Fundamenta Informaticae 85 (2008) 1–16
IOS Press

Memory-Based State-Estimation

Matthias Jüngel* **and Heinrich Mellmann**
Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 , Berlin, Germany
mail@matthias-juengel.de; mellmann@informatik.hu-berlin.de

Abstract. In this paper we introduce a state-estimation method that uses a short-term memory to calculate the current state. A common way to solve state estimation problems is to use implementations of the Bayesian algorithm like Kalman filters or particle filters. When implementing a Bayesian filter several problems can arise. One difficulty is to obtain error models for the sensors and for the state transitions. The other difficulty is to find an adequate compromise between the accuracy of the belief probability distribution and the computational costs that are needed to update it. In this paper we show how a short-term memory of perceptions and actions can be used to calculate the state. In contrast to the Bayesian filter, this method does not need an internal representation of the state which is updated by the sensor and motion information. It is shown that this is especially useful when information of *sparse sensors* (sensors with non-unique measurements with respect of the state) has to be integrated.

Keywords: State Estimation, Localization

1. Introduction

In this section we introduce what distinguishes our memory-based approach of state estimation (*MBSE*) from the well known approach based on Hidden Markov Models (*HMMs*).

The purpose of state estimation is to find the best approximation to a state of a system. Usually this state can not be observed directly. Assume that the state we want to determine is the position of a robot. When there is a perfect position sensor available, there is nothing left to be done for state estimation.

*Address for correspondence: LFG Künstliche Intelligenz, Institut für Informatik, HU-Berlin, Unter den Linden 6, 10099 Berlin, Germany

More generally there is no need for a sophisticated state estimation technique when there are sensors which provide at least as much information as is needed to determine the position in every time step. [2] Formally such *Direct State Estimation* can be seen as a function f_D which calculates the current state $\vec{x_t}$ at time t based on the vector $\vec{z_t}$ of observations at time t:

$$\vec{x_t} = f_D(\vec{z_t})$$

This direct form of state estimation does not use the knowledge about the actions u_t which led the system from state x_{t-1} to state x_t. Additionally the knowledge about probabilistic properties of the sensors can not be used (unless there is a really large number of sensors measuring simultaneously). Furthermore *Direct State Estimation* is not applicable when there are less sensors than needed to determine the state directly.

These drawbacks are overcome by the Hidden Markov Model approach of state estimation [10, 11]. In this approach the current state is represented as a probability function $bel(x_t)$ which is called the *belief* of x_t. The current state is calculated by a function f_{HMM} which is usually realized by a variant of a Bayesian filter using the belief of the last state $bel(\vec{x_{t-1}})$, the current measurement $\vec{z_t}$, and the action $\vec{u_t}$ executed at the transition from $\vec{x_{t-1}}$ to $\vec{x_t}$:

$$\vec{x_t} = f_{HMM}(bel(\vec{x_{t-1}}), \vec{z_t}, \vec{u_t})$$

Bayesian filters can use probabilistic sensor and motion models and accumulate information over time. Following the Markov Assumption this aggregation of information can be done from state to state as long as the current state contains enough information. Therefore it is often referred as recursive state estimation.

If we give up the assumption to have a representation of a complete state in each time step, we can define a function f_M which calculates the current state x_t based on the current and all past observations $\vec{z_{0:t}}$ and all past control actions $\vec{u_{0:t}}$:

$$\vec{x_t} = f_M(\vec{z_{0:t}}, \vec{u_{0:t}})$$

Such variants of state estimation we will call *Memory Based State Estimation (MBSE)* [4]. Strictly speaking *Direct State Estimation* is a variant of MBSE which only uses the current observation. *Hidden Markov Model* based state estimation can also be seen to be a variant of *Memory Based State Estimation*. In this case the function f_M has to calculate all recursion steps of function f_{HMM} in each time step (which would break the idea of recursion). There are already some memory-based extensions for the classical localization methods [6, 9, 1, 8]. In this paper we want to change the paradigm towards using a memory-based approach to solve the state estimation problem for scenarios with sparse senor readings.

While the concept of Memory Based State Estimation introduced above is nothing more than a prototype, in this article we will give a more detailed template for *MBSE* which can be used to solve a wide class of state estimation problems.

Section 2 gives our motivation to do memory based state estimation, section 3 introduces the formalisms we need to describe and argue about our system, section 4 describes the method itself, in section 5 we present some algorithmic implementations, and in section 6 we show the results of the experiments we conducted.

2. Motivation

To explain our motivation to introduce *Memory Based State Estimation* we will have a look at an example from the world of sailing. In any book about sailing, e.g. [7] you will find a description on how to calculate a *running fix*. To do so you need a compass, a way to measure the distance you travel, and two bearing measurements to a single landmark with known position. Then the calculation of the current position (the position where the second observation was made) follows simple geometric considerations.

