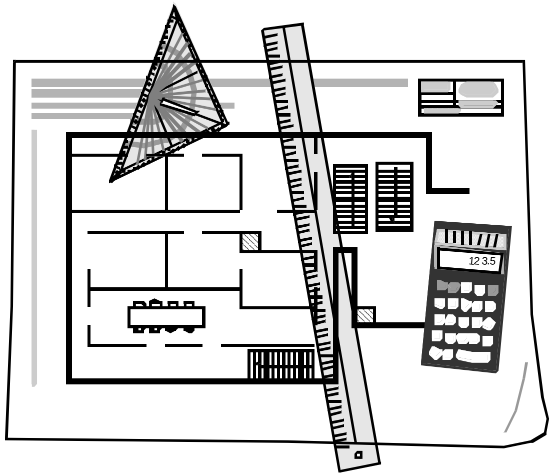
and localize itself relative to this map.

SLAM

The Simultaneous Localization and Mapping Problem

Map Building:
How to Establish a Map

1. By Hand



2. Automatically: **Map Building**

The robot **learns** its environment

Motivation:

- by hand: hard and costly
- dynamically changing environment
- different look due to different perception

3. Basic Requirements of a Map:

- *a way to incorporate newly sensed information into the existing world model*
- *information and procedures for estimating the robot's position*
- *information to do path planning and other navigation task (e.g. obstacle avoidance)*

- Measure of Quality of a map
  - *topological correctness*
  - *metrical correctness*

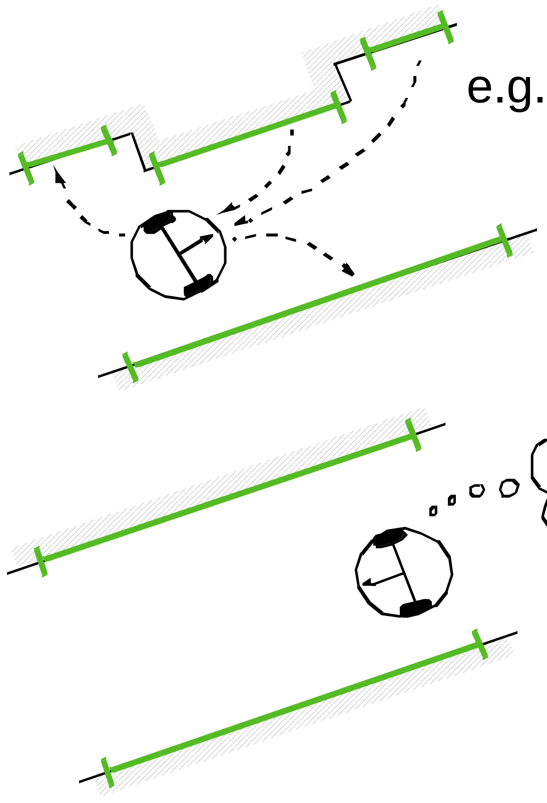
  $\Rightarrow$ *predictability*

- But: Most environments are a mixture of predictable and unpredictable features
  $\rightarrow$ hybrid approach

