ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Comparison and Experimental Validation of Stochastic Search Algorithms

Master Thesis

Stefan A. Frei

Advisors

Dipl.-Ing Stephan Huck
M.Eng. Nikolaos Kariotoglou
Prof. Dr. J. Lygeros

October 30, 2013

Abstract

The lately developed Markov chain Monte Carlo (MCMC) algorithm for stochastic localization of sources is a tool to solve the problem of optimizing a non convex function in the two-dimensional space. It utilizes autonomous vehicles to estimate the source(s) of the underlying concentration field. We compare its efficiency and robustness to a variety of existing methods with numerical simulations. In addition, their application is studied on an experimental test bed with mobile robots. First, auxiliary software is developed such as a visual robot detection method and a controller for the robots. The performance of the setup with the additional software and is analyzed. Second, we devise a collision avoidance technique for Stochastic Localization and show in simulation and experiments that it maintains its convergence property. All methods are implemented in this environment and compared to the simulation. We found that the significance of the results obtained from Stochastic Localization and the compared algorithms depend strongly on tuning and the MCMC methods exhibit in general exhibit a rather slower convergence rate to stationarity. Therefore, different criteria were investigated to analyze and rate the applicability of the methods to practical search scenarios.

Acknowledgments

I am very thankful to my supervisors, Stephan Huck and Nikolaos Kariotoglou. Their comments and remarks provided me with valuable inputs and insights, which without I would not have been able to write my thesis. Also, I'd like to thank Alex Trofimov who also wrote his thesis using the E-pucks at the same time. He helped me with understanding the hardware and during programming, and also when we painted the ground plate white. Further I thank my two predecessors, Pascale Meier and Rafael Caminada, for implementing the initial E-Puck detection method and the GUI. Without that, I would not have been able to implement my algorithms from scratch. Thanks go to Nathalie Schenk, who has revised my thesis. Further thanks go to Prof. J. Lygeros, representing the ifA and ETH Zurich, for equipping me with the necessary knowledge and providing me with the work space and hardware.

Stefan Frei

Contents

Chapter 1

Introduction

This thesis bases on the recently devised MCMC (Markov chain Monte Carlo) like algorithm for stochastic localization of sources [2].[1] The algorithm originates from a scenario where an underwater source emits a substance into the environment, which gives rise to a certain concentration field. Instead of trying to capture this field by applying modeling techniques and deduce the position of the source, autonomous underwater vehicles are employed to explore the region of interest. In essence, this approach would let the vehicles take measurements of the field at random positions to estimate the underlying concentration field. A different scenario is for example a rescue mission with drones in a vast area, where the goal is to locate the source of some signal. Stochastic Localization is thus a tool to locate the source(s) of some concentration field in a physical setup with multiple autonomous vehicles. Due to its convergence guaranty [2] we expect the search to be successful. A drawback which has Stochastic Localization in common with the others MCMC methods is the relative slow rate of convergence to its stationary distribution, i.e. how fast the chain converges to a distribution that reflects the real concentration field qualitatively.

Several approaches that address the same problem using autonomous vehicles have been developed. For instance, the idea, to let the vehicle follow the steepest ascent to the maximum, works perfectly in a concave function with a single maximum. Since in a real environment this kind of well behaved function is rarely the case we need to develop more general methods. Such a real world function could contain multiple local extrema, steep peaks on a flat ground or large regions with a small gradient. Furthermore in a real world environment we only have vague a priori information about the shape of the concentration field. Hence we cannot rely on mathematical optimization techniques that assume a predefined function.

So far Stochastic Localization is only described theoretically, it is proved in [1] and [2] to converge to a unique stationary distribution. It is therefore interesting how effective Stochastic Localization performs compared to a variety of alternative algorithms. In their paper the authors of [1] made some simplified assumptions about the properties of the autonomous vehicle, e.g. vehicle modeled as point mass, which are not suitable for a real application. Thus the question remains, 'is Stochastic Localization with multiple autonomous vehicles applicable to a real world setting?'

[1] For later refer to this algorithms as *Stochastic Localization*[2].

From a more general perspective, we face the problem of finding the global maximum of a function in the two dimensional space. Given the compact convex subspace $\mathcal{D} \subseteq \mathbb{R}^2$ and the concentration function $f\colon \mathcal{D} \to [0,1]$, the objective is to find

$$x^* = \arg\max_{x \in \mathcal{D}} f(x). \tag{1.1}$$

The focus of this thesis lies on Stochastic Localization, the main goal is to answer the above posed question. Thus we implement Stochastic Localization in an environment with multiple two-wheeled robots. The robots are modeled as point-masses with nonholonomic unicycle dynamics that are capable to take point measurements of the concentration field. Two of them are of deterministic nature and are used to provide non stochastic benchmarks. The first is called Grid Search, which is a simple algorithm that probes the function f on a predefined grid over the whole space. The second one is called Line Search. It is also deterministic, but the search axes depend on the current global maximum during the search, rather than on a predefined grid. The remaining two algorithms (Simulated Annealing, Metropolis-Hastings), as Stochastic Localization, fall in the class of MCMC type methods. They have in common that the next measurement position is randomly distributed and may either be accepted or rejected depending on some criteria which itself depends on the measured concentration. In order to benchmark Stochastic Localization, we conduct a theoretical assessment. We implement the five algorithms in MATLAB to carry out simulations in order to compare the performance of the algorithms. We rate the performance in terms of success rate and mission time. Before we can implement the methods on the robots, we need to develop a robot detection software, a controller for the robots as well as some sort of collision avoidance.

The thesis is structured as follows. The algorithms we consider are presented in Chapter 2. In Chapter 3 we show the results of our simulations. We select parameters for each algorithm and compare their performance for different scenarios. Then, in Chapter 4, the mobile robots and the setup are introduced. Chapter 5 presents the controller and explains the additional robot detection software. In Chapter 6, we implement a collision avoidance strategy into Stochastic Localization and, based on extensive simulations, provide evidence that it does not corrupt the algorithm. The final results of the implementation on the robots is given in Chapter 7. And in the last Chapter 8 we present our conclusion.

Chapter 2

Search Algorithms

In this chapter we present the agents model of the system and provide mathematical definitions, then we present the five different algorithms with which we try to solve the problem (1.1). The algorithms are path planning methods that provide the agents with positions of where to take probes of f. They rely on a model of the dynamical system, consisting of the vehicle dynamics and the control input defined by the five algorithms. The model assumes some kind of low-level controller that is implemented on the agents themselves which controls their actual motion. For a block diagram of the whole system, the reader is referred to Figure 5.1 in Chapter 5. Furthermore, in the first section a small example is introduced, which is used throughout the thesis to illustrate the functionally of the algorithms.

2.1 Models and Definitions

Not all algorithms require the same model. We make the distinction between model A for the two deterministic algorithms and Stochastic Localization and model B for Metropolis-Hastings and Simulated Annealing.

Model A We consider a moving vehicle on a bounded and convex region $\mathcal{D} \subseteq \mathbb{R}^2$ with unicycle time-discrete dynamics. The state of the vehicle consists of its position $x_k \in \mathcal{D}$ and the heading angle $\theta_k \in [0, 2\pi]$, i.e.

$$s_k = \begin{pmatrix} x_k \\ \theta_k \end{pmatrix}. \tag{2.1}$$

The dynamics of the vehicle are given by

$$\begin{pmatrix} x_{k+1} \\ \theta_{k+1} \end{pmatrix} = \begin{pmatrix} x_k + v(\theta_k)\, T_s \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ u_k \end{pmatrix} + \begin{pmatrix} w_k \\ 0 \end{pmatrix}, \tag{2.2}$$

where

$$v(\theta) = \overline{v} \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \tag{2.3}$$

and $w_k \sim U\{w \in \mathbb{R}^2 \mid \|w\| \leq r_D\}$ with $2\, r_D < \overline{v}\, T_s$ for a constant speed $\overline{v}$ and sampling time T_s and $u_k \in \Psi(s_k) \subseteq [0, 2\pi]$ is the input. In words, the input is the next heading angle. At time k, the next position x_{k+1} is already fixed. By choosing the input, we implicitly calculate the second next position x_{k+2}. This model is used for the algorithms Grid Search, Line Search and Stochastic Localization.

Model B In contrast to model A , the state consist only of the position, $s_k = x_k$ and at time k the next position is fixed. Furthermore, the step length is not constant, that is the sampling time $T_{s,k}$ is time-dependent. The dynamics are

$$x_{k+1} = x_k + v(u_k)\, T_{s,k} + w_k. \tag{2.4}$$

The input must be contained in the set of valid angles, $u_k \in \Omega(s_k) \subseteq [0, 2\pi]$. Model B is used for the algorithms Metropolis-Hastings and Simulated Annealing.

For the time being we assume that the vehicle has a low level controller which realizes every change of orientation fast enough and that moves the vehicle from x_k to x_{k+1}. The factor $\overline{v}\, T_s = l_s$ we refer to as the step length. You might wonder why we need two models. We will see later in this chapter that Metropolis-Hastings and Simulated Annealing propose next positions x_{k+1} depending on some proposal distribution, so it is impossible to know x_{k+2} at time k. Also, since the step length is unknown for fairness we let the velocity constant for all algorithms, we need to adjust the sampling time. Therefore, model A is not applicable to these two algorithms. The velocity $\overline{v}$ does not correspond directly to the actual velocity of the vehicle, which we will see later in Chapter 5. We rather think of the distance from current state to next state (i.e. step length) and the time that is given/required (the sampling time).

A note to the position noise: Although the position noise introduces some stochasticity to Grid Search and Line Search we still consider them as deterministic algorithms, since they do not rely on any other form of randomness. In general we neglect the aspect of the position noise when we describe the algorithms later in this chapter. Whenever it is stated that the vehicle is steered to x_{k+1} or similar, in the simulations we add the random position noise. In the real world implementation we do not need to add artificially the noise, because it will be created by the vehicles movement and the measurement (see Chapter 5).

