Input combination for Monte Carlo Localization

David Obdržálek

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské náměstí 25, 118 00 Praha 1, Czech Republic
david.obdrzalek@mff.cuni.cz

Abstract. *One of the basic skills for an autonomous robot is the ability to determine its own position. There are numerous high-level systems which provide precise position information, but the same task may be also solved using less advanced and less accurate sensors. In our paper, we show how a good output may be acquired from not so good inputs if it is combined using modified Monte Carlo Localization (MCL) system. The combination of more inputs also helps to acquire plausible results even in situations, where adding new sensor(s) to an existing system raises doubts about the position calculation, because the newly added sensor may provide different position information (or data from which the position is calculated) than what is provided by the already used sensors. We will show that using such "incorrect" data may be beneficial for determining the position with a reasonable probability.*

1 Introduction

Good localization is for an autonomous robot one of the keys to success. A robot which does not know its position, or which gets lost while performing its task, is not something what could be used in real life well. For some tasks, localization can be quite simple, but in general, the better autonomous robot we want, the better (and usually more complicated) localization it needs. Some systems use single input for the localization task and do not need any complex mechanisms to accomplish all needed goals. Other systems combine more inputs (and more input types) to acquire data for the localization process; it may be because of different nature of the data which is available to be measured for the localization as well as because different sensing methods may help to overcome problems with erroneous data coming from individual sensors. However, combining more inputs may at the same time rise questions about the preciseness of the localization process: What went wrong if two or more inputs showed different positions? Is it because of faulty sensor, inexact measurement, cumulative errors or improper data handling?

Recently, the research in the robot localization started to bring very good results using probabilistic methods. In this paper, we discuss this problem in general, and we present one particular implementation which uses Monte Carlo Localization (MCL).

The following text is organized as follows: Section 2 gives characterization of the task and presents the specifics of our selected problem. Section 3 gives brief outline of existing localization techniques. Section 4 presents Monte Carlo Localization in general, Section 5 shows our modification to MCL by differentiating between sensor classes and Section 6 shows some implementation aspects. Sections 7 and 8 summarize the results and conclude the paper.

2 Motivation and characterization of the task

The goal of robot localization is to determine the position of the robot which moves through its working environment. It may use data about the environment and data about the robot, both using measured data as well as data known in advance. To a great extent, the localization algorithm itself can be application independent and the usage for specific purposes can vary just by choosing different inputs.

Inputs for the localization system may come from different sources: data may be acquired for example using different sensors mounted on the robot (measuring either internal or external properties), by receiving signals sent from external sources, or even created as virtual data which does not represent any real measurements (e.g. expected position change based on commands issued by the control system). Because the different sensors provide data with different level of accuracy and trustworthiness, data should be also handled with different weights.

The localization algorithm processes the selected inputs to calculate the robot position. If the algorithm itself does not depend on a specific type(s) of input data, then it is possible to create a generic solution which works with different (and configurable) sets of sensors.

In this paper we present one possible solution of the localization problem which has been tested on a real robot. This robot was used to play in the Eurobot autonomous robot contest in 2009 (see [1]). Although the system was created for the 2009 edition of the contest, it was designed so that it could be used without reprogramming for future editions too. Moreover, it was

designed to be independent on this particular task at all and it may be reused in other projects with different inputs as well.

The Eurobot contest rules change every year, but they always share the same core of basic characteristics (for more details, see the Eurobot Association web pages at [1]):

- Known indoor environment with highlighted important landmarks
- Small working area (2.1 × 3 meters)
- Possibility to place localization beacons at predefined places around the working area
- Target objects placed semi-randomly on the working area
- Predefined starting position of the robot
- Limited height and circumference of the robot
- Two robots moving in the area at the same time with the necessity to avoid collisions

This list obviously affects the development of robot hardware and software. However, our aim was to create a localization algorithm which is not that much application dependent and can be used for other applications in different conditions too.

The exactly needed precision level is application dependent and varies a lot from one application to another. For the specific conditions it is however important to maintain the precision in an acceptable range. Therefore, our aim was to create a localization algorithm which would be able to reach the required precision level even using less precise inputs. The precision should not be predefined nor implied by the algorithm.