  model-based vs. behavior-based

Map Building:
The Problems

1. Map Maintaining: Keeping track of changes in the environment

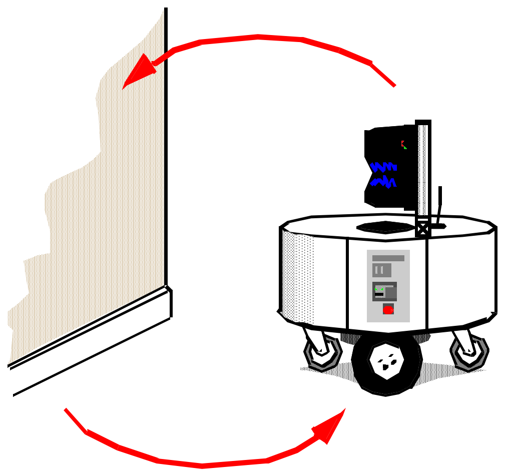


e.g. disappearing cupboard

- e.g. measure of belief of each environment feature

2. Representation and Reduction of Uncertainty

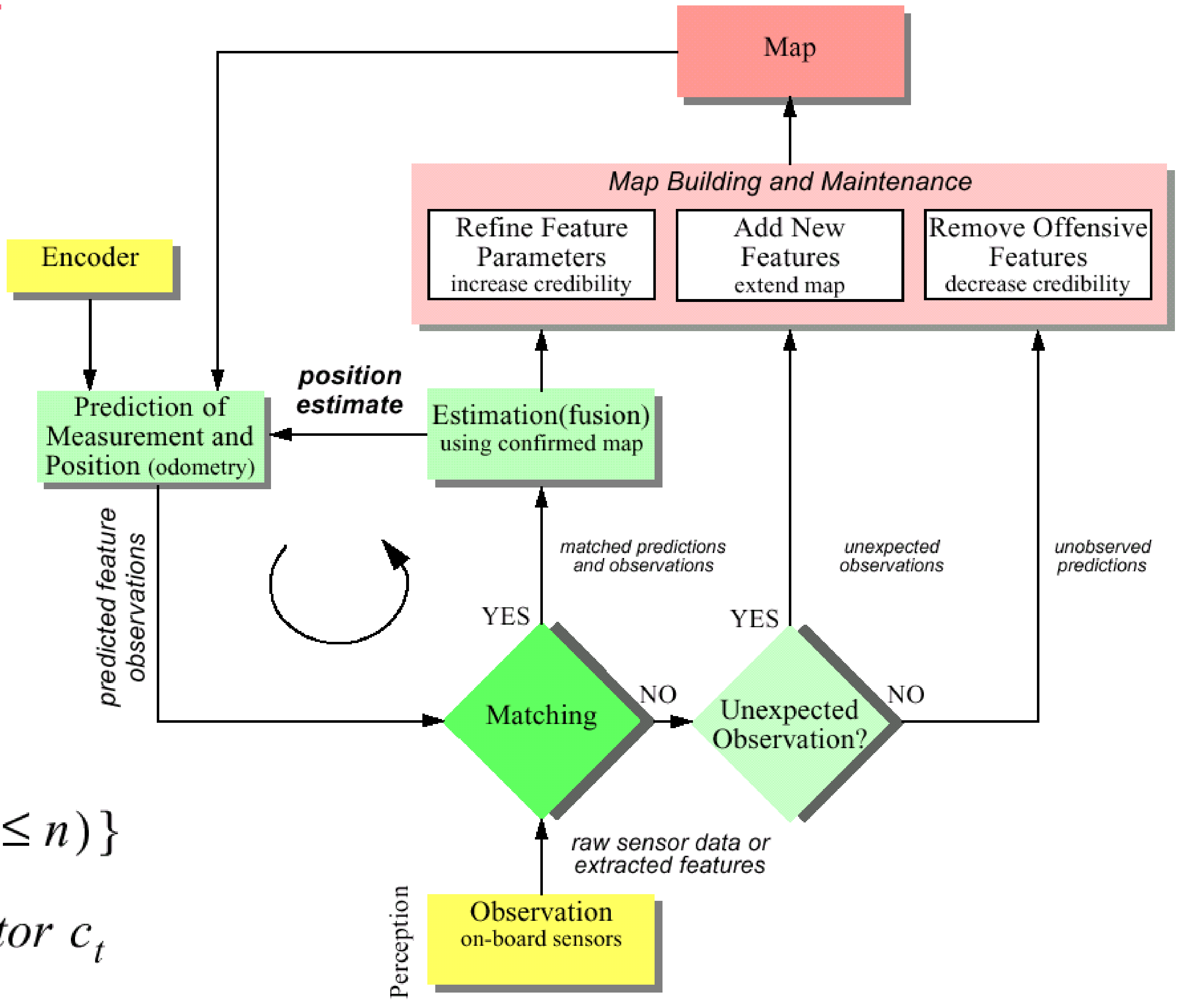
position of robot -> position of wall



position of wall -> position of robot

- probability densities for feature positions
- additional exploration strategies

General Map Building Schematics



$$M = \{\hat{z}_t, \Sigma_t, c_t | (1 \le t \le n)\}$$

credibility factor c_t

Map Representation

- M is a set n of probabilistic feature locations
- Each feature is represented by the covariance matrix Σ_t and an associated credibility factor c_t

$$M = \{\hat{z}_t, \Sigma_t, c_t | (1 \le t \le n)\}$$

- c_t is between 0 and 1 and quantifies the belief in the existence of the feature in the environment

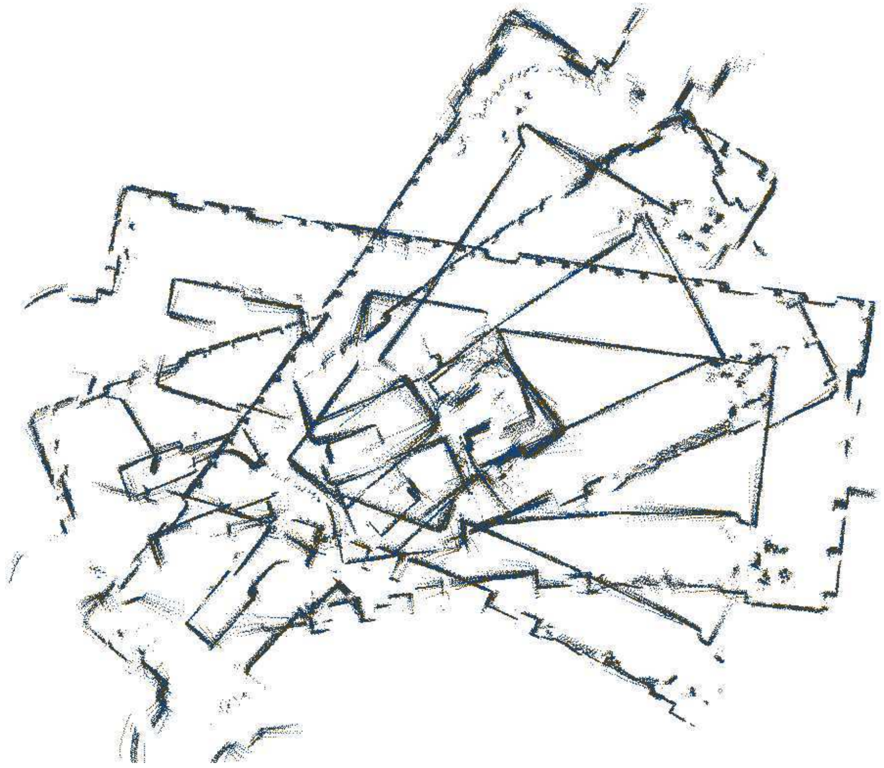
$$c_t(k) = 1 - e^{-\left(\frac{n_s}{a} - \frac{n_u}{b}\right)}$$