For the description of the algorithms and the above posted model we need some definitions.

Definition 2.1.1 (State space)**.** *The state space $\mathcal{S} \subset \mathbb{R}^3$ is the extension of $\mathcal{D}$ with the set of valid angles, defined by*

$$\mathcal{S} = \{(x, \theta) \in \mathcal{D} \times [0, 2\pi] \mid \theta \in \Omega(x)\}.$$

Definition 2.1.2 (Set of valid angles)**.** $\Omega(x_k)$ *denotes the subset of* $[0, 2\pi]$ *which ensures that the next position is in $\mathcal{D}$. It is given by*

$$\Omega(x_k) = \{\theta \in [0, 2\pi] \mid x_k + v(\theta)\, T_s \in \mathcal{D}_r\},$$

where $\mathcal{D}_r = \{x \in \mathcal{D} \,\big|\, ||x - y|| \geq r_D,\, y \in \partial\mathcal{D}\}$.

Definition 2.1.3 (Set of proposed angles)**.** $\Psi(s_k)$ *is the subset of angles in* $[0, 2\pi]$ *such that* $x_{k+2} \in \mathcal{D}$. *It is given by*

$$\Psi(s_k) = \{\theta \in [0, 2\pi] \mid x_k + v(\theta_k)T_s + v(\theta)T_s \in \mathcal{D}_{2r}\},$$

where $\mathcal{D}_{2r} = \{x \in \mathcal{D} \,\big|\, ||x - y|| \geq 2\, r_D,\, y \in \partial\mathcal{D}\}$.

Each vehicle is capable of measuring a certain function $c : \mathcal{D} \mapsto \mathbb{R}^+$ at its current position. The function $c(x)$ is called the measurement of $f(x_k)$ at position x. The measurement is also corrupted by noise,

$$\hat{c}(x) = f(x) + \nu, \qquad \nu \sim \mathcal{N}(0, \sigma_\nu). \tag{2.5}$$

From the definition of f, we know that no values smaller than 0 are possible. Thus the sensor cuts off a measurement that is smaller than 0:

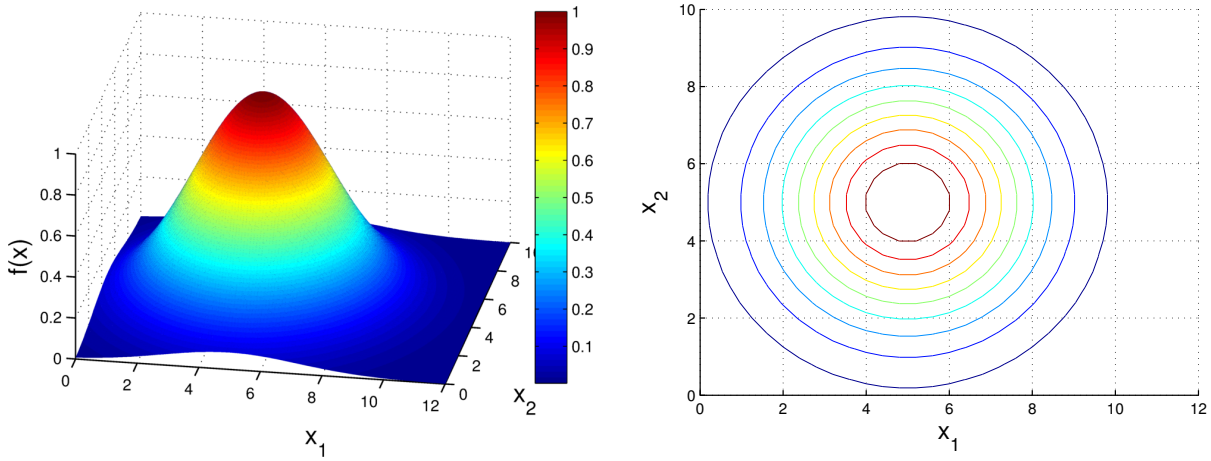
$$c(x) = \begin{cases} \hat{c}(x) & \text{if } \hat{c}(x) \geq 0 \\ 0 & \text{if } \hat{c}(x) < 0 \end{cases} \tag{2.6}$$

We define that the sampling instances of the measurements and of the system coincide. At each instance of time the vehicle takes a measurement of $f(x)$ and calculates the input u_k. Usually the input u_k depends on $c(x_k)$.

To demonstrate how the different algorithms operate we illustrate them with small a example. The region $\mathcal{D}$ and the function f of the example are given below in (2.7) resp. (2.8). A plot of f is drawn in Figure 2.1c and the contour of f in Figure 2.1d. The function is symmetric and has one maximum at (5,5). Noise on the movement and on the measurement is considered as well ($r_D = 0.02$, $\sigma_\nu = 0.02$). We set $\overline{v} = 1\,\mathrm{cm/s}$ and $T_s = 1\,\mathrm{s}$, which yields a step length of 1 cm.

$$\mathcal{D} = [0, 12] \times [0, 10] \tag{2.7}$$

$$f(x) = e^{-0.1\left((x_1-5)^2+(x_2-5)^2\right)} \qquad x \in \mathcal{D} \tag{2.8}$$



(c) 3D plot of $f(x)$ **(d)** Contour plot of $f(x)$

Figure 2.1: Plots of the example function. Although this particular structure is considered unrealistic, because there is only one extremum, it serves best to explain the mode of operation of the algorithms.

2.2 Grid Search

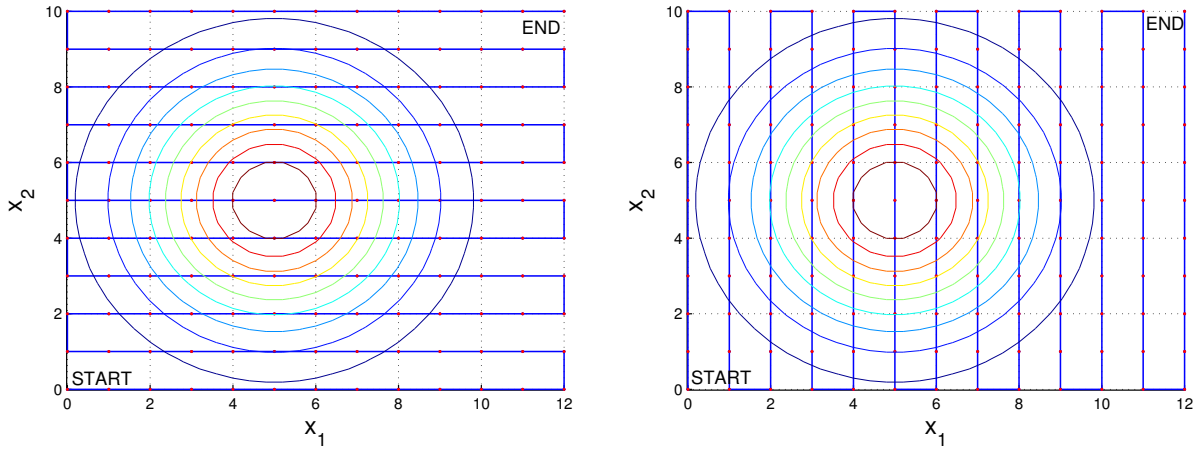
A very simple method is to locate the global maximum of $f(\cdot)$ on a bounded region is to evaluate the values of f at the nodes of a grid over $\mathcal{D}$. Suppose the vehicle starts at the origin with an angle of zero degrees. We let the vehicle drive in the same direction until it reaches the opposite border of $\mathcal{D}$. Then the vehicle takes on step in x_2-direction and turns. Repeat this until the vehicle has covered the whole space. This strategy will visit all nodes specified by the grid and is outlined in Algorithm 2.1. First, a random start state is drawn form the state space, then the vehicle drives to the closest corner. The function $driveTo(s, x, l)$ moves the vehicle from state s to x with step length l and takes measurements of f along its path. Its output is an array $path'$ with the states of the path and the an array $meas$ with the corresponding measurements. The brackets [] represent a concatenation operation.

Algorithm 2.1 Grid Search

Input: $f, \mathcal{D}, g$
Init:
$s \leftarrow random(\mathcal{S})$ `// Initialize randomly over` $\mathcal{S}$
$(path, c) \leftarrow driveTo(s, closestCorner(s), g)$ `// Drive to the closest corner`
Procedure:
while not covered $\mathcal{D}$ **do**
  $(path', meas) \leftarrow driveTo(s, opposite(s, \mathcal{D}), g)$ `// Drive straight on a grid line, turn at the end and take measurements on the nodes`
  $path \leftarrow [path, path']$ `// Save the driven path`
  $c \leftarrow [c, meas]$ `// Save the corresponding measurements`
end while
$s^* \leftarrow \arg\max_{s \in path} c(s)$ `// Find the maximum argument`
return s^*

The only parameter for this algorithm is the grid size g. If we choose the grid size too large, then we do not possess enough nodes to make a valid statement about the maximum, because the maximum may fall between the mesh. On the other hand, if the grid size is small, the vehicle travels a long distance before the algorithm terminates. Of course, with a small grid size we are almost sure to find the maximum. Different grid sizes are realized by adjusting the sampling time T_s since the velocity remains constant. The goal is to find an acceptable grid size such that we find the maximum with decent accuracy and that we do not need to much time. Since we let the vehicle drive with a constant speed, the time consumed is proportional to the distance it travels. Ignoring any noise on the position we can calculate the distance and the number of visited positions before we even start the algorithm. Let us develop the formulas on the example.