3 Localization algorithms

The area of autonomous robot localization is well researched (see e.g. [2]), and several ways can be used to solve the localization problem. Therefore we do not try to invent a new algorithm. Instead, we will outline some existing localization algorithms and discuss some of their implementation details, together with technical problems we have met.

For localization based on various input values one can choose from many algorithms; the most know are:

Kalman filter [3, 4] generalizes the floating mean. It can handle noisy data so it is suitable for processing the data from less precise sensors. However, the model must be described with the expected value and variability which is often too difficult constraint.

Markov localization based on grid [5] resolves one of the problems of Kalman filter, which needs to know the expected value and the variance of input data. This algorithm splits the area to the grid of proper size and tries to determine the one the robot is in. Unfortunately, this requires large operational memory to store the data and computing power to handle it.

Monte-Carlo localization [6] can represent multimodal distribution and thus localize the robot globally. It can process inputs from many sensors with different accuracies. Moreover, it can be easily implemented.

For our given task, it is not possible to use standard Kalman filter, because its basic requirements are not met: in our case, the expected value and variance of the measured values are not known.

The second mentioned localization algorithm, Markov grid-based localization, would overcome this problem, but would impose another problem at the same time – the need to handle lot of data. The position of a robot is specified as one cell in a grid covering the whole working area. It is needed to store some data for each grid cell, and to reach good precision level, the grid must be fine. Our hardware platform provided sufficient storage with reasonable power for processing the navigation task and for controlling the hardware, but including Markov localization would cause overloading of the system and the goal to create a successful autonomous robot could not be reached: neither the memory volume nor the computational power of our hardware were strong enough to handle Markov grid-based localization alone, not talking about combining it with all the other needed tasks.

Therefore, we have decided to implement Monte-Carlo localization and let it use the remaining resources in the system as long as it does not affect its functionality. This also directly implied the selection of the MCL parameters in the tuning phase – "eat as much as you like as long as supply lasts". At the same time, we gained the possibility to use more different sensors for the localization task.

The Monte-Carlo localization will be further discussed in Section 4 and our implementation in Sections 5 and 6.

4 General description of MCL

In this section we will briefly outline Monte Carlo Localization (MCL), introduced by Dieter Fox in the late 1990s [6]. This algorithm meets all the requirements mentioned in problem statement section earlier in this paper. It is a well defined and researched algorithm and it is also well established in many applications (see e.g. [7–10]).

Monte Carlo Localization maintains a list of possible states of the state space (representing the positions of the robot). Each state is weighted by its probability

of correspondence with the actual state of the robot. In the most common implementation, the state represents the coordinates in 2D Cartesian space and the heading direction of the robot. It may be of course easily extended to 3D space and/or contain more information depicting the robot state. All these possible states compose the so called probability cloud.

The Monte Carlo Localization algorithm consists of three phases: Prediction, Measurement, and Resampling.

During the Prediction phase, a new value for each item of the cloud is computed, resulting in a new probability cloud. To simulate various inaccuracies that appear in a real hardware, random noise is added to each position in the prediction phase. This is very useful. For example: If the wheels were slipping and no random noise was added, the probability cloud would travel faster than the real hardware.

During the measurement phase, data from real sensors are processed to adjust probability of the positions in the cloud. The probability of samples with lesser likelihood (according to sensors) is lowered and vice versa. For example, when the sensors show the robot orientation is northwards, weight for samples representing other orientations is lowered.

The last phase - resampling - manages size and correctness of the cloud. Samples with probability lower than a given threshold are removed from the cloud. To keep the number of positions constant, new positions are added. These new positions are placed around existing positions with high probability.

Formally, the goal is to determine robot's state in step k, presuming the original state and all the measurements $M^k = \{m_i, i = 1..k\}$ are known. Robot's state is given by a vector $x = \langle x, y, \alpha \rangle$, where x and y is the robot position and α is its heading.