Let's consider the three concepts of state estimation introduced in the previous section: The calculation of a running fix is not possible via *Direct State Estimation* as information has to be accumulated over time. The calculation from the sailing book using simple geometry would fall into the class of *Memory Based State Estimation*. Using a HMM-based approach is also possible. To do so we have to determine two things: First a probabilistic sensor model which gives the expected compass and bearing measurement for each possible position. And second a probabilistic motion model which describes the probability to move from one position to another, given a certain action (where the action can be derived from the measurement of moved distance). The rest is done automatically by the variant of the Bayesian Filter we choose.

While the geometry-based approach is simple and straight forward, the HMM-based approach is more general. For a lot of state estimation problems it is much easier to provide a sensor and a motion model than to dive into geometric considerations which starts getting complicated when contradictory information has to be processed. The HMM-approach can cope with that without additional efforts. On the other hand the HMM-approach needs a complete representation of the current state. This has to be a probability density function over the state space in order to be able to accumulate information over time. While a complete discrete representation leads to high memory usage and high computational costs, there are several solutions to reduce this complexity. Particle filters approximate the function using a set of particles, Kalman Filters assume the function to be a multivariate normal distribution. However these approximations have their weaknesses. For example a very high number of particles is needed when the information that has to be integrated is sparse (does not provide a lot of information at once).

In this article we present a template for *Memory Based State Estimation* that combines advantages from the HMM-approach and from geometric considerations. As in the HMM-approach, the sensor and the motion model are the only information needed about the environment. Like in the geometry-based approach no internal representation of the state is needed to calculate the state, this is done in one step considering all data at once.

3. Definitions

In this section we introduce some vocabulary needed in the rest of the article.

3.1. Sensor and Motion Models

Definition 3.1. *(Observation function)*
An observation function f_s gives the expected observation z for each state x:

$$z = f_s(x)$$

Definition 3.2. *(Control function)*
A control function f_c describes the new state x_i when the previous state was x_{i-1} and the action u_i was performed:

$$x_i = f_c(x_{i-1}, u_i)$$

Note that with these functions no probabilistic properties of the senors can be modeled. In the same way only a deterministic effect of the control actions can be described.

Definition 3.3. *(Inverse control function)*
We require that the control function is invertible in the first argument such that we can define the inverse of a control function $f_c^{-1}(x, u)$ with

$$f_c(f_c^{-1}(x, u), u) = x \text{ and}$$
$$f_c^{-1}(f_c(x, u), u) = x$$

Of course in any real-world scenario there is noise in the motion (which makes the motion model non-invertible), as well there is noise in the sensor readings. However, below we will define an algorithm which does not consider noise in the motion model and does not consider systematic errors in the sensor readings. In section 6 we show that the algorithm nevertheless can cope with large errors in the motion (small kidnappings) and significant error in the sensor readings.

Definition 3.4. *(Concatenated Controls)* The concatenation of two controls $u_i u_j$ is defined such that

$$f_c(x, u_i u_j) = f_c(f_c(x, u_i), u_j)$$

Note that concatenation usually is not commutative.

Proposition 3.1. $f_c^{-1}(x, u_i u_j) = f_c^{-1}(f_c^{-1}(x, u_j), u_i)$

Definition 3.5. *(Accumulated Controls)*
For the reason of simplification we introduce the symbol v_m to describe the concatenation of the last m control actions:

$$v_m := u_{n-m+1}, ..., u_n$$

With this definition and proposition 3.1 it is easier to describe a sequence of previous states, given a current state x_n and a sequence of controls $u_1, ..., u_n$ that led to this state:

$$x_{n-m} = f_c^{-1}(x_n, v_m)$$

Figure 1 visualizes the connection between u_1, ..., u_n and v_1, ..., v_n.

3.2. Hidden Markov Models

As mentioned in the introduction, Hidden Markov Models are the basis for many state estimation techniques that are based on Bayesian filtering. Figure 1 shows such a model.

An important prerequisite are the probabilistic sensor model and the probabilistic motion model:

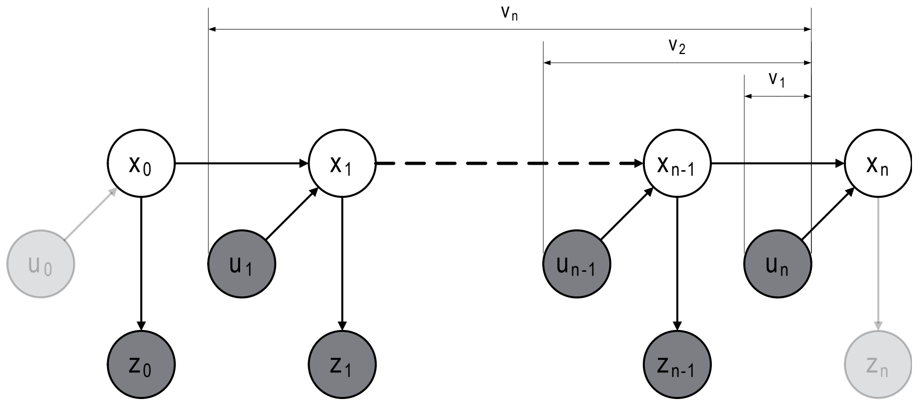

Figure 1. Hidden Markov Model. The states $x_0, x_1, ..., x_n$ are not directly visible. A state x_i depends only on state x_{i-1} and the control action u_i. An observation z_i depends only on the state x_i. The nodes u_0 and z_n are shaded as in our considerations we always start with an observation (z_0) and end with a control (u_n). Additionally the connection between the controls u_1, ..., u_n and the accumulated controls $v_1, ..., v_n$ is visualized (v_m is the sum of the last m controls, which is the sum of all controls since u_{n-m+1}, cf. Definition 3.5).

Definition 3.6. *(Sensor model)*
A sensor model is a probability density function $p(z|x)$ which describes a sensor by predicting the probability to measure z, given that the current state is x.

Definition 3.7. *(Motion model)*
A motion model describes the probability $p(x_i|x_{i-1}, u)$ for the system to change from state x_{i-1} to state x_i, given the control u.

With these definitions the recursive algorithm for Bayesian filtering can be given (The version we show here can be found in [10]):

Algorithm 1: Discrete_Bayes_filter($p(X_{t-1} = x_k), u_t, z_t$)

```
foreach $k$ do
    $\bar{p}(X_t = x_k) = \sum_i p(X_t = x_k|X_{t-1} = x_i, u_t) \cdot p(X_{t-1} = x_i)$
    $p(X_t = x_k) = \eta \cdot p(z_t|X_t = x_k) \cdot \bar{p}(X_t = x_k)$
return $p(X_t = x_k)$
```

This algorithm is shown here as we later want to compare our *MBSE*-variant to this HMM-approach.

4. Memory-Based State Estimation (MBSE)

This section describes the main principles and the basic math of our state estimation approach.

4.1. Organization of Memory

There are two types of information available. The sequence of sensor data and the sequence of control data. These two types of information are organized in a memory. This memory is a matrix M with two columns and n rows. The first column is a vector $\vec{z} = (z_0, z_1, ..., z_n)$ that contains the sequence of observations, the most current observation is z_n. The second column is a vector $\vec{u} = (u_0, u_2, ..., u_n)$ that stores the according control data.

Thus each row is a tuple (z_i, u_i) that contains an observation and the control data at the time of the observation. The number of rows in the matrix increases with every observation. To be able to process more than one observation at the same time we allow a control action to be of type *do-nothing*.

4.2. State Estimation Using Least Squares

The whole process of memory based state estimation is based on the memory described above. In this section we show what kind of information a single measurement delivers, what is known from a single observation in the past, and what is known from all observations in the past.

4.2.1. Information Gain of a Single Measurement

Usually a single measurement does not provide enough information to determine the state. However, a measurement z constrains the set of possible states according to the observation function defined in 3.1:

$$X_z := (x \in X : f_s(x) - z = 0)$$

We can also define a larger set based on an expected maximum error of t in the measurement:

$$X_{(z),(t)} := (x \in X : (f_s(x) - z)^2 < t^2)$$

We can also define the likelihood for a state x as the mean squared error, defined by the difference between the real and the expected measurement.

$$f_1(x) := (f_s(x) - z)^2$$

Figures 2 and 3 show this function for selected examples.

4.2.2. Information Gain of a Past Measurement

When constraining the state based on information of a past measurement, the control actions since the observation have to be taken into account. With the inverse of the control function defined in 3.2, we get the set

$$X_{z,u} := (x \in X : f_s(f_c^{-1}(x, u)) - z = 0)$$

of possible states (assuming perfect measurements and execution of controls). With the assumption of a maximum error t_z in the measurement, we can define a larger set

$$X_{(z,u),(t_z)} := (x \in X : (f_s(f_c^{-1}(x, u)) - z)^2 < t_z^2)$$

of possible states. And finally we can define a function that describes the likelihood for being in state x after executing control action u and having made the observation z in the beginning:

$$f_2(x) := \left(f_s(f_c^{-1}(x, u)) - z\right)^2$$

This likelihood is given by the squared difference between the real measurement and the expected measurement in the state $f_c^{-1}(x, u)$ calculated from x using the inverted motion model.