Example 2.2.1. *Consider the subspace $\mathcal{D} = [0, 12] \times [0, 10] \subseteq \mathbb{R}^2$ and a grid size $g = 2$. Suppose we start at $s = (0, 0, 0)^T$. This implies that the vehicle drives through the space on horizontal lines as illustrated by Figure 2.2a the resulting path of the vehicle is shown. We see that the vehicle drives the length of the x_1-axis 6 times and one time the length of the x_2-axis. The total distance traveled is thus $L = 11 \cdot 12 + 10 = 142$. Alternatively, if we we start in $s = (0, 0, \pi)^T$ and drive on a vertical lines, it passes the grid as shown in Fig. 2.2b. In this case the vehicle drives a total of $L = 13 \cdot 10 + 12 = 142$. Not surprisingly it seems not to matter in which direction we start.*



(a) Horizontal **(b)** Vertical

Figure 2.2: Two possible paths for grid search. The paths are indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. Also the contour of the concentration field is indicated. The sampling point at the maximum is coincidental.

From the above example we derive the total length[1], starting in x_1 direction, as

$$L_{x_1} = \left(\frac{X_2}{g} + 1\right) X_1 + X_2 = \frac{X_1 X_2}{g} + X_1 + X_2 \tag{2.9}$$

and starting in x_2 direction we have

$$L_{x_2} = \left(\frac{X_1}{g} + 1\right) X_2 + X_1 = \frac{X_1 X_2}{g} + X_1 + X_2 = L_{x_1}.$$

This proves that it does not matter in which direction we start. It even does not matter if we choose to make a mix of vertical and horizontal grid searches, since the amount of nodes is fixed by the grid size as

$$N = \left(\frac{X_1}{g} + 1\right)\left(\frac{X_2}{g} + 1\right) = \frac{X_1 X_2}{g^2} + \frac{X_1 + X_2}{g} + 1. \tag{2.10}$$

The accuracy of the algorithm depends on the visited positions, the larger N is the more accurate the result will be. Fortunately N is inversely proportional to the square of the grid size, leading to the property that we do not have to choose g too small to obtain an accurate result. The point in the middle of four nodes is $g/\sqrt{2}$ away of a node. In other words, with decreasing g we do not gain as much more accuracy in the solution, but we elongate the search time. In the case with position noise, it could happen that global maximum is further away than $g/\sqrt{2}$ from a measuring point. Therefore, g should be chosen a little smaller than in the case without noise.

Since this method lets the vehicle travel through the whole region $\mathcal{D}$ we may travel through large regions that are of little interest, and therefore could be omitted. Obviously, Grid Search suffers from the curse of dimensionality, but the vehicle cannot get stuck in a local maximum. All its nodes to visit are determined before the algorithm starts and do not depend on f. This algorithm gives us thus an upper bound on the search time of the vehicle. Each other search method should have a lower search time than Grid Search.

[1]The grid size should be chosen such that $\frac{X_1}{g}$ and $\frac{X_2}{g}$ yield natural numbers. Else we do not travel through the whole region, or some strip at the border is omitted. We only consider rectangular shapes of $\mathcal{D}$. If $\mathcal{D}$ exhibits a different geometric form, the path needs to be adjusted

2.3 Line Search

We consider a second deterministic method which is called Line Search. It proceeds as follows. Similar to Grid Search, the vehicle maintains a record of where it has traveled and of its measurements. Given an initial state s, the vehicle drives straight and takes measurements every T_s seconds along its path for length l. This trajectory is called a leg L or search axis, where l is called the leg length. The next leg is directed perpendicular to the previous leg at the position of the single leg maximum of the previous leg. After a specified number of legs produced no improvement in the estimate of the global maximum, the search axis is rotated by ϕ, instead of $90°$, and the leg length is shortened by a factor $\gamma \in [0, 1]$. If the new search axis has already been explored (no matter what the initial rotate angle was), the axis is rotated once more by ϕ. The algorithm terminates when the estimate of the maximum has not changed for N consecutive legs or when there is no new search axis around the current leg maximum. Another example of Line Search employed to an optimization problem can be found in [3].

Algorithm 2.2 describes Line Search in pseudo-code. There are four parameters as inputs. Parameter N describes the maximal number of consecutive iterations that did not improve the global solution. If the solution has not changed for N iterations we are almost sure that we have a correct estimate. N needs to be chosen sufficiently large, but not too large to prevent repeated search axes around the global maximum. The second end condition is met if at the current legs maximum the vehicle cannot find an unexplored leg, which is equal to not finding an unvisited starting state (within some tolerance boundaries). An easy routine to check this is implemented in $IsNoNewStart()$ by counting the number of rotations. This counter will be larger than the number of unique rotations N_r in case a new start cannot be found. The starting leg length is defined by the parameter l. It should be large enough to ensure a good coverage of $\mathcal{D}$. ϕ depicts the rotating angle when the solution has not improved for 2 legs. Last, the parameter γ describes the reduction of the leg length, but we set a lower limit to the leg length of three times the step length (i.e. the number of possible reductions by γ is restricted). The function $random(\mathcal{S})$ generates a uniformly distributed initial state. The call of the function $driveLeg(D, l, s)$ in row 7 determines the ends of the new leg, drives the leg and takes measurements of $f(\cdot)$ along its path. The end points of a leg are chosen such that the leg length is l, and if possible that it is centered at the maximum of the previous leg, if this is not possible the leg is shifted. If for some reason, for example the start state is close to two borders in a corner and the angle faced towards a border, the parts of the leg would lie outside $\mathcal{D}$, the leg length is adjusted such that the vehicle does not leave $\mathcal{D}$. If the position and the orientation of the vehicle do not permit that it drives straight for l cm, then the end points lie on the border. $Rotate(D, s, \alpha)$ is the function that rotates the vehicle by α.

Algorithm 2.2 Line Search

Input: $f, \mathcal{D}, N, l, \phi, \gamma$
Init:
$s \leftarrow random(\mathcal{S})$ // Initialize randomly over $\mathcal{S}$
$n, m \leftarrow 0$ // Set counters n, m to zero
$s^* \leftarrow s$
$N_r \leftarrow ceil(\pi/\phi)$ // Maximum number of search axis at one point
Procedure:
while $n \leq N \,\&\, m \leq N_r$ **do**
  $[L, c] \leftarrow driveLeg(\mathcal{D}, l, s)$ // Drive one leg and take measurements
  $\hat{s} \leftarrow \arg\max_{s \in L} c(s)$ // Look for the local maximum over the leg
  $s \leftarrow driveTo(\hat{s})$ // Drive to the local maximum
  if $c(s) \geq c(s^*)$ **then** // If a new global maximum is found
    $s^* \leftarrow s$ // then save that maximum
    $n \leftarrow 0$ // and set the counter to zero
  else
    $n \leftarrow n + 1$ // Increment the counter
  end if
  if $n \mod 2 = 1$ **then** // If the solution has not improved for 2 legs
    $s \leftarrow rotate(\mathcal{D}, s, \phi)$ // iterations, rotate the search axis by ϕ
    $l \leftarrow \gamma l$ // Decrease the leg length
  else
    $s \leftarrow rotate(\mathcal{D}, s, \pi/2)$ // Rotate the search axis by $\pi/2$
  end if
  if $IsNoNewStart(s)$ **then** // Check whether s is a new start
    $l \leftarrow \gamma l$
    $m \leftarrow 0$
    while $IsNoNewStart(s) \,\&\, m \leq N_r$ **do**
      $s \leftarrow rotate(\mathcal{D}, s, \pi/2)$ // Rotate s until s is a new start
      $m \leftarrow m + 1$ // or if not possible end algorithm
    end while
  end if
end while
return s^*

Example 2.3.1. *Let us illustrate Line Search on the example. The parameters are the following,* $N = 2$, $l = 12$, $\phi = \pi/3$, $\gamma = 0.5$. *One possible path is shown in Figure 2.3 below. Please note that the leg length is adjusted to be a multiple of the step length.*

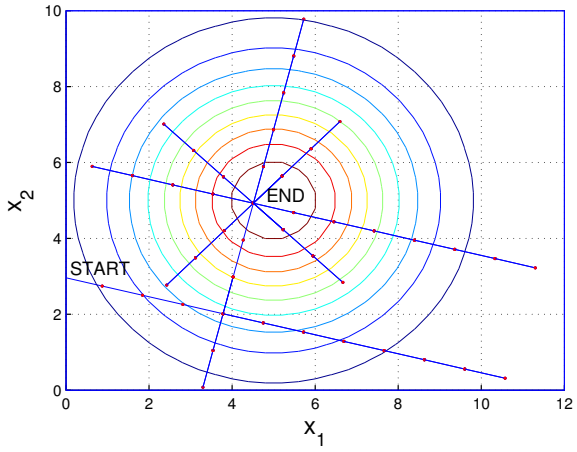


Figure 2.3: Example path for Line Search. The blue straight lines depict the path of the vehicle, the start and end point are marked in black and the red dots represent measurement instances. The concentric circles refer to the contour of f. By rotating the search axis, the precision is increased and by shortening the leg lengths only the regions of interest are explored.

2.4 Markov Chain Monte Carlo

In order to describe the three stochastic algorithms, we introduce in the following the basic concepts and definitions. This section is largely taken from [8, Chp. 4] and [12]. In those papers the interested reader will also find a more detailed approach to this topic.