During the prediction phase, the probability density $p\left(x_k \mid M^k\right)$ for step k is enumerated. It is based only on presumed movement of the robot without any input from real sensors. Therefore, for any known command u_{k-1} given to the robot, we have

$$p\left(x_k \mid M^{k-1}\right) = \\ = \int p\left(x_k \mid x_{k-1}, u_{k-1}\right) p\left(x_{k-1} \mid M^{k-1}\right) dx_{k-1}$$

In the measurement phase, we will compute the final value of probability density for actual step k. To do so, data from sensors is used. It implies the probability of $p\left(m_k \mid x_k\right)$, where m_k is the actual state and x_k is the assumed position. The probability density in step k is then described by the following equation:

$$p\left(x_k \mid M^k\right) = \frac{p\left(m_k \mid x_k\right) p\left(x_k | M^{k-1}\right)}{p\left(m_k | M^{k-1}\right)}$$

During the initialization of MCL, it is needed to set the probability cloud. If the robot's position is known, then for its (known) state x the probability $p\left(x \mid M^0\right) = 1$, and for all other states $y \neq x$ the probability $p\left(y \mid M^0\right) = 0$.

If the robot's position is not known, the probability of all positions is the same and $p\left(x \mid M^0\right)$ must be set for all x so that

$$\int p\left(x \mid M^0\right) dx = 1$$

One of the most important features of this method is its ability to process data from more than one source. Every sensor participates on computing the probability for the given state. For example, if we have a compass sensor and it reads that the robot is heading to the north, we can lower the probability of differently oriented samples. If robot's bumper signalizes collision, there is a high probability for the robot to be near a wall or another obstacle. It is therefore possible to discard the part of the probability cloud which lies in an open space.

The Monte Carlo Localization can be implemented easily, yet it provides very good results especially in cases, where the sensors do not provide exactly correct data (e.g. the distance measurement is subject to errors). Our implementation of the MCL algorithm, which shows its great usability, is described in more detail in the following section.

5 Sensor classes in modified MCL

We have decided to modify the original MCL algorithm and use sensor input also for the prediction phase. We allow selected reliable sensors to change the position of MCL samples in addition to changing the weights of samples based on the sensor readouts. This allows to maintain the probability cloud more in shape with the actual robot movement, yet we keep the core MCL idea of adding random noise to handle unexpected inputs or inputs which are not in accordance to actual robot movement.

In our implementation, we divide the inputs coming from sensors in two categories, which will be further discussed in following paragraphs:

- Advancing inputs
- Checking inputs

Our system contains two interfaces for these two types of inputs. The device or its abstraction in Hardware Abstraction Layer implements the corresponding interface based on its type, so the MCL core can use it as its input. The MCL core calls each device when it has new data, and the work with the samples is done

by each device separately. This keeps the main code easier to read, simpler, and input independent. Also, the device itself knows the best how to interpret the raw data it measures.

The level of reliability can be specified for each input device. Then, the samples are adjusted by the devices with respect to their configured 'credibilities'. For example: two sets of odometry encoders, one pair on driven wheels and one pair on dedicated wheels, have different accuracy because the driven wheels may slip on the surface when too much power is used. Then, the credibility of driven wheels encoders will be set lower than the credibility of the sensors mounted on undriven wheels. In addition, setting the reliability level helps to deal with different frequencies of data sampling.

5.1 Advancing inputs

This input type is used for changing the samples which form the probability cloud. Such input could be for example odometry, from which relative movement since last iteration is calculated. This difference is then used to change the samples properties. i.e. to move them. The information provided by these kind of inputs applied to samples is 'blurred' by randomly generated noise as described earlier in the general MCL description. After moving the samples, boundary conditions are checked (i.e. to exclude samples which fell out of the physical working area). As a result the probability of samples representing impossible positions is decreased.

These advancing inputs are added to the original MCL algorithm, which deals only with theoretical movement based on movement commands. It is not necessary to use it so, but it brings much better precision for little cost.

Our robot currently uses only one advancing input: the odometry information from encoders mounted on dedicated sensor wheels.

5.2 Checking inputs

Checking inputs do not affect the position of the samples. Instead, they are just adjusting their probability (also called sample weights). The reason for this is that inputs of this type provide absolute position information and not relative difference from the last measurement. This also does not need to be one exact point, but an area or position probability distribution, which fits perfectly to the Monte Carlo Localization algorithm.