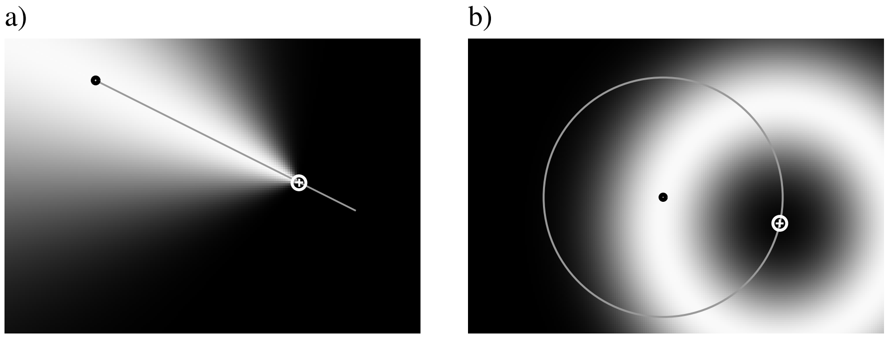

Figure 2. Information gain of a single measurement. White circle: position of the landmark. Black circle: simulated position of the robot. a) Gray line: bearing measurement from the robot to the landmark. The squared difference between the real and the expected bearing measurements to the landmark is shown by the visualization of the function in the background: $f(x, y) = (\arctan(y_l - y, x_l - x) - \alpha)^2$ where α is the angle between north and the bearing from the robot to the landmark. Note that bright areas stand for low values of the function and dark areas for a high value. b) Gray circle: distance measurement from the robot to the landmark. The squared difference between the real and the expected distance measurements to the landmark is shown by the visualization of the function in the background: $f_1(x, y) = (\sqrt{(y_l - y)^2 + (x_l - x)^2} - d)^2$ where d is the distance between the robot and the landmark.

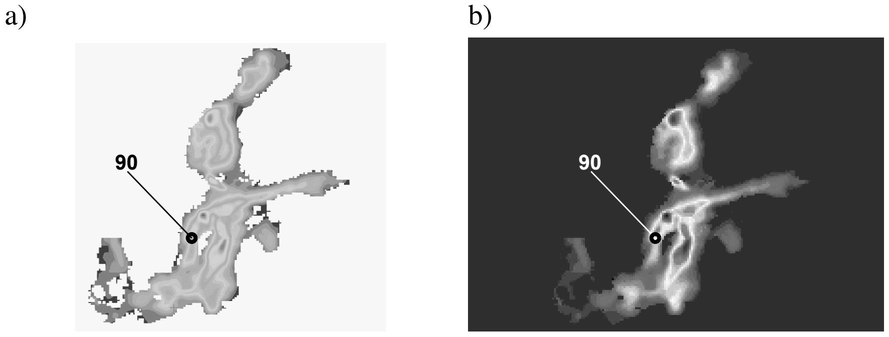

Figure 3. Information gain of a single depth measurement. a) Map of the Baltic Sea. The black circle with the small number shows the simulated position of the robot and the result of a single depth measurement (90m) b) Function $f(x, y) = (d(x, y) - s)^2$ which shows the squared difference between the real and the expected measurements for each location. The function $d(x, y)$ delivers the depth for position (x, y) as stored in the map, s is the depth measurement. Places with a small difference are shown bright.

4.2.3. Information Gain of Multiple Past Measurements

To accumulate the information of multiple past measurements we define the function

$$f_M(\vec{z}, \vec{u}) := \arg\min_k \sum_{t=1}^{n} (f_s(f_c^{-1}(x_k, v_t)) - z_{n-t})^2$$

which calculates the state x with the least sum of squared differences between the real measurement z_{n-i} and the expected measurement for all states $f_c^{-1}(x, v_i)$. Additionally the value of the minimum is a measure how good the measurements fit. If all measurements are perfect, the minimum has a value of 0. The more contradictions there are between the single measurements, the higher the value of the minimum is. The shape of the function gives additional information on the *contribution* of single measurements to the total result. How this can be used to filter out measurements with a low contribution is discussed in section 5.2.

4.3. Properties of Memory-Based State Estimation

In this section we show, that the way to calculate the state using $f_M(\vec{z}, \vec{u})$ as defined above produces the same results as Bayesian filtering under the assumption that there is no error in the execution of actions and Gaussian error in the sensor measurements.

Assumption 4.1. There is no error in the motion model:

$$p(X_t = x_k | X_{t-1} = x_i, u_t) = \begin{cases} 1, & f_c^{-1}(x_k, u_t) = x_i \\ 0, & otherwise \end{cases}$$

Assumption 4.2. The sensor model has Gaussian error:

$$p(z | X_t = x_k) = \mathcal{N}\left[f_s(x_k), \sigma\right](z)$$

Theorem 4.1. Let $\vec{u} = u_1, ..., u_n$ be a sequence of controls and $\vec{z} = z_0, ..., z_{n-1}$ be a sequence of according measurements. Then the state $x_m = f_x(\vec{z}, \vec{u})$ determined by *MBSE* is equal to the state $x_b = \arg\max_k p(X_t = x_k)$ calculated by iterative Bayesian filtering when the assumptions 4.1 and 4.2 hold.