Markoc Chain Monte Carlo (MCMC) is a method to draw samples from a probability distribution. The basic idea is to iteratively form a Markovian chain $\{X_t\}$, such that for large t X_t is approximately distributed according to the desired distribution π. Then we can use approximate the expected distribution of X given the target distribution π by

$$\mathbb{E}_\pi(X) \approx \frac{1}{N - t_b + 1} \sum_{t=t_b}^{N} X_t, \tag{2.11}$$

where r is the burn-in time. This is the time required until we reach the target density π. For our considerations this parameter has little influence, but to prevent premature termination of Metropolis-Hastings and Stochastic Localization, we set it to t_b=10. The big question is how do we generate the X_t's, i.e. define the Markov Chain? We are looking for a transition rule that tells us how to get to X_{t+1} from X_t. We need X_{t+1} to only depend on X_t and new random variables, but not on X_{t-1}. Let $\mathcal{X}$ be any space with a σ-algebra $\mathcal{F}$. We define the transition kernel P to be

$$P(x, A) = \mathbb{P}(X_{t+1} \in A | X_t = x), \quad \forall\, x \in \mathcal{X}, A \in \mathcal{F}.$$

To describe the used approaches and algorithms, we introduce in the following the basic concepts and definitions.

Definition 2.4.1 (Transition Kernel)**.** *[8, p.42] A transition kernel P of $(\mathcal{X}, \mathcal{F})$ onto itself is a mapping from $\mathcal{X} \times \mathcal{F}$ to [0,1] such that $P(x, \cdot)$ is a probability on $(\mathcal{X}, \mathcal{F})$ for all $x \in \mathcal{X}$ and $P(\cdot, A)$ is a measurable function for all $A \in \mathcal{F}$.*

For the Markov chain we need an initial distribution ν_0. Hence

$$\mathbb{P}(X_0 \in A) = \nu_0(A)$$

and

$$\mathbb{P}(X_{t+1} \in A | X_t = x_t, \ldots, X_0 = x_0) = \mathbb{P}(X_{t+1} \in A | X_t = x) = P(x, A).$$

Definition 2.4.2 (Stationary)**.** *[8, p.43] A probability distribution π on $(\mathcal{X}, \mathcal{F})$ is called stationary or invariant for a transition kernel P, if*

$$\int_{x \in \mathcal{X}} \pi(dx) P(x, dy) = \pi(dy)$$

In shorthand notation stationarity means that $\pi P = \pi$. In other words, an invariant distribution is not changed by the transition kernel P.

Definition 2.4.3 (Reversible)**.** *[8, p.43] A probability distribution π on $(\mathcal{X}, \mathcal{F})$ is called reversible for a transition kernel P, if*

$$\pi(dx) P(x, dy) = \pi(dy) P(y, dx), \quad \text{for } x, y \in \mathcal{X}.$$

This means that the direction of time does not matter, i.e. $(X_0, X_1, \cdots, X_t)$ and $(X_t, X_{t-1}, \cdots, X_0)$ have the same distribution. A very important property of reversibility follows immediately.

Theorem 2.4.1. *[12, Prop. 1] If a Markov chain is reversible w.r.t. π, then π is stationary for the chain.*

Please note that the reverse from Theorem 2.4.1 is not always true.

2.5 Metropolis-Hastings

One way to construct such an MCMC algorithm is the called Metropolis-Hastings algorithm. From [12, Sec. 2.1] we restate the basic methodology. Suppose that $\pi(\cdot)$ has an unnormalized density f satisfying at least $0 < \int_{\mathcal{X}} f(x) < \infty$, moreover we have

$$\pi(A) = \frac{\int_A f(x)\, \mathrm{d}x}{\int_{\mathcal{X}} f(x)\, \mathrm{d}x}.$$

Let $\mathcal{Q}(x, \cdot)$ be any other Markov chain, whose transitions have a density of $\mathcal{Q}(x, dy) \propto q(x, y) dy$. First, choose some initial state X_0. Then, given X_t, draw a proposal state Y_{t+1} from $\mathcal{Q}(X_t, \cdot)$. The proposal is accepted with a probability of $\alpha(X_t, Y_{t+1})$, where

$$\alpha(x, y) = \min\left(1, \frac{f(y)\, q(y, x)}{f(x)\, q(x, y)}\right). \tag{2.12}$$

If the proposal is accepted we set $X_{t+1} = Y_{t+1}$, otherwise the proposal is rejected and we set $X_{t+1} = X_t$. Then increment t and repeat.

Theorem 2.5.1. *[12, Prop. 2] The Metropolis-Hastings algorithm (as described above) creates a Markov chain $\{X_t\}$ which is reversible w.r.t. $\pi(\cdot)$.*

Note that we only need the ratio of $f(x)/f(y)$ to compute α and hence do not need to compute any normalizing constant, which is an advantage. The other big questions are "how should we choose the proposal distribution $\mathcal{Q}(x,\cdot)$?" and "how long would we have to run the chain to obtain $\mathbb{E}(X_t) \approx \pi(\cdot)$?" [12, p. 24] We will use the so called "random walk", where $q(x,y) = q(y,x)$ and $q(x,y) = q(x-y)$. [12, p. 24] Namely we choose $\mathcal{Q}(x,\cdot) = \mathcal{N}(x, \sigma^2 \cdot I_2)$. To answer the above questions exactly lies outside the scope of this thesis. Instead the interested reader is referred to [10], [11], [12], [13], [14], [15]. For the development and an example of the Metropolis-Hastings algorithm in the discrete case the reader is referred to appendix A.

2.5.1 Metropolis-Hastings adapted to optimization problem

Since the original Metropolis-Hastings as proposed by W.K. Hastings [4] is a method to draw samples from a distribution we need to adapt the algorithm to our problem. Remember that we need to optimize f over $\mathcal{D}$ (think of f as an unnormalized density function) and therefore we do not need the exact distribution as the Metropolis-Hastings algorithm would produce. Thus we assume that we do not need to run the algorithm until convergence is reached. Instead we need to find an appropriate end criterion as soon as the global maximum has been found. Therefore we can artificially choose a required number of iterations and mission time, but loose the guaranties coming with the Metropolis-Hastings theory. This increases the risk of producing a false result. Later we will consider two cases of how to terminate Metropolis-Hastings. First, we look at the transient behavior of the chain, that is we terminate it before stationarity is reached (as described above). Second, we set a time limit on the search and neglect the aspect that X_t forms a converging Markov chain.

Remember that we use model B for Metropolis-Hastings. Furthermore, for the first case from above, we need to divide the space $\mathcal{D}$ into equally shaped squares, whose size is determined by the bin size b. Then, we count how often the vehicle finds itself in a given bin. The maximum of f lies then in the bin with the highest count.

The algorithm starts at a random position. We propose a new position $\hat{x}$ according to the proposal distribution $\mathcal{Q}(x,\cdot)$. We take $\mathcal{Q}(x,\cdot)$ to be a bivariate normal distribution with a given variance σ^2,

$$q(x,\hat{x}) \propto \begin{cases} \exp\left(-\frac{1}{2\sigma^2}(\hat{x}-x)^T(\hat{x}-x)\right), & \text{if } \hat{x} \in \mathcal{D} \\ 0 & \text{else.} \end{cases}$$

The vehicle drives to $\hat{x}$ and evaluates the acceptance criterion (2.12), which simplifies to

$$\alpha(x,\hat{x}) = \min\left(1, \frac{c(\hat{x})}{c(x)}\right)$$

due to the symmetric proposal distribution ($q(x,\hat{x}) = q(\hat{x},x)$). The factor α represents the probability of accepting the proposed state. If the value of c at the proposed position is higher than at the current position, then α equals 1 and the proposed position is always accepted. In the other case when $c(\hat{x}) \leq c(x)$, it is still possible to accept the proposal with probability α. By accepting unfavorable steps the algorithm is able to escape local minima. In the implementation we draw a sample $u \sim \mathcal{U}(0,1)$. If $u \leq \alpha$ we accept $\hat{x}$, otherwise we reject $\hat{x}$ and drive back to x. To keep track which positions the vehicle has visited we introduce the matrix $V \in \mathbb{R}^{m \times n}$. The elements of V represent how often the vehicle has visited a position in the respective bin. The dimension of V depends on the chosen bin size and the dimension of $\mathcal{D}$. Furthermore, V

is normed such that the sum of all its elements add up to 1. If we accept the proposed position we update the bin in which $\hat{x}$ lies. If we reject $\hat{x}$ we increase the counter of the bin in which x is located. The matrix V can be regarded as the current normalized probability distribution of the chain. For the termination we compute

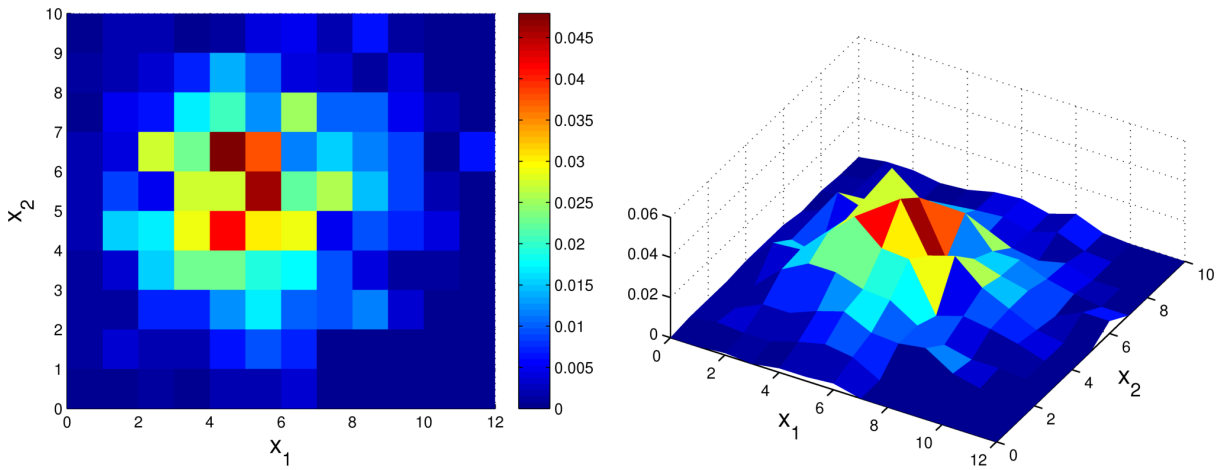
$$\Delta = ||V_{k+1} - V_k||_1 \leq \epsilon.$$

If Δ reaches a certain value ϵ we assume that the algorithm has found the global maximum and the algorithm terminates. In the end we just need to find the maximum value of V_k, i.e. the bin with he most visits. The maximum of c (and hopefully also of f) lies then in this bin. Choosing an appropriate bin size is critical. If b is too large, the result is very inaccurate and not useful. Think of having only one bin as large as $\mathcal{D}$. Running the algorithm would not give us any more information, since we already know that there is a maximum in $\mathcal{D}$. On the other hand, if we choose b very small, we would have not converged to the global maximum in time and hence need to reduce ϵ, which is equal to increase the search time. But of course once V has converged correctly, we have a very accurate result. We describe our implementation of Metropolis-Hastings in Algorithm 2.3 below. The additional function $driveTo(y)$ moves the vehicle from the current position to position y according to model B . It also chooses the sampling time for that step. The function $update(V, b, x)$ adds 1 to the bin in which x lies and normalizes V.