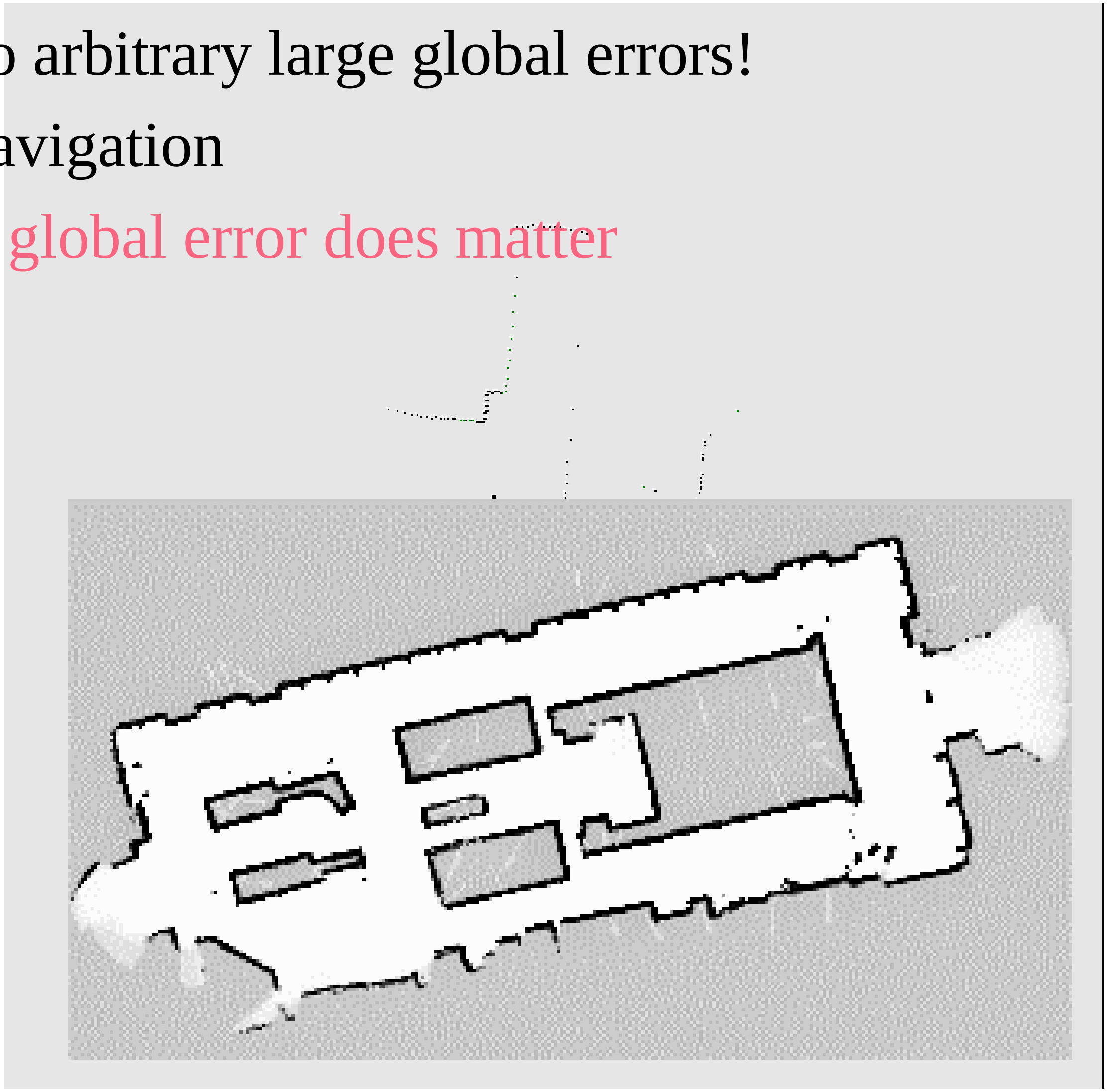
- a and b define the learning and forgetting rate and n_s and n_u are the number of matched and unobserved predictions up to time k, respectively.

Cyclic Environments

Courtesy of Sebastian Thrun

- Small local error accumulate to arbitrary large global errors!
- This is usually irrelevant for navigation
- However, when closing loops, global error does matter