Proof:
Consider line 2 of algorithm 3.2

$$\bar{p}(X_t = x_k) = \sum_i p(X_t = x_k | X_{t-1} = x_i, u_t) \cdot p(X_{t-1} = x_i).$$

Following the assumption that there is no random in the motion model:

$$p(X_t = x_k | X_{t-1} = x_i, u_t) = \begin{cases} 1, & f_c^{-1}(x_k, u_t) = x_i \\ 0, & otherwise \end{cases}$$

we can eliminate the sum in the calculation and get:

$$\bar{p}(X_t = x_k) = p(X_{t-1} = f_c^{-1}(x_k, u_t))$$

Substitution in line 3 of algorithm 3.2

$$p(X_t = x_k) = \eta \cdot p(z_t | X_t = x_k) \cdot \bar{p}(X_t = x_k)$$

gives

$$p(X_t = x_k) = \eta \cdot p(z_t | X_t = x_k) \cdot p(X_{t-1} = f_c^{-1}(x_k, u_t))$$

The iteration of n observations and controls results in

$$\begin{aligned} p(X_n = x_k) &= \eta_{n-1} \cdot \eta_{n-2} \cdot \ldots \cdot \eta_1 \cdot \\ & \quad p(z_{n-1} | X_{n-1} = f_c^{-1}(x_k, u_n)) \cdot p(z_{n-2} | X_{n-2} = f_c^{-1}(x_k, u_{n-1} u_n)) \cdot \ldots \cdot \\ & \quad p(X_0 = f_c^{-1}(x_k, u_1 u_2 \ldots u_n)) \\ &= p(X_0 = f_c^{-1}(x_k, v_n)) \cdot \prod_{i=1}^{n} \eta_i \cdot p\left(z_{n-i} | X_i = f_c^{-1}(x_k, v_i)\right) \end{aligned}$$

using definition 3.4. The first term[1] and the η_i can be joined to a constant ζ_1 resulting in:

$$p(X_n = x_k) = \zeta_1 \prod_{i=1}^{n} p(z_{n-i} | X_i = f_c^{-1}(x_k, v_i))$$

Since $\ln$ is a strictly increasing function, it holds true

$$\arg\max_k p(X_n = x_k) = \arg\max_k \ln(p(X_n = x_k))$$

thus we can deduce

$$\arg\max_k p(X_n = x_k) = \arg\max_k \left(\ln \zeta_1 + \sum_{i=1}^{n} \ln p(z_{n-i} | X_i = f_c^{-1}(x_k, v_i)) \right)$$

According to the Assumption 4.2 it holds true

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

which implies

$$\ln(p(z|x)) = -\frac{1}{2} \left(\frac{z - f_s(x)}{\sigma} \right)^2 + \zeta_2$$

Applying this, we finally get

$$\arg\max_k p(X_n = x_k) = \arg\min_k \sum_{i=1}^{n} \left(f_s(f_c^{-1}(x_k, v_i)) - z_{n-i} \right)^2$$

which is exactly the formula from section 4.2.3. □

[1]The term $p(X_0 = x)$ describes the a-priori probability distribution of the robot-state. At this point we assume that the initial state of the robot is unknown and thus uniformly distributed, which implies that $p(X_0 = x)$ is a constant for all x.

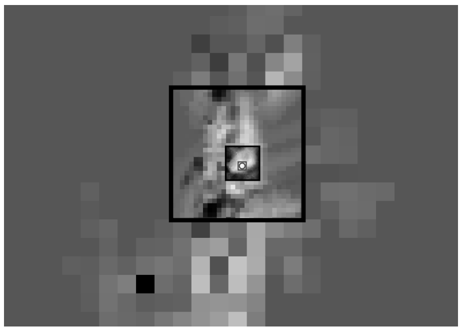

Figure 4. Stepwise Grid Refining. The global minimum of the function is found using a grid. The spacing of the grid is reduced in several steps.

5. Algorithmic Variants of MBSE

Calculating

$$f_M(\vec{z},\vec{u}) := \arg\min_k \sum_{t=1}^{n} (f_s(f_c^{-1}(x_k,v_t)) - z_{n-t})^2$$

can be done in several ways. In this section we show a few of them. Some of the methods reduce the number of states when looking for the minimum. Other variants reduce the number of measurements that have to be incorporated.

5.1. Reducing the number of Locations

In this subsection we discuss several ways to reduce the number of locations for which the function f(x,y) has to be calculated.

Stepwise Grid Refining This method calculates the function only for positions (x, y) that are located on a grid with a spacing of d. The value of d has to be chosen depending on the domain in a way such that there is always a grid point close to the real minimum of the function. The algorithm finds the grid cell with the minimal value of the function. This grid cell is used to define a new grid around this cell with a smaller spacing. This process is repeated until the desired precision of the position is determined. Figure 4 shows an example.

Gradient Descent The minimum of the function can also be found using the gradient descent method. Parameters for the algorithm like number of iterations and step length have to be chosen depending on the domain. Figure 5a) shows an example.The gradient descent algorithm might have the problem to run into local minima. However, once the true minimum was found (for example using the stepwise grid refining method described above), choosing this position as starting position in the next step should prevent the algorithm from switching to the wrong local minimum.