Algorithm 2.3 Metropolis-Hastings

Input: $f, \mathcal{D}, \mathcal{Q}, \epsilon, b$
Init:
$x_0 \leftarrow random(\mathcal{D})$ `// Initialize randomly over` $\mathcal{D}$
$\Delta \leftarrow 1$
$V_0 \leftarrow matrix(\mathcal{D}, b)$ `// Allocate matrix over` $\mathcal{D}$ `with binsize b`
Procedure:
while $\Delta \geq \epsilon$ **do**
    $\hat{x} \sim driveTo(\mathcal{Q}(x))$
    $\alpha \leftarrow \min\left(1, \frac{c(\hat{x})}{c(x)}\right)$ `// Measure` f `and evaluate` α
    if $u \sim \mathcal{U}(u \in [0,1]) \leq \alpha$ **then** `// Accept`
        $x_{k+1} \leftarrow \hat{x}$
    else `// Reject`
        $x_{k+1} \leftarrow driveTo(x_k)$ `// Drive back to original position`
    end if
    $V_{k+1} \leftarrow update(V_k, b, x_{k+1})$ `// Count the position to the respective bin`
    $\Delta \leftarrow \|V_{k+1} - V_k\|_1$ `// Calculate the relative difference`
    $k \leftarrow k + 1$
end while
$A \leftarrow \arg\max_{bin \in D} V_k$ `// Calculate the bin with the most visits`
return A

Example 2.5.1. *Let us continue the example. Instead of plotting the path the vehicle drove we show the matrix V after termination. Thus we should only have one highest bin at (5,5).* $\mathcal{Q}(s, \cdot) = \mathcal{N}(s, 10 \cdot I_2)$, $\epsilon = 0.001$, $b = 1$.



(a) Top view **(b)** 3D view

Figure 2.4: The normalized distribution V after termination of Metropolis-Hastings. The maximum bin lies at (5,5).

2.6 Simulated Annealing

Simulated Annealing originates from annealing in metallurgy. To obtain a material with minimal crystal lattice defects and relatively large crystals the technique of annealing is used. First, the material is heated to allow the atoms to diffuse more easily and to remove dislocations. Once an equilibrium state has reached, the temperature is decreased. The reducing needs to happen at a slow pace such that at each temperature the atoms have enough time to reach the optimal equilibrium again. If the temperature decreases to fast, the atoms might get trapped in a suboptimal equilibrium. Simulated Annealing takes up this idea and adapts it to mathematical optimization. The method was first published by S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi in 1983 [5].

As for the Metropolis-Hastings algorithm, we apply model B . The vehicle is able to measure $c(x_k)$ at some position x_k, then calculate the next position and drive to it. Also, the step length is not constant, but for comparability the velocity still is. The algorithm starts with a random position uniformly distributed over $\mathcal{D}$. At the beginning of each iteration, a new state is proposed according to the density $\mathcal{Q}(x)$. Again we could choose $\mathcal{Q}(x)$ to be a bivariate normal distribution, but we decided for a circle of given radius R with a Gaussian angle distribution around the current orientation. The mean of the angle is either directed towards the last state, if $c(x_{k-1}) > c_(x_k)$ or in the opposite direction if $c(x_k) > c(x_{k-1})$. Its variance σ_θ^2 is a parameter and left as a choice to the designer. Later we will also refer to the angle distribution as bias on the angle, in contrast to a uniform distribution. For the two first of steps, the angle distribution is uniform. Furthermore, states outside $\mathcal{D}$ have zero probability. Following the idea of MCMC, the proposed position $\hat{x}$ is either accepted or rejected. If $c(\hat{x})$ is higher than $c(x)$, the proposed state is accepted and a new iteration starts. Otherwise we only accept the proposal with probability

$$\alpha(x, \hat{x}) = \min\left(1, e^{\frac{c(\hat{x})-c(x)}{T}}\right) \leq 1,$$

where the temperature T is a tuning parameter. This particular form of the metropolis criteria is motivated by the thermodynamic theory. Similar to Metropolis-Hastings, the vehicle needs to drive to $\hat{x}$ in order to evaluate $\alpha(x, \hat{x})$. If the proposal is rejected it drives back to x and a new iteration starts. Accepting unfavorable states enables the algorithm to escape a local maximums without the need of restarting the search.

The choice of T is important for the balance of exploration and exploitation. Is the temperature high we are more likely to accept disadvantageous states and hence search through a larger area and could still evade a local maximum. While the temperature decreases such unfavorable positions become less likely and getting trapped in a local extremum becomes and issue. The assumption is though that when this effect grows stronger, we find ourselves already in the region of the global maximum (which we do not want to leave anymore). Several methods on when and how fast to decrease the temperature exist, e.g. [5], [6] or [7]. One could specify a given number of drawn proposals at each temperature, one could count the iterations of not improving the solution or one could let the vehicle drive a certain distance at each temperature. We select the first option and specify a number of drawn proposals at each temperature, N_t. After one temperature period the temperature decreases with a factor $\gamma \in [0, 1]$. In addition, we also let the radius of the proposed states, R, decrease at the same instance but with a different rate $\eta \in [0, 1]$. The radius has a lower limit $R_{min} = const$ to force the algorithm to terminate. The algorithm terminates when N_r consecutive proposals were rejected. Usually this condition is met when the temperature is low. Alternatively, we could also terminate the algorithm after a fixed number of temperature periods. The Simulated Annealing algorithm is described in pseudo code in Algorithm 2.4.

Algorithm 2.4 Simulated Annealing

Input: $f, \mathcal{D}, N_t, \mathcal{Q}, T_0, \gamma, R_0, \eta, N_r, \sigma_\theta$
Init:
$x_0 \leftarrow random(\mathcal{D})$ // Initialize randomly over $\mathcal{D}$
$j, k, n \leftarrow 0$ // Initialize counters
$x^* \leftarrow s_0$ // Initialize solution with x_0
Procedure:
while $n \leq N_r$ **do**
  for $i = 1 \rightarrow N_t$ **do**
    $\hat{x} \sim driveTo(\mathcal{Q}(R_j, x_k, x_{k-1}, c(x_k), c(x_{k-1}), \sigma_\theta))$ // Propose a new position
    // and drive to it
    $\alpha \leftarrow \min\left(1, e^{\frac{c(\hat{x})-c(x)}{T_j}}\right)$ // Measure f and evaluate α
    if $u \sim \mathcal{U}(u \in [0, 1]) \leq \alpha$ **then** // Accept
      $x_{k+1} \leftarrow \hat{x}$
      $n \leftarrow 0$
    else // Reject
      $x_{k+1} \leftarrow driveTo(x_k)$ // Drive back to x_k
    end if
    if $c(x_{k+1}) \geq c(x^*)$ **then**
      $x^* \leftarrow x_{k+1}$ // Keep track of the maximum
    end if
    $k \leftarrow k + 1$
  end for
  $T_{j+1} \leftarrow \gamma \cdot T_j$ // Reduce the temperature
  $R_{j+1} \leftarrow \eta \cdot R_j$ // Reduce the step length
  if $R_{j+1} < R_{min}$ **then**
    $R_{j+1} \leftarrow R_{min}$
  end if
  $j \leftarrow j + 1$
end while
return x^*

Example 2.6.1. *We continue the example. Figure 2.5 below shows one sample path. The initial position is drawn randomly and lies far away from the maximum. The path shows all the states that the vehicle drives to, including the ones that are rejected. The vehicle has to drive to the proposed positions in order to measure f, where it evaluates the metropolis criterion. If that state is rejected the vehicle needs to drive back to the original position. The parameters are set to $N_t = 4$, $T_0 = 1$, $\gamma = 0.8$, $R_0 = 6$, $\eta = 0.8, \sigma_\theta = 0.75$. In the paper of E. Burian e al. [3] another example of Simulated Annealing applied to a similar problem is given.*

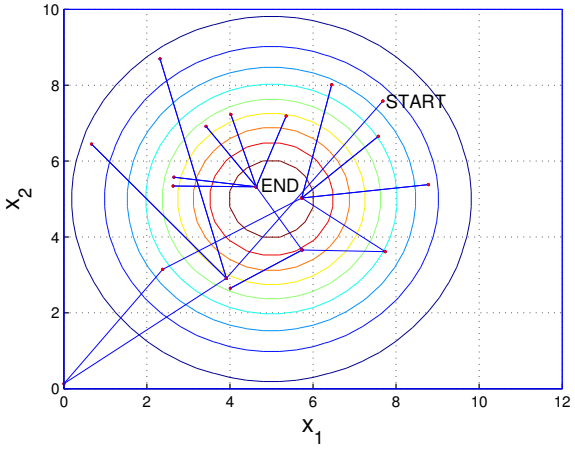


Figure 2.5: Example path for Simulated Annealing. The path is indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. Also the contour of the concentration field is indicated. Note that also the rejected positions belong to the path of the vehicle.