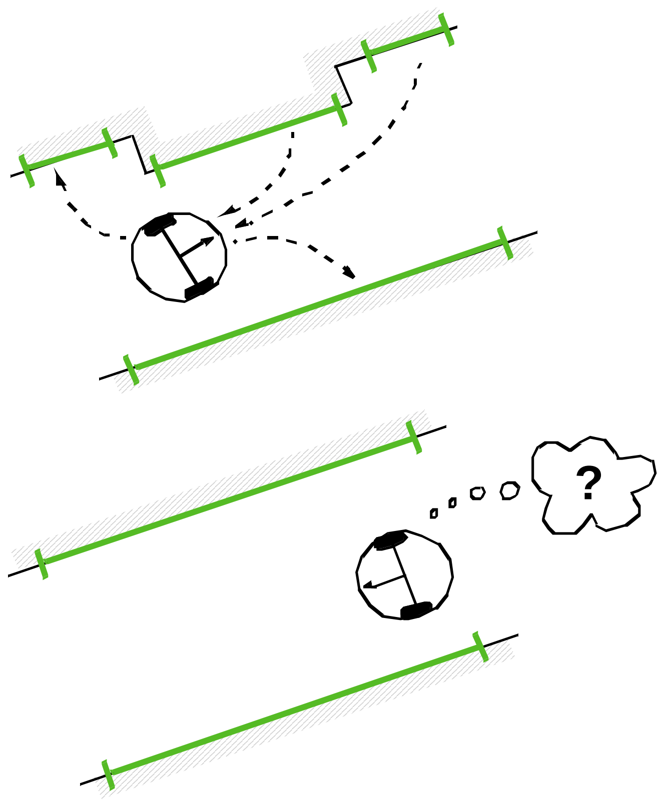




Dynamic Environments

- Dynamical changes require continuous mapping

- If extraction of high-level features would be possible, the mapping in dynamic environments would become significantly more straightforward.
  - *e.g. difference between human and wall*
  - *Environment modeling is a key factor for robustness*



Case Study: A Different Approach to Multi-Robot Localization

- Work at University of Southern California
- SLAM (Simultaneous Localization and Mapping)
  - *Relaxation on a mesh*
  - *Maximum likelihood estimation*
- Scenario modeled with:
  - *The Stage simulator*
- Multi-operator, multi-robot tasking

Localization

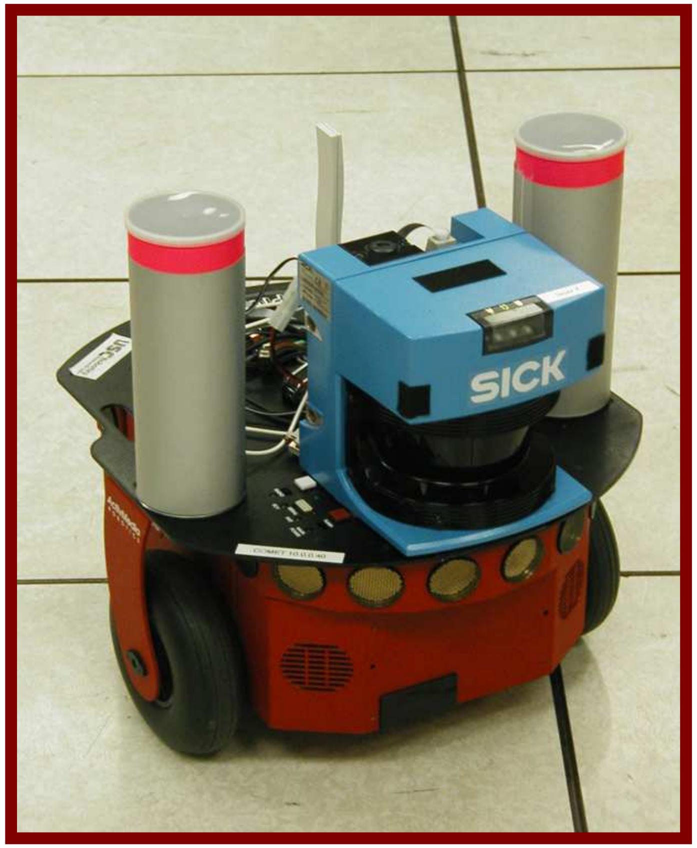
- Past approaches include:
  - *Filtering inertial sensors for location estimation*
  - *Using landmarks (based on vision, laser etc.)*
  - *Using maps*

- Algorithms vary from Kalman filters, to Markov localization to Particle filters

- This case study approach
  - *Exploit communication for in-network localization*
  - *Physics-based models*

Static Localization

- System contains beacons and beacon detectors
- Assumptions:
  - *beacons are unique,*
  - *beacon detectors determine correct identity.*
- Static localization:
  - *determine the* relative *pose of each pair of beacons/detectors*



Mesh Definition: Damped spring mass system

A set of nodes:

$$\begin{aligned} N &= \{n_1, n_2, ..., n_i, ...\} \\ n_i &= (x_i, m_i) \\ x_i &\equiv \text{pose of node } i \\ m_i &\equiv \text{mass of node } i \end{aligned} \tag{1}$$

A set of links:

$$\begin{aligned} L &= \{l_1, l_2, ..., l_j, ...\} \\ l_j &= (a_j, b_j, z_j, k_j) \\ a_j &\equiv \text{node index} \\ b_j &\equiv \text{node index} \\ z_j &\equiv \text{spring extent} \\ k_j &\equiv \text{spring constant} \end{aligned} \tag{2}$$

Mesh Energy

Kinetic energy

$$V_i = \frac{1}{2} m_i \dot{x}_i^2$$

$$V = \sum_i V_i$$

Potential energy

$$U_j = \frac{1}{2} k_j (\bar{z}_j - z_j)^2$$

$$\bar{z}_j = \Gamma(x_{a_i}, x_{b_j})$$

$$(\bar{z}_j = \left| x_{b_j} - x_{a_j} \right|)$$

$$U = \sum_j U_j$$

Mesh Forces and Equations of Motion

Forces

$$F_i = -\nabla_{x_i} U = -\sum_j \frac{\partial \bar{z}_j}{\partial x_i} \frac{\partial U_j}{\partial \bar{z}_j}$$

Equations of motion

$$0 = \ddot{x}_i + \nu \dot{x}_i - F_i / m_i$$

$$As \quad t \to \infty \quad V \to 0, U \to U_{\min}$$

Encoding

Each node n_i represents a beacon or detector *at some time* t such that:

$$\begin{aligned} x_i &\longleftarrow \text{pose of beacon/detector at time } t \\ m_i &\longleftarrow 1 \end{aligned}$$