Gradient Descent with Multiple Starting Points Another way to solve the problem of local minima is to select a set of starting positions and run the gradient descent method for each of these positions. The

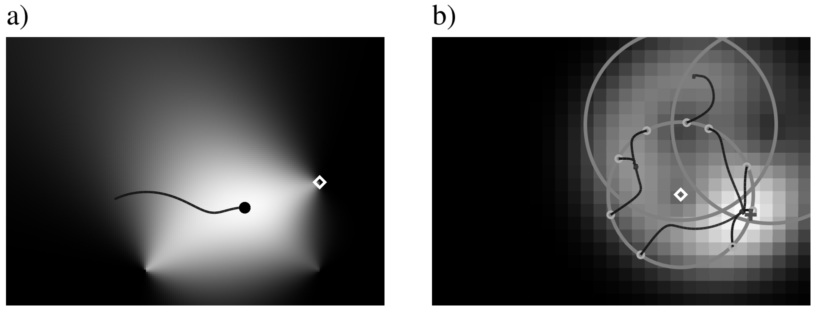

Figure 5. a) *Gradient Descent.* The function in the background results from combining three bearing measurements to the landmark (white rhombus). As there is only one local and global minimum the gradient descent finds the position of the robot (marked by the black circle). The line illustrates the path of the gradient descent. b) *Gradient Descent with Multiple Starting Points.* To be sure not to end in a local minimum, gradient descent is started at several positions. The function in the background results from combining three distance measurements. The starting points are chosen by randomly selecting points out of the set $X_{(z,u),(t_z)}$ which contains all possible positions taking only measurement s into account. The set of gradient descent runs finds all three local minima of the function.

run that ends up in the lowest minimum can be considered to have found the global minimum. However, some care has to be taken to select starting positions. We propose a way to choose starting positions based on the knowledge obtained by the last measurement s. We define the set of starting positions as a random subset of $X_{(z,u),(t_z)}$ (cf. 4.2.1) with a fixed number of elements, where z is the last measurement, u the control action performed since that measurement, and t_z the maximal expected measurement error. The size of the set and the value of the threshold t_z have to be chosen depending on the domain. This set is expected to contain at least one position near the global minimum. Which is a good starting position for gradient descent. Figure 5b) gives an example.

Dimension reduction In this section we show how the calculation can be simplified when the sensor model fulfills special requirements. Let $g(\vec{x})$ be the sensor model which predicts a measurement for a state vector $\vec{x} = (x_1, ..., x_n)$ of dimension n. In section 4.2.2 we defined the function

$$f_2(\vec{x}) := \left(f_s(f_c^{-1}(\vec{x}, u_i)) - z_i\right)^2$$

which calculates the difference between the measurement and the expectation for a given state $\vec{x}$ for an observation (z_i, u_i). Therefore the solution of equation

$$z_i = f_s(f_c^{-1}(\vec{x}, u_i))$$

is a set of states $\vec{x}$ for whose elements the expectation is identical to the measurement. If we fix the first $n-1$ components of $\vec{x}$ and require the expectation to be identical with the measurement we get a constraint for the component x_n of $\vec{x}$ which we use to define the function

$$c\left((x_1, ..., x_{n-1}), z_i, u_i\right) := x_n.$$

Note that from our examples only the *bearing world* has a sensor model that can be used to define such a function.

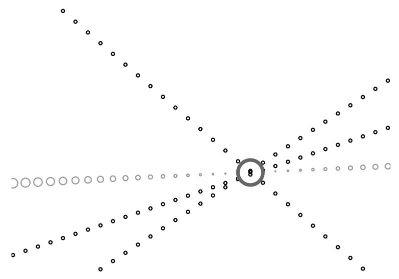

Figure 6. Dimension Reduction in the *bearing world*. Three measurements were taken. The lines of black circles show the result of the computation of function $c(x, \alpha, \Delta)$ for several discrete values of x for each observation (α, Δ). The light gray circles show the function $\bar{c}(x)$. The size of the gray circles visualizes function $h(x)$. The position x_{best} with the smallest value of h is marked by the large dark gray circle. Figure 5 shows the function f for the same experiment.

We use that function c to define a function

$$h(\tilde{x}) := \sum_{i=1}^{k} (\bar{c}(\tilde{x}) - c(\tilde{x}, z_i, u_i))^2$$

which gives for each partial state $\tilde{x} = (x_1, ..., x_{n-1})$ how good several observations (z_i, u_i) fit together. The function $\bar{c}(\tilde{x})$ is defined as

$$\bar{c}(\tilde{x}) = \frac{1}{k} \sum_{i=1}^{k} c(\tilde{x}, z_i, u_i).$$

With these functions we can estimate the position:

$$\vec{x_{best}} = \begin{pmatrix} \text{argmin}_{\tilde{x}}(h(\tilde{x})) \\ \bar{c}(\text{argmin}_{\tilde{x}}(h(\tilde{x}))) \end{pmatrix}$$

This increases the calculation speed as the dimension of $\tilde{x}$ is only $n-1$.