All checking inputs may be processed separately; we regulate them only by setting their reliability levels.

Our robot uses these checking inputs:

- Compass - checks the direction of samples
- Beacons - check the distance from stationary beacons
- Bumpers - check collisions with playing field borders and other objects
- IR distance sensors - check distance to borders and obstacles

6 Implementation aspects

Our implementation of MCL is based on previous work on a robot which participated in Eurobot 2008 contest (see [11]). That first "pilot" MCL implementation in 2008 was not complete; it was rather proof-of-concept than a reliable software unit, and we also knew the computational power will be different if 2009 so performance was not considered at all. However, the results seemed very promising, so it has not been dropped but has been further developed and extended to use it in 2009 for real. In the following paragraphs, we emphasize several implementation aspects we consider as important for the successful result.

6.1 Position estimation

It is expected that the MCL outputs the estimation of robot position. Because of its nature, the resulting position ("most probable position") can be computed from the samples at any time. This estimation is very simple, just computing the weighted average of all samples. In addition we can determine the overall reliability of this estimation. Therefore, we have made the interface to the MCL asynchronous to the inputs, and the MCL core can be called at any time whenever needed. This approach obviously dramatically saves the computational power in comparison to incremental localization methods which might need periodical updates or re-calculations.

6.2 Localization without initial knowledge

Monte Carlo Localization can also determine the robot position from scratch. At the beginning of localization (when the robot is lost) samples are spread uniformly all over the playing field as described in Section 4. The sensors providing absolute positioning information lower the weight of misplaced samples and new samples are placed in regions with higher probability (see Figure 2). This is repeated until sufficiently reliable position estimation of the robot is reached.

At the same time, it is possible to reach a result even without absolute sensors – as the robot moves, the sensors which provide relative information (advancing inputs) will move the samples as usually and

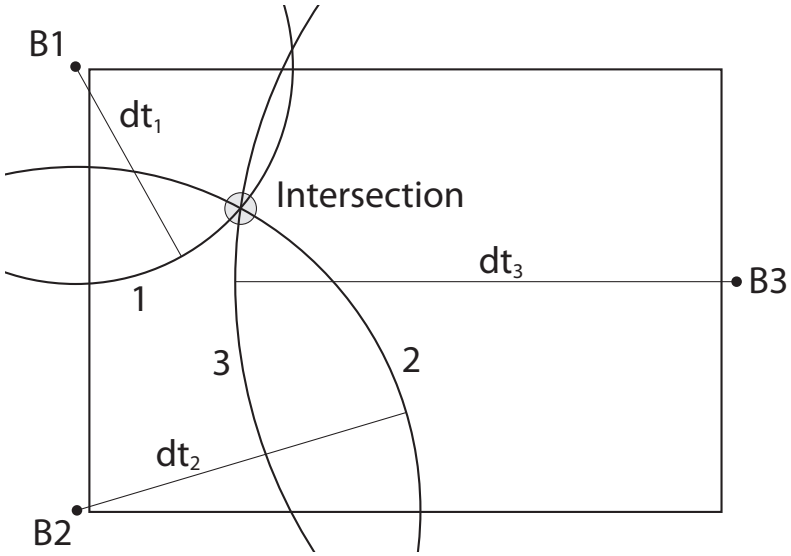

Fig. 1. Beaconing system; $B1$, $B2$, $B3$ are the beacons, and the robot (marked by grey circle) measured distances dt_1, dt_2, dt_3 from individual beacons.