Each link l_j represents a *beacon* measurement such that:

$$\begin{aligned} z_j &\longleftarrow \text{relative pose at time } t \\ k_j &\longleftarrow \text{measurement uncertainty} \end{aligned}$$

or a *motion* measurement such that:

$$\begin{aligned} z_j &\longleftarrow \text{change in pose between time } t \text{ and } t + \Delta t \\ k_j &\longleftarrow \text{measurement uncertainty} \end{aligned}$$

SLAM: Simultaneous Localization and Mapping

```
nodes 7 links 1
energy 0.0000 steps 210290
```

Multi-robot SLAM

Team Localization using MLE

- Construct a set of estimates *H* = {*h*} where
  h *is the pose of robot* r *at time* t.

- Construct a set of observations *O* = {*o*} where *o* is either:
  the measured pose of robot r_b *relative to robot* r_a *at time* t, *or*
  the measured change in pose of robot r *between times* t_a *and* t_b.

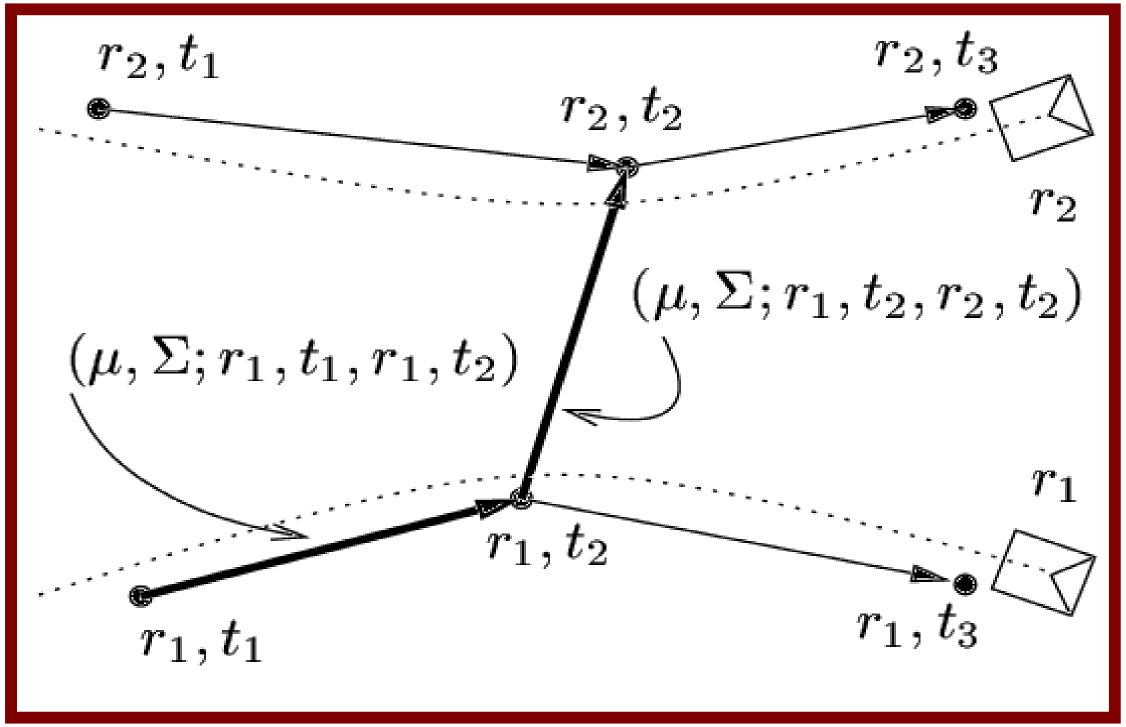
- Assuming statistical independence between observations find the set of estimates *H* that maximizes:

$$P(O \mid H) = \prod_{o \in O} P(o \mid H)$$

Approach

Equivalently, find the set *H* that minimizes:

$$U(O \mid H) = \sum_{o \in O} U(o \mid H)$$



r_2, t_1
r_2, t_2
r_2, t_3
r_2
$(\mu, \Sigma; r_1, t_2, r_2, t_2)$
$(\mu, \Sigma; r_1, t_1, r_1, t_2)$
r_1
r_1, t_2
r_1, t_1
r_1, t_3

Gradient-based Estimation

- Each estimate

$$h = (\hat{q}, r, t)$$

- Each observation

$$o = (\mu, \Sigma, r_a, t_a, r_b, t_b)$$

- Measurement uncertainty assumed normal

$$U(o \mid H) = \frac{1}{2}(\mu - \hat{\mu})^T \Sigma (\mu - \hat{\mu})$$

- Relative →Absolute

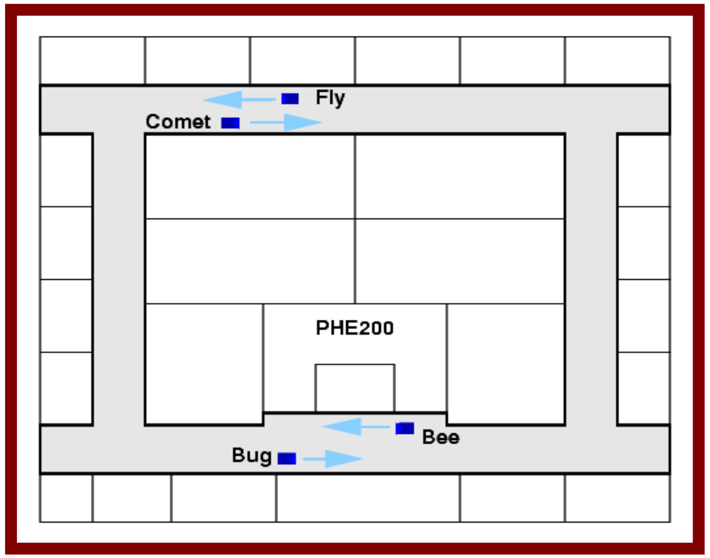
$$\hat{\mu} = \Gamma(\hat{q}_a, \hat{q}_b)$$