In a 2-D world with bearing sensors we get

$$y = c(x, \alpha, \Delta) = y_l - \Delta_y - (x_l - x - \Delta_x) \cdot \tan(\alpha)$$

where Δ_x and Δ_y are the components of $\Delta := f_c(v_i)$. Figure 6 visualizes functions c and h for an example.A detailed description of the dimension reduction for a three-dimensional example (position and rotation) can be found in [3].

5.2. Reducing the number of Used Measurements

The second method to reduce the computational costs of the algorithm is to reduce the number of measurements. So far we have no restrictions on the size of the memory, which means that the longer the robot runs, the more information is processed.

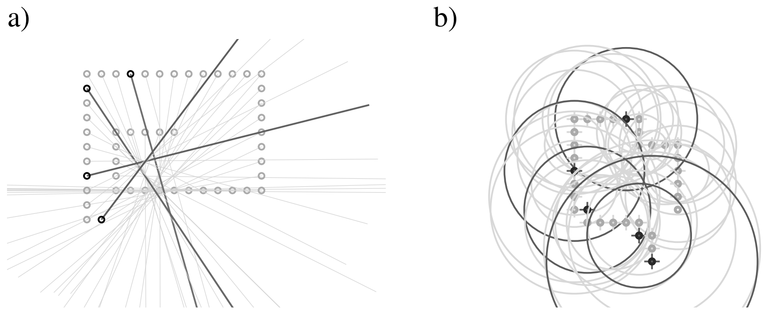

Figure 7. Selecting observations based on their contribution. The figures show two examples. One from a bearing measurement example and another one from a distance measurement example. The faded measurements were not considered as they were marked to have a low contribution during the past run of the robot. Only the measurements taken at the highlighted places are used to calculate the position.

Remove Old Measurements Removing older measurements is the obvious solution to this problem. We tested and compared several strategies of selecting observations in [5] and showed that with the right strategy there is almost no loss of information.

Remove measurements with a low contribution In this section we want to motivate an observation selection strategy based on the *contribution* of a measurement. First we define a function which calculates the direction of the steepest decent of function f at position x

$$a(\vec{x}, M) = \max_{\vec{y}:|\vec{y}|=\varepsilon} \{\nabla f(\vec{x}+\vec{y}, M)\}$$

Note that this function can also be applied to a local extremum as for the steepest decent is searched at all locations $\vec{x}+\vec{y}$ with a distance of ε to $\vec{x}$. With that we define the *contribution* of a set of observations N with respect to all observations M as the angular difference between the directions of the steepest decent.

$$c(N, M) := 1 - \frac{a(x_{best}, M) \cdot a(x_{best}, N)}{|a(x_{best}, M)| \cdot |a(x_{best}, N)|}$$

where $x_{best} = \operatorname{argmin}_x f(x, M)$. For practical reasons ε should be greater than the difference between the real minimum and the estimated minimum x_{best} of the function f.

The function $c(N, M)$ that calculates the *contribution* can be used to define selection strategies for observations. The simplest one is to keep the number of elements in N fixed and as soon as a new observation is made, the observation with the lowest contribution is deleted. Figure 7 shows the result of this strategy for experiments with bearing and distance sensors in a 2-D world.

6. Experiments

To test and demonstrate the method of state estimation we chose a self-localization scenario from RoboCup. The experiments we conducted are described in this section.