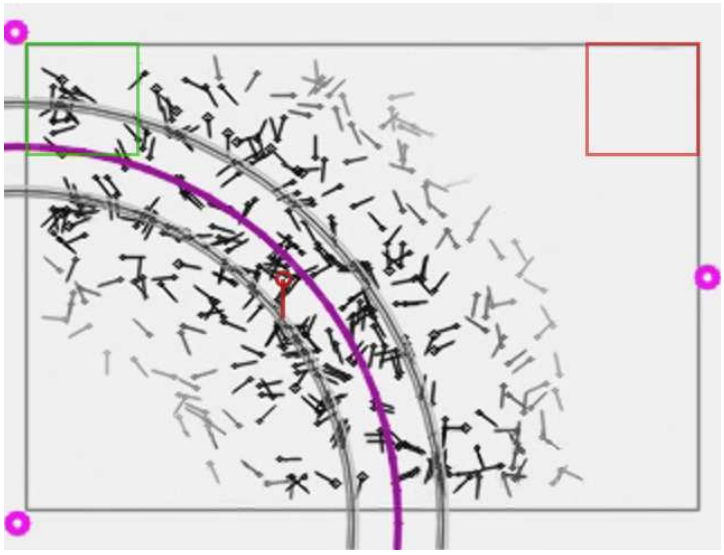

Fig. 2. MCL after processing one beacon input: The circular belt marks the input from the bottom left beacon. The "pins" represent oriented MCL samples; sample probability is proportional to their darkness.

the boundary checks gradually cut the impossible configurations until the required precision is reached (e.g. only one and small cloud remains).

6.3 Adding / removing sensors

When new sensors are added to a system, the information they provide may affect the position the robot "thinks" it is in. It may be because the new sensor gives better data (in which case we certainly appreciate the change). On contrary, the new sensor could provide data with lesser quality then the already existing sensors. This is not a big problem in MCL, because such low quality data may change the samples properties (position) or weight, but because of the nature how MCL works, such change may be perceived as adding the random noise which is part of MCL anyway. So, it may even help the algorithm to work well. It could be said in general – the more different inputs, the better.

If we remove a sensor from the robot or if it stops providing data and there are other sensors available, it does not imply the MCL results will be worse. It means there are fewer inputs which adjust the samples or their weights but the remaining sensors will adapt the probability cloud in accordance to their inputs and so sufficient level of result preciseness can be maintained. Therefore, the system is less vulnerable than a system which relies during the localization on one sensor or sensor set.

6.4 Beacons

In the following paragraphs, we present our design of a sensor set which provides relatively good information about the robot position and in cooperation with other sensors it helps to create a robust localization system – the beacon system.

The main idea of this beacon system is to mount several beacons around the working area of the robot and let the robot measure the distance to these beacons. Then, the robot will be able to estimate its position because the beacons position is known. This sensor set provides absolute position information, but its correctness may be attacked by the environment features, as for example the signal may get reflected by close obstacles like walls or the robot could not be able to measure the distance to one beacon because the signal may get lost (or get blocked by an obstacle).

The principle In our system, we measure the time the signal travels from the transmitter at the beacon to the receiver mounted on the robot (TOF – Time of Flight). Of course this works only if the speed is constant. This condition is met as we are using ultrasonic waves in a small environment with more or less constant conditions and the robot speed is negligible.

Since the speed of the sound waves is known, we can measure the time difference, from which the distance may be easily calculated. It is possible to use two beacons for good position calculation (provided the measurements are precise and we know at which halfplane the robot is). If there 3 or more beacons located around the working area, the trilateration will theoretically give a single solution. Practically, the measurements may not be precise and so the calculation may not lead to any intersection of the circles. In our system, this is not a problem, because the measurements are not used for trilateration but handled separately to adapt the probability cloud used in MCL.

To correctly measure the traveling time, we synchronize the transmitter-receiver system by using infrared light (see below).

Many other systems based on the TOF principle have been developed; for examples see e.g. [12, 13].

Hardware The transmitting system consists of three stationary interconnected beacons located at specified places around the working area of the robot. When the system receives signals from the three beacons and calculates the three distances, it can theoretically determine its position by using trilateration. In praxis, such simplest form does not fully work. The robot may move between receiving signals from all the beacons, some signals may not be received or they may provide incorrect information because of reflections. Even that, good estimation may be acquired as was discussed in Section 6.

The signal is sent one-way only, the receivers do not communicate with the transmitters. Therefore, there can be multiple independent identical receivers, which are able to determine its individual positions. These receivers may be then used for localizing more objects and if the information is passed to a single center, it may be of substantial help (for example to create opponent avoidance system by placing one receiver on the opponent and reading it by wireless transfer).