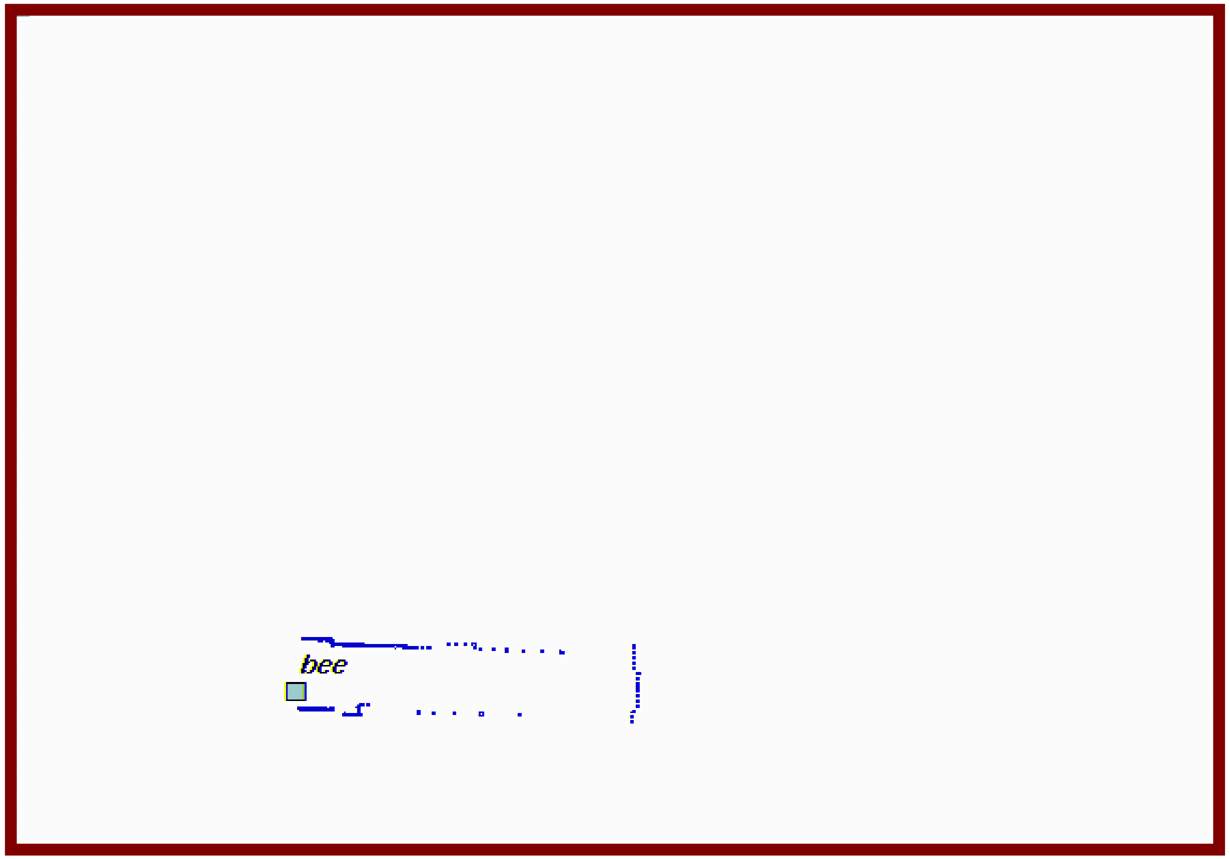
Gradient Descent

$$\frac{\partial}{\partial h} U(O|H) = \sum_{o \in O} \frac{\partial \hat{\mu}}{\partial h} \frac{\partial}{\partial \hat{\mu}} U(o|H)$$