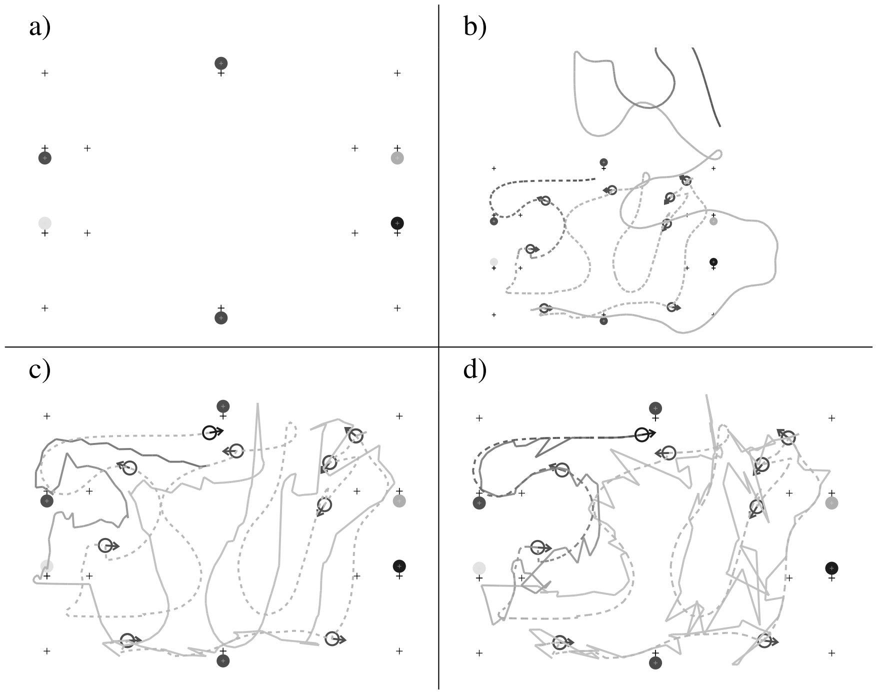

Figure 8. Experimenal results. a) The artificial landmarks of a RoboCup field. Circles: unique markers for horizontal bearing measurements (center beacons, goal posts). Small crosses: Intersections of field line (not distinguishable). b) Dashed line: The true path the robot took in simulation, starting in the left bottom corner; Solid line: result of localization based only on the knowledge about the effect of actions. Note that the result gradually drifts away from ground truth due to the noise added to the motion model and do to the small kidnappings. Circles with arrows along the solid line: Poses where the robot was kidnapped randomly (simulating charging by other robots). c) Solid line: result of MBSE-localization using all perceptions. d) Solid line: result of MBSE-localization using only the last perceptions of each type of landmark.

6.1. Experimental Setup

Our testing scenario was bearing-only localization [5, 4] on a RoboCup field. In this case the only measurements the robot is able to use are horizontal bearings to landmarks. There are six unique landmarks (the two goal posts of each goal and the center beacons) and fourteen ambiguous landmarks (intersections of field lines). The setup of landmarks is shown in figure 8a). All tests were done in simulation which enabled us to compare the results of localization to true positions. Like on a real robot, in simulation the bearing sensors had a limited field of view (60 degrees) and an Gaussian error with a standard deviation of 6 degrees. The motion model we used had an error of 10 percent of the amount of the action in all three dimensions (x, y, rotation). This error causes the dead-reckoning robot position to drift away from ground truth (see figure 8b). Additionally we added small kidnappings as they appear regularly in RoboCup games when robots collide. To test self-localization we made the robot follow a virtual ball that was moved around on the field. Resulting in the path shown in figure 8b) and compared the results to the ground truth position.

Selection strategy	Error in mm
motion only	$277,4 \pm 183,4$
all landmarks	$52,3 \pm 21,7$
last 5 of each type of landmarks	$25,7 \pm 19,8$

Table 1. Experimental results. The selection strategies are denoted in the left column. The right column contains the resulting averaged difference between the calculated position and the ground truth.

6.2. Experimental Results

To measure the quality of localization, we measured for each time step the difference between the estimated pose found by our memory based approach and the ground truth position. We measured this for three different scenarios. The first one is localization based only on motion data (initialized with the true position in the first step). The second version used the complete history of measurements and executed motions for position calculation. In the last experiment we used only the last observations of each type of method. Results are depicted in table 1 and figure 8c,d).

As explained above, using only motion information leads to a drift of the estimated position leading to a high deviation. Using all information still has this problem due to the kidnappings. Using only the last information leads, as expected to the best results. However, due to the small amount of measurements and due to the noise in this case the position is subject to small jumps.

7. Conclusion

In this paper we motivated and described *Memory Based State Estimation (MBSE)* which works on a buffer of stored observations which contains the measurements as well as control data. We introduced the mathematical prerequisites, the formal model and possible applications. The system is perfectly suited to process sparse information which was shown in experiments. In future work we want to further examine how the contribution and the knowledge about possible contradictions derived from the value of the minimum of the likelihood function can be used to create observation selection strategies.

References

[1] M. Deans and M. Hebert. Experimental comparison of techniques for localization and mapping using a bearingonly sensor, 2000.

[2] D. Göhring, K. Gerasymova, and H.-D. Burkhard. Constraint based world modeling for autonomous robots. 2007. Proceedings of the CS&P 2007.

[3] M. Jüngel. Bearing-only localization for mobile robots. In *Proceedings of the 2007 International Conference on Advanced Robotics (ICAR 2007), Jeju, Korea,*, August 2007.

[4] M. Jüngel. Memory-based localization. 2007. Proceedings of the CS&P 2007.

[5] M. Jüngel. Self-localization based on a short-term memory of bearings and odometry. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, San Diego, October 2007. to appear.

[6] S. Lenser and M. M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA 2000)*, pages 1225–1232. IEEE, 2000.

[7] D. Seidman. *The Complete Sailor - Learning the art of Sailing*. International Marine, Camden, Me., 1995.

[8] J. Sola, A. Monin, M. Devy, and T. Lemaire. Undelayed initialization in bearing only slam, 2005.

[9] M. Sridharan, G. Kuhlmann, and P. Stone. Practical vision-based monte carlo localization on a legged robot. In *IEEE International Conference on Robotics and Automation*, April 2005.

[10] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, Cambridge, Mass., 2005.

[11] J. Wolf, W. Burgard, and H. Burkhardt. Robust vision-based localization for mobile robots using an image retrieval system based on invariant features. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.