Transmitters at each beacon work in the following way:

1. Send timing information (infrared)
2. Wait for a defined period of time
3. Send distance measuring information (ultrasonic)
4. Wait for a defined period of time
   (this is sequentially repeated for all beacons)

As we want to measure time difference between the signal being sent and received, we need to have syn-

chronized clocks. This is done in step 1 by using infrared modulated signals. Besides that, the transmitted information contains additional information about the beacon which is going to transmit sound waves.

Sound waves are transmitted as ultrasonic waves, and are always transmitted only from one beacon at a time. The transmitted signal contains also the identification of the source beacon.

Receiver waits for the infrared timing information. When it is received, the receiver resynchronizes its internal timer and generates a message. These messages are transported to the localization unit. Every message contains time stamp, information that synchronization occurred, and the information about beacon which is going to transmit ultrasonic information in this time step. Upon reception of the timing information, the receiver switches its state to wait for ultrasonic signal. When correct ultrasonic information arrives, the receiver generates similar message as is the message after IR reception, but containing time stamp for ultrasonic reception and beacon identification transmitted in the ultrasonic data. The difference in these two timestamps is linearly dependent to distance with a constant offset (the two signals are not transmitted exactly at the same time). Since each beacon identifies itself in both infrared and ultrasonic transmissions, the probability of mismatch is reduced.

When the infrared information is not received, a message is generated saying the synchronization did not occur and the timestamp is generated from previously synchronized internal clock. When the ultrasonic information is not received, localization unit is notified that nothing was received.

The situation after three successfully received ultrasonic signals with synchronized clock can be seen in Figure 1.

6.5 Beacons and MCL

As described earlier, our beacon system consists of three transmitting and one receiving beacons. The information is passed from the beacon system to the main computing unit via messages containing beacon id (i.e. transmitter identification) and time difference between the infrared and ultrasonic transmissions.

There are two reasons why each message contains the time difference (delta) instead of the calculated distance: computational power of the microcontroller and the degree of robustness. The main computing unit is more powerful than the receiving beacon, so we let the beacon do less work and we even benefit from this decision. We considered deltas to be the perfect raw data for our purpose - distance measurement. The computation is done in the main computing unit which controls all the other devices and is highly configurable. It means that all the parameters of the equation for distance calculation can be changed easily without the need of changing the beacons hardware or device firmware. It even allows us to calculate or adjust the parameters on the flight if distance information is provided based on external measurement.

The configuration of the main computing unit contains not only the important constants for the equation, but also the positions of the transmitting beacons. As we know the distance and the beacon id, we can increase the weights of the MCL samples in the circular belt formed by these two values and a range constant. MCL samples far from the belt are penalized (see Figure 2).

This approach is much better and more robust than just waiting for intersections and then computing the robot position using simple trilateration. These intersections may not happen very often because of the time gap between individual beacon transmissions (especially when the robot is moving fast). At the same time, it is good to implement different weighting for the samples on a belt, near an intersection of two belts and near the intersection of all three belts.

6.6 Camera

The idea of using camera for absolute robot positioning seemed very hard at the first time. Later, when we had the modular MCL implementation finished, we realized there is a great opportunity to use the information we get from the camera while looking for the playing elements positioned at predefined places of the playing area. Now, we can compare the playing element positions (acquired from the camera) with their fixed positions (defined by the Eurobot contest rules) and adjust the weight of the MCL samples to merge the two positions. For more details, see [14].

6.7 Gyroscope

In the early stages of robot design, we proposed to use a compass as one of the input sensors. However, using a compass in a small indoor competition is not a very good idea, because its precision can be degraded by influence of many factors (e.g. huge metallic cupboard, electromagnetism, steel concrete walls or metal structure building). Using a gyroscope instead of a compass would be much more efficient for our purposes, because gyroscope works completely independently and the influence of the environment is minimal. The only problem is the placement of the gyroscope itself, because it should be placed in the rotational axis of the robot.

7 The results and performance

Apparently, processing of a large number of MCL samples may have impact on performance of the whole localization system. In general, the more samples are taken into computation, the more precise the localization is, but the slower the computation is.

In our project, we have achieved acceptable speed using 400 samples. The acquired precision lies within a margin of single millimeters which is sufficient for the current task; should higher precision be needed, it could be easily reached by increasing the samples count and fine-tuning the weighting functions. Also, this number of samples and the resulting precision are independent on the working area size.