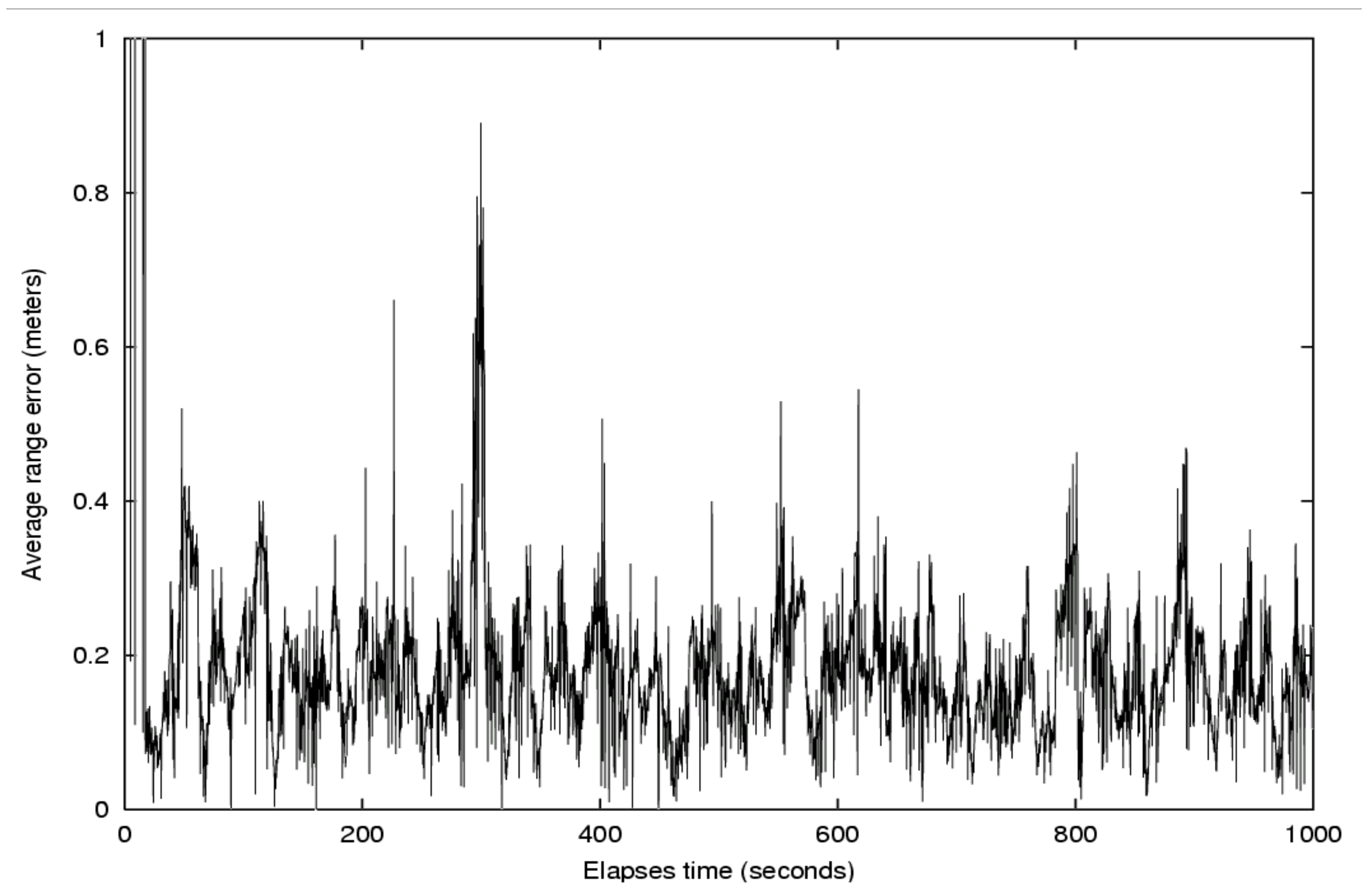
- Compute set of poses q that minimizes U(O|H)
- Gradient-based algorithm

Results (large environment)