Selecting the parameter correctly is crucial. Since we do not have any a priori information about the shape of f we run into troubles. If f is bowl shaped with one maximum as in the example, the parameters have little influence on the success rate. Only the drive time could be optimized. On the other range of possible functions of f are functions with multiple extrema and large shallow regions. In addition, more problems occur if the extrema have steep gradients around them. Parameter selection will be discussed in Section 3.1.4.

In contrast to Metropolis-Hastings, we do not bin the space. Simulated Annealing has the property that the vehicle converges to a maximum, where it will most probably stay because T is very low. Thus we can look directly at the path of the vehicle, rather than at the distribution.[2]

[2]This we will do too for Metropolis-Hastings when we set a time limit.

2.7 Stochastic Localization

We investigate on a third MCMC method. The newly proposed method in [1] and [2] is analyzed in this section.

The algorithm is initialized by extracting s_0 uniformly on $\mathcal{S}$, that is $s_0 \sim \mathcal{U}\{s \in \mathcal{S}\}$. At each time step $k = 1, 2, \ldots$ we propose an angle $u_k \sim \mathcal{U}(\theta \in \Psi(s_k)))$ which is accepted with probability $\alpha(s_k)$. If the proposal is not accepted, then $u_k = \theta_k$,

$$u_k = \begin{cases} \theta_{k+1} \sim \mathcal{U}(\theta \in \Psi(s_k)) & \text{w.p. } \alpha(s_k) \\ \theta_k & \text{w.p. } 1 - \alpha(s_k). \end{cases} \tag{2.13}$$

The function $\alpha(\cdot) : \mathcal{S} \mapsto [0, 1]$ is the acceptance criterion. It can be any function with the following three properties, (see [1] for details):

1. $\alpha(\cdot)$ is monotonically increasing in $c(\cdot)$.
2. $\alpha(\cdot)$ depends continuously on s.
3. $\alpha(s_k) = 1$ if $\theta_k \notin \Psi(s_k)$.

For the sake of simplicity we choose

$$\alpha(s_k) = \begin{cases} 1, & \text{if } \theta_k \notin \Psi(s_k) \\ 1 - e^{-(Kc(x_k))^J}, & \text{otherwise.} \end{cases} \tag{2.14}$$

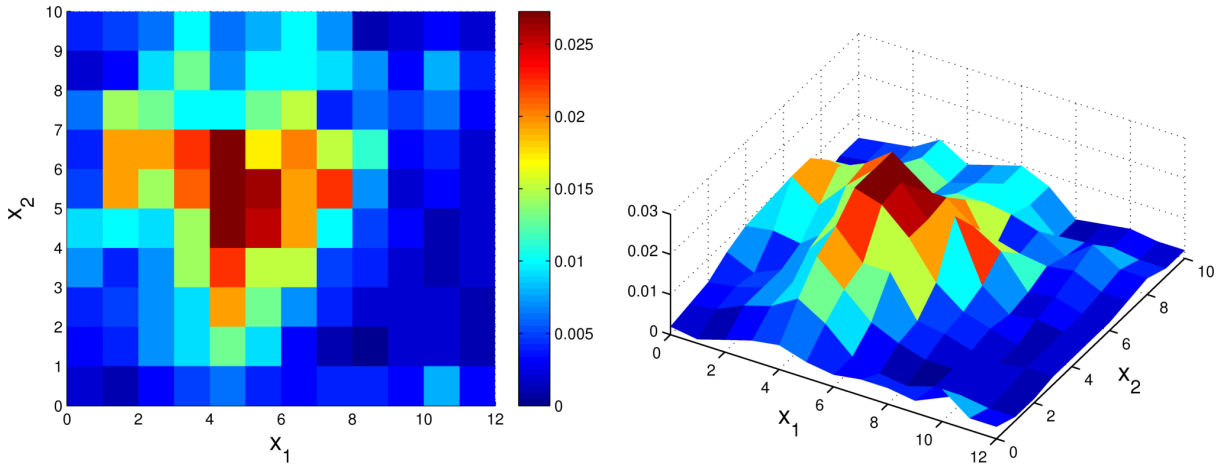
Please note that α theoretically needs to be continuous, but our simulations show that Stochastic Localization still converges with this particular non-continuous criterion. The criterion (2.14) is non-continuous because close the the border it exhibits jumps to 1 stated in the first line. The function $\alpha(\cdot)$ has two parameters J and K. The tuning of those parameters will be addressed in Section 3.1.5. The closed loop motion of the system with the control (2.13) generate a discrete Markov chain $\{s_k\}_{k \geq 0}$. If we were to run this chain until it has converged, we would obtain an estimation of f [2]. But this is not our actual goal. We need to think about a stopping condition that reflects the believe in having found a global maximum. Similar as for the Metropolis-Hastings, we divide the space $\mathcal{D}$ into smaller bins. Again, we count how often the vehicle finds itself in a given bin. If $c(x)$ is high at some point, the likeliness of the vehicle to turn is higher. Hence the vehicle resides more often in regions with a high $c(x)$ than in regions with a low $c(x)$. We also have a bin size b and a stop criterion ϵ and we are looking at the count of the bins, not the exact path of the vehicle. We use the same stop criterion as for Metropolis-Hastings given in equation 2.5.1. The algorithm Stochastic Localization is outlined in Algorithm 2.5.

Algorithm 2.5 Stochastic Localization

Input: $f, \mathcal{D}, J, K, b, \epsilon$
Init:
$s_0 \leftarrow random(\mathcal{S})$ // Initialize randomly over $\mathcal{S}$
$\Delta \leftarrow 1$
$V_0 \leftarrow matrix(\mathcal{D}, b)$ // Allocate matrix over $\mathcal{D}$ with binsize b
Procedure:
while $\Delta \geq \epsilon$ **do**
  $\hat{\theta} \sim \mathcal{U}(\theta \in \Psi(s_k))$ // Draw angle proposal
  $\alpha \leftarrow \begin{cases} 1 & \text{if } \theta_k \notin \Psi(x_{k+2}) \\ 1 - e^{-(Kc(x_k))^J} & \text{otherwise} \end{cases}$
  if $u \sim \mathcal{U}(u \in [0,1]) \leq \alpha$ **then** // Accept
    $\theta_{k+1} \leftarrow \hat{\theta}$
  else // Reject
    $\theta_{k+1} \leftarrow \theta_k$
  end if
  $s_{k+1} \leftarrow driveTo((x_k + v(\theta_k)T_s, \theta_{k+1})^T)$ // Drive to next state
  $V_{k+1} \leftarrow update(V_k, b, s_{k+1})$ // Count the position to the respective bin
  $\Delta \leftarrow \|V_{k+1} - V_k\|_1$ // Calculate the relative difference
  $k \leftarrow k + 1$
end while
$A \leftarrow \arg\max_{bin \in D} V_k$ // Calculate the bin with the most visits
return A

The result of the algorithm is a bin A which has the most visits. The maximum lies in this bin, thus the bin size is an important parameter for the accuracy. If we choose it too high, we are not able to obtain an accurate result. However, if the bin size is small, the result is very sensitive and we might obtain an incorrect result.

Example 2.7.1. *For the example we choose the parameters as* $J = 2.8, K = 3.25, \epsilon = 0.001, b = 1$. *Figure 2.6 depicts the matrix* V *after termination.*



(a) Top view **(b)** 3D view

Figure 2.6: The normalized distribution V after termination of Stochastic Localization The maximum bin lies at (4,5).

Chapter 3

Simulation and Comparison

In this chapter we present the findings of our simulations. We investigate on the performance and robustness of the algorithms.

The main performance criteria to compare the algorithms is the success rate $\overline{r}$. A second criteria is the mission time which is relevant if we let the algorithms terminate with their given terminal condition (see Chapter 2.1). In Section 3.4 we set a limit on the mission time, therefore the total mission time becomes irrelevant. This will be especially important for Metropolis-Hastings and Stochastic Localization, because in the latter case we neglect that they form Markov chains that converge to a distribution. We will directly look at the measured concentrations as we do for the other three algorithms.

We assume that the vehicle drives with a constant speed of $\overline{v} = 10\,\text{cm/s}$. As mentioned in Chapter 2, Metropolis-Hastings and Simulated Annealing operate with model B (2.4), whereas Grid Search, Line Search and Stochastic Localization operate with model A (2.2). Since the algorithms are not purely deterministic, either by their construction or because of the position and measurement noise, there is a probability that the result is incorrect. We thus define the success rate as follows. Let r_i be the indicator whether the i-th execution of an algorithm was successful ($r_i = 1$) or not ($r_i = 0$). If the estimated maximum x_i^* is within a radius of 10 cm to the global maximum x_{max}, the run is considered as successful. Then we calculate

$$\overline{r} = \frac{1}{N} \sum_{i=1}^{N} r_i \tag{3.1}$$

with the indicator

$$r_i = \begin{cases} 1, & \text{if} \quad \|x_i^* - x_{max}\| \leq 10\,\text{cm} \\ 0, & \text{else} \end{cases} \tag{3.2}$$

for Grid Search, Line Search and Simulated Annealing and

$$r_i = \begin{cases} 1, & \text{if} \quad \|x_i^* - x_{max}\| \leq \max\left(\sqrt{2} \cdot b, 10\right) \\ 0, & \text{else} \end{cases} \tag{3.3}$$

for Metropolis-Hastings and Stochastic Localization with bin size b. The different criteria for Metropolis-Hastings and Stochastic Localization accounts for the fact that the space is binned.