On contrary, the Markov grid-based localization requires to handle a grid with the size of the robot working area and the number of cells proportional to required precision. In the case of Eurobot contest, to reach the same precision we would need to handle a grid of 2100 x 3000 samples which is magnitudes higher than the 400 samples needed when using MCL.

The depicted modification of using sensors for the prediction phase instead of using just the information of expected robot movement has increased the precision too, as it takes into account not only where the robot was supposed to move, but where it actually moved. To be able to use sensors for this modification, it is needed to assure their good credibility. For our real robot, we have used odometry sensors mounted on dedicated wheels, which are not subject to skids and slides of the powered wheels. It has proven this is sufficient to reach a good level of precision.

8 Conclusion and future work

In our paper, we have described the advantages of the Monte Carlo Localization compared to other methods of position estimation and how we benefit of it in our implementation. Based on available sensor types, we have decided to adapt the MCL algorithm to use one sensor class to change the samples instead of using all sensors to change the sample weights only.

As a practical result, we have developed a modular system for robot localization which allows easy extension by different kinds of modules. Our implementation allows us to add more facilities with almost no or just minimal work effort and with no changes to the core localization itself at all, while increasing the precision of the resulting position. The created system was successfully used for Eurobot 2009 contest edition, and its design allows using it for other purposes too.

Because the system has been created for 2009 edition of Eurobot contest, we want to continue gathering testing data throughout the whole year of 2009 in the contest and all connected events when the working conditions for the robot remain unchanged. After finishing the year, we want to evaluate the performance of this MCL implementation to be able to judge its usage in other environments and with other hardware.

References

1. Eurobot Association: Eurobot autonomous robot contest: `http://www.eurobot.org`, 2009.
2. S. Thrun: *Robotic mapping: a survey.* In: Exploring artificial intelligence in the new millennium. Morgan Kaufmann Publishers Inc., 2003, 1–35.
3. R. Negenborn: *Robot localisation and kalman filters: On finding your position in a noisy world.* Master's thesis, Utrecht University, 2003.
4. G. Welch, G. Bishop: *An introduction to the Kalman filter.* Technical Report TR 95-041, University of North Carolina at Chapel Hill, 2004.
5. W. Burgard, A. Derr, D. Fox, A.B. Cremers: *Integrating global position estimation and position tracking for mobile robots: The dynamic Markov localization approach.* In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)., 1998.
6. F. Dellaert, D. Fox, W. Burgard, S. Thrun: *Monte Carlo localization for mobile robots.* In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA99), 1998.
7. E. Menegatti, M. Zoccarato, E. Pagello, H. Ishiguro: *Image-based Monte-Carlo localisation with omnidirectional images.* Robotics and Autonomous Systems **48**, 2004, 17–30.
8. D. Hähnel, W. Burgard: *Mapping and localization with rfid technology.* In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA05), 2004, 1015–1020.
9. O. Wulf, M. Khalaf-Allah, B. Wagner: *Using 3D data for Monte Carlo localization in complex indoor environments.* In: 2nd Bi-Annual European Conference on Mobile Robots (ECMR05), 2005, 170–175.
10. S. Lenser, M. Veloso: *Sensor resetting localization for poorly modelled mobile robots.* In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA00), 2000.
11. A. Mikulik, D. Obdrzalek, T. Petrusek, S. Basovnik, M. Dekar, P. Jusko, R. Pechal, R. Pitak: *Logion – a robot which collects rocks.* In: Proceedings of the EUROBOT Conference 2008, 276–287.
12. S.Y. Yi: *Global ultrasonic system with selective activation for autonomous navigation of an indoor mobile robot.* Robotica **26**, 3, 2008, 277–283.
13. L. Dazhai, F.H. Fu, W. Wei: *Ultrasonic based autonomous docking on plane for mobile robot.* In: IEEE International Conference on Automation and Logistics (ICAL 2008), 2008, 1396–1401.
14. S. Basovnik, L. Mach, A. Mikulik, D. Obdrzalek: *Detecting scene elements using maximally stable colour regions.* In: Proceedings of the EUROBOT Conference 2009.