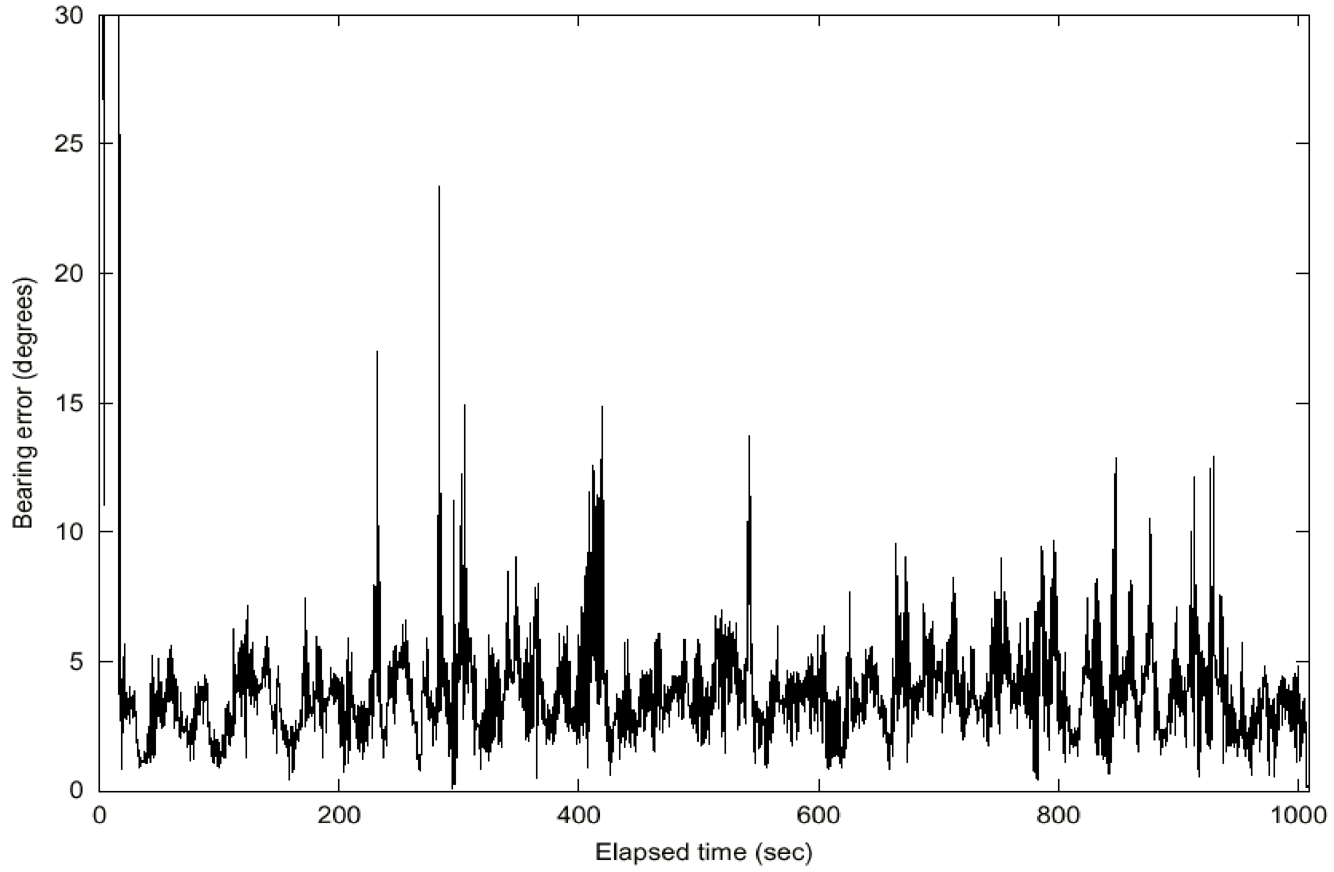




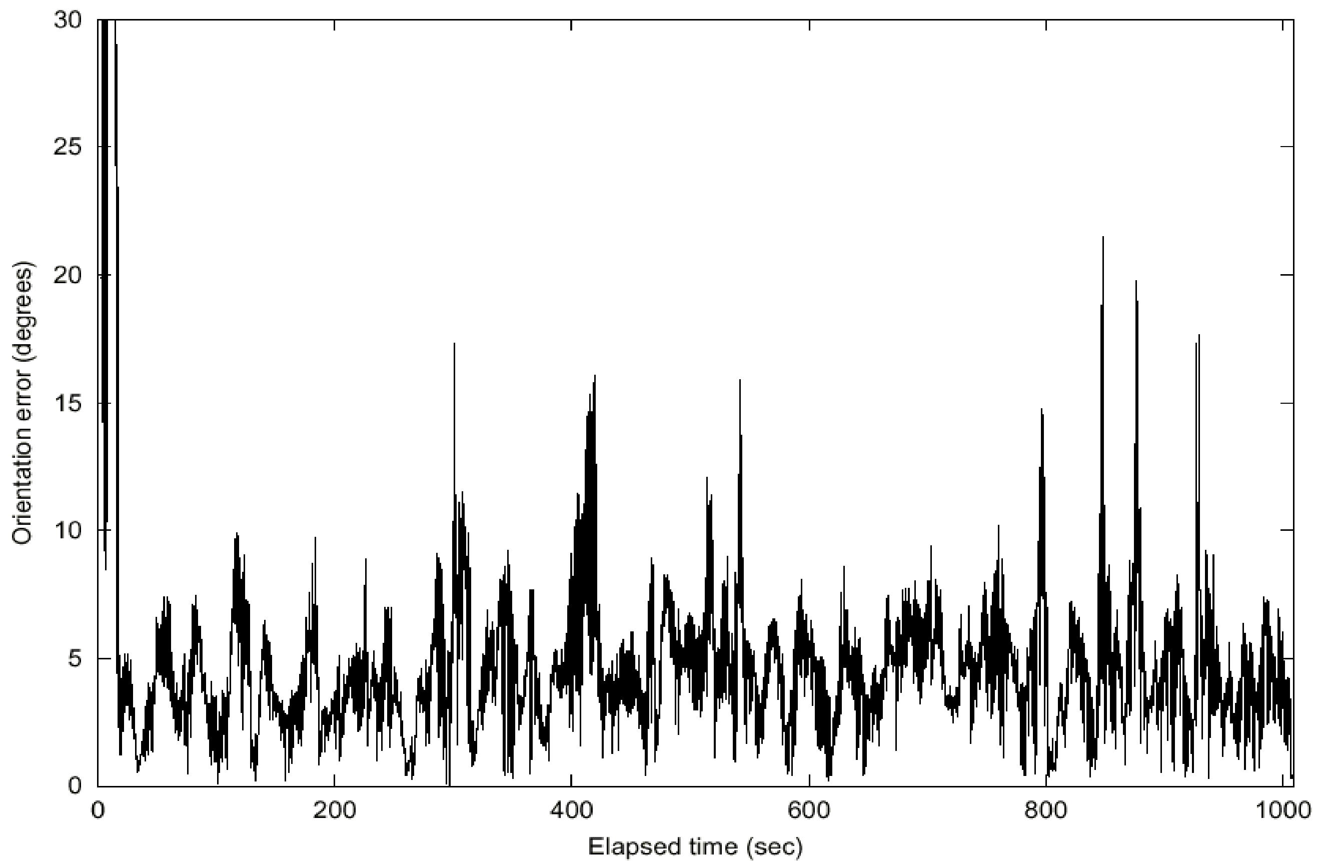
Range Error vs. Time



Bearing Error vs. Time



Orientation Error vs. Time



Summary of USC Team Localization Approach

- Runs on any platform as long as it can compute its motion via inertial sensing
- Unique beacons: robots, people, fixed locations etc.
- No model of the environment
- Indifferent to changes in the environment
- Robust to sensor noise
- Permits both centralized and distributed implementation