Let us now define the mission time. Let t_i be the time (in seconds) the vehicle drove during the i-th run. Then the t_i's are samples of the random variable M with unknown distribution.

We can define the average mission time

$$E[M] = \bar{t} = \frac{1}{N} \sum_{i=1}^{N} t_i. \tag{3.4}$$

The distribution of the mission time for TF1 is denoted as M_1 and for TF2 M_2 respectively. Furthermore we define D_1 and D_2 to be the distances from the estimated maximum to the true global maximum for TF1 respectively TF2. Ideally the success rate (3.1) is equal to 1 and the mission time as low as possible.

For the initial simulations we choose a position noise of $r_D = 0.2\,\text{cm}$ and a measurement noise of $\sigma_\nu = 0.02$ and we will use these values if not stated otherwise. We will later change this setting and test algorithms for robustness with different values of r_D and σ_ν. Although we utilize several vehicles for Stochastic Localization in later chapters for comparability we use only one vehicle for all algorithms in this chapter.

3.1 Parameter Selection and Simulation

In this section we choose two test functions, which we use to select acceptable parameters for the upcoming comparisons. Then we compare the behavior in detail of the algorithms under normal conditions ($r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$). The robot test bed, which is used later for experimental validation, described in Chapter 4, has an approximate dimension of $300\,\text{cm} \times 250\,\text{cm}$. Thus we use the same dimension also in the simulations,

$$\mathcal{D} = [0, 300] \times [0, 250]. \tag{3.5}$$

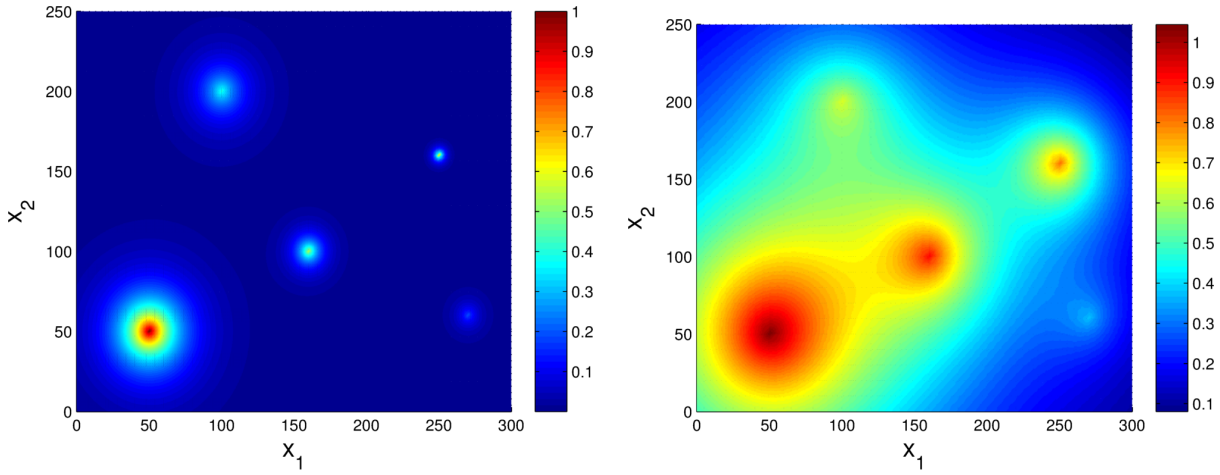
As test functions we choose exponential fields. Alternately, one could choose functions with quadratic decay to model an electromagnetic signal for instance. They are given by

$$f_k(x) = \sum_{i=1}^{5} c_i e^{-\lambda_{ki}\sqrt{(x_1 - X_i)^2 + (x_2 - Y_i)^2}} \qquad \text{for } x \in \mathcal{D},\, k = 1, 2\,, \tag{3.6}$$

where

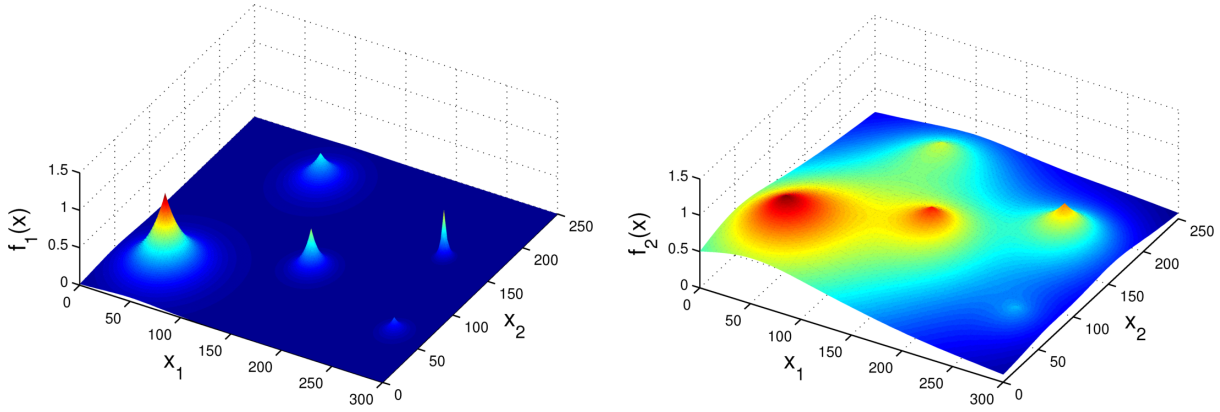
$$\begin{gathered} c_1 = 1,\, c_2 = 0.4,\, c_3 = 0.55,\, c_4 = 0.65,\, c_5 = 0.2 \\ X_1 = 50,\, X_2 = 100,\, X_3 = 160,\, X_4 = 250,\, X_5 = 270 \\ Y_1 = 50,\, Y_2 = 200,\, Y_3 = 100,\, Y_4 = 160,\, Y_5 = 60 \\ \lambda_{11} = 0.06,\, \lambda_{12} = 0.07,\, \lambda_{13} = 0.13,\, \lambda_{14} = 0.35,\, \lambda_{15} = 0.145 \\ \lambda_{21} = 0.01,\, \lambda_{22} = 0.02,\, \lambda_{23} = 0.025,\, \lambda_{24} = 0.3,\, \lambda_{25} = 0.04. \end{gathered}$$

The two functions differ only in their λ_i parameters. The graph of function $f_1(\cdot)$ (also called test field 1 or TF1) has steep peaks while $f_2(\cdot)$ (also called test field 2 or TF2) has a more gentle slope. The global maximum is at $(50, 50)^T$. Both graphs are plotted in Fig. 3.1 and Fig. 3.2 in a 2D and 3D view respectively.



(a) Test field 1, $f_1(x)$ **(b)** Test field 2, $f_2(x)$

Figure 3.1: Top view of the two test fields. The peaks in TF1 are steeper than in TF2.



(a) Test field 1, $f_1(x)$ **(b)** Test field 2, $f_2(x)$

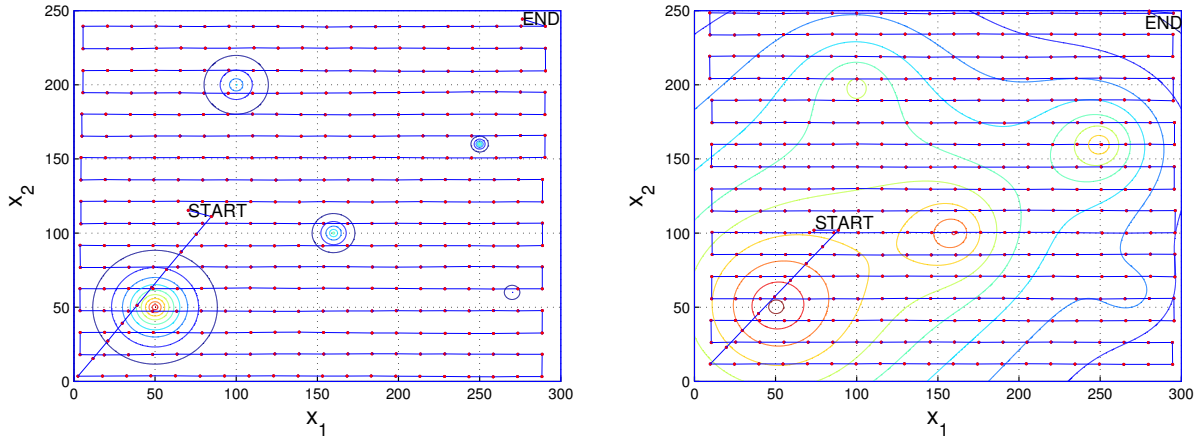
Figure 3.2: 3D view of the two test fields

Most algorithms have one or more parameters. Their performance depends on the proper selection of their respective parameters. We try to estimate a set of parameters with good performance of each algorithm to have a fair comparison. However, selecting the parameters is usually a trade-off between mission time and success rate and thus dependent on the designers preferences.[1] The parameters we present are also a trade-off between the two fields. For each field, there exists a better parameter set, but we select the parameters such that the performance is good (or at least not bad) for both fields. This aspect reflects the assumption of the field to be unknown.

[1]The selection of the parameters is itself an optimization problem, but we will not further pursue this trail.

3.1.1 Grid Search

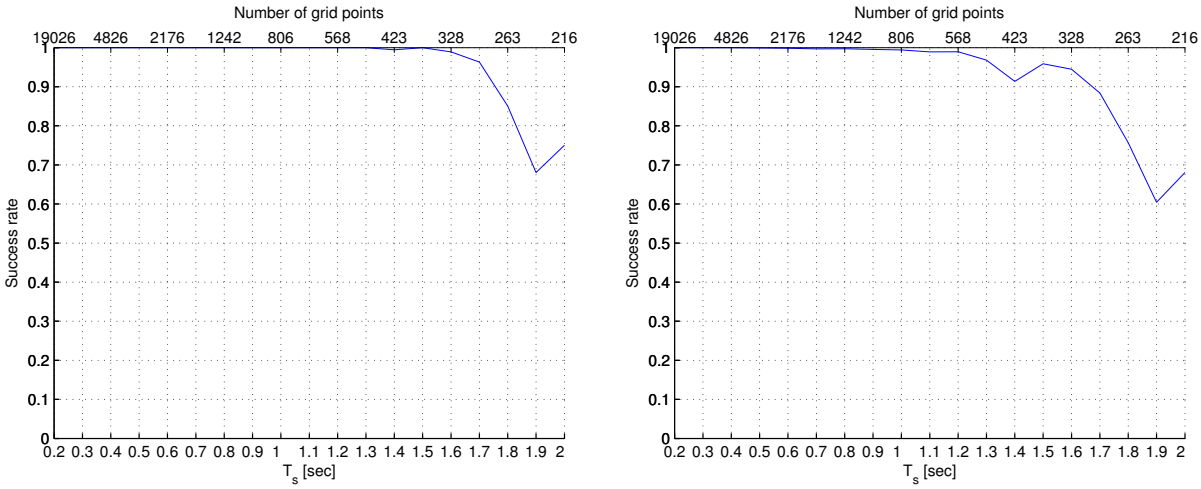
A small remark at the beginning. The starting point of this algorithm is always a corner. It does not make sense to drive into the search region, just to drive back to the border. But for the sake of completeness, we let Grid Search, like any other algorithm, start randomly in $\mathcal{D}$. Then the vehicle drives to the closest corner. A sample path for each test field is depicted in the Figure 3.3 below.



(a) TF1 (b) TF2

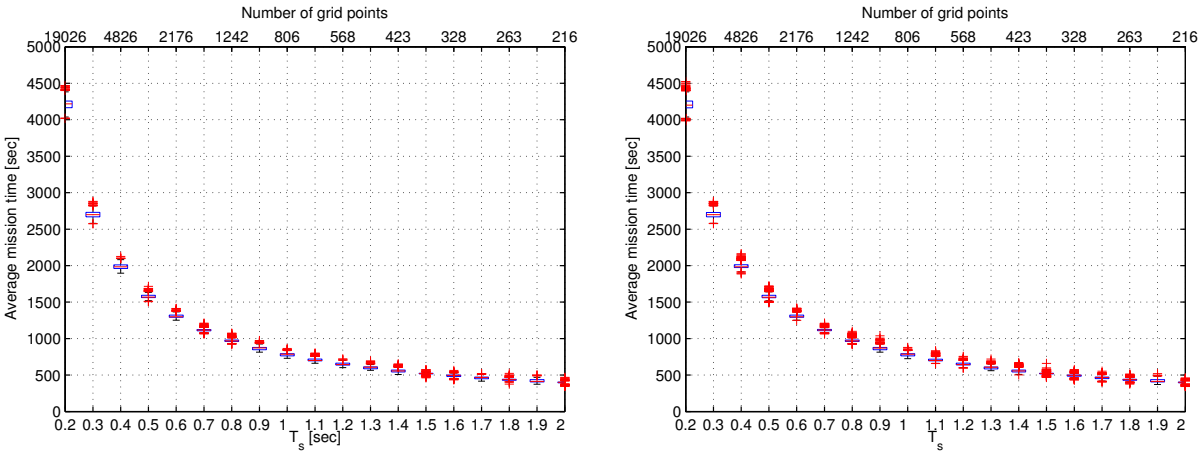
Figure 3.3: Simulated sample paths of Grid Search. The vehicles path is indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. Also the contour of the concentration field is indicated.

The only parameter Grid Search possesses is the grid size. Since the velocity is constant, we adapt the sampling time according to the relation $g = \overline{v}\, T_s$. Selecting an appropriate sampling time is easy. We just simulate the algorithm with different sampling times ($T_s = 0.2\,\mathrm{s}$–$2\,\mathrm{s}$) for $N = 1000$ times. The resulting success rates (Figure 3.4) and mission times (Figure 3.5) are shown below. Since in this case the mission time is linear correlated to the total length of the path, the expected mission time shows same behavior as the length with respect to the grid size given by (2.9).



(a) TF1 **(b)** TF2

Figure 3.4: Success rates for both test fields. In addition to the sampling time on the lower x-axis, the corresponding number of grid points is shown in the top x-axis.



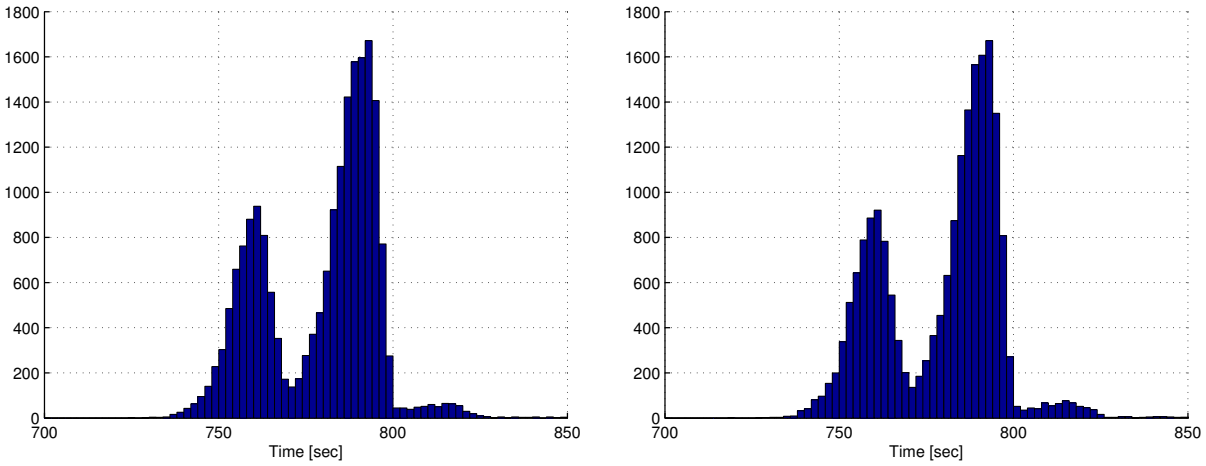
(a) TF1 **(b)** TF2

Figure 3.5: Mission time of Grid Search. The medians of each group are marked by the red line inside the boxes. The edges are the 25th and 75th percentiles, the whiskers extend to the most extreme data points, while outliers are plotted in a red cross. In addition to the sampling time on the lower x-axis, the corresponding number of grid points is shown in the top x-axis.

For a grid size larger than 13 cm, the success rate starts to drop, for the test field 2 even a little before. Of course we could choose very small grid size equal to 2, but we do not gain much except a larger mission time. Thus we decided for a grid size of 10 cm ($T_s = 1$ s). For fairness we must point that these results are expected since we defined the success rate with a distance of 10 cm

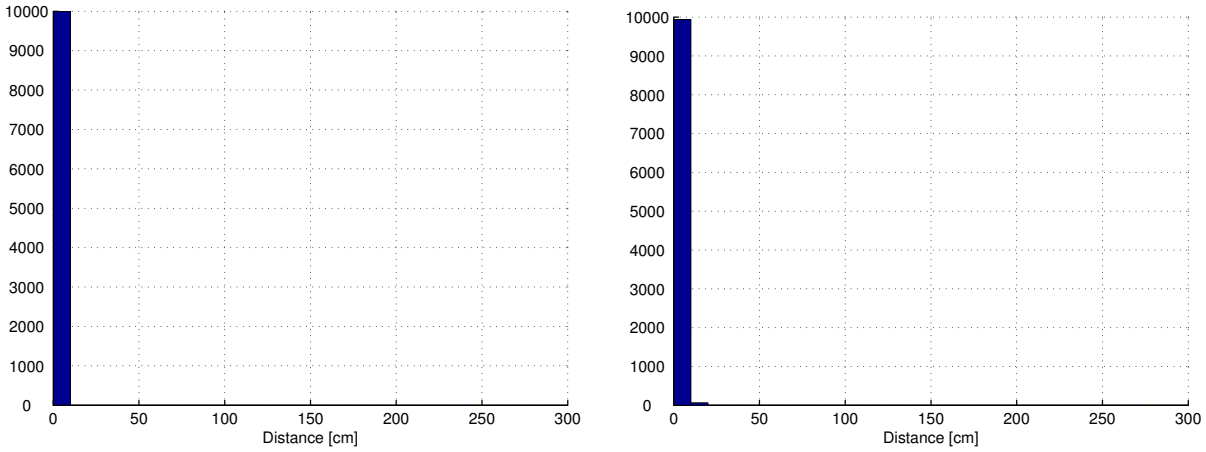
For the chosen parameters we run the algorithm for $N = 10\,000$ instances and take a look at the distribution of the mission time (Figure 3.6) and the distribution of the distance from the found maximum to the global maximum (Figure 3.7). As expected the difference between M_1 and M_2 and D_1 and D_2 are minimal. This is logical since Grid Search is independent of $f(\cdot)$. The two peaks in M_1 and M_2 result because the vehicle does not drive the same number of lines

each time, because of the random start position and the noise on the position. In Table 3.1 the success rates and average mission times are summarized. The grid size can always be chosen such that the success rate is 100 % (or very close), on the cost of mission time.



(a) M_1 **(b)** M_2

Figure 3.6: Distribution of the mission time ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$). The two peaks exist because the line number depends on the start position and on the position noise.



(a) D_1 **(b)** D_2

Figure 3.7: Distance from estimated to global maximum ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$). The first bar from 0 to 10 cm represents the success rate.

	TF1	TF2
Success rate	100 %	99.4 %
Average mission time	779 s	779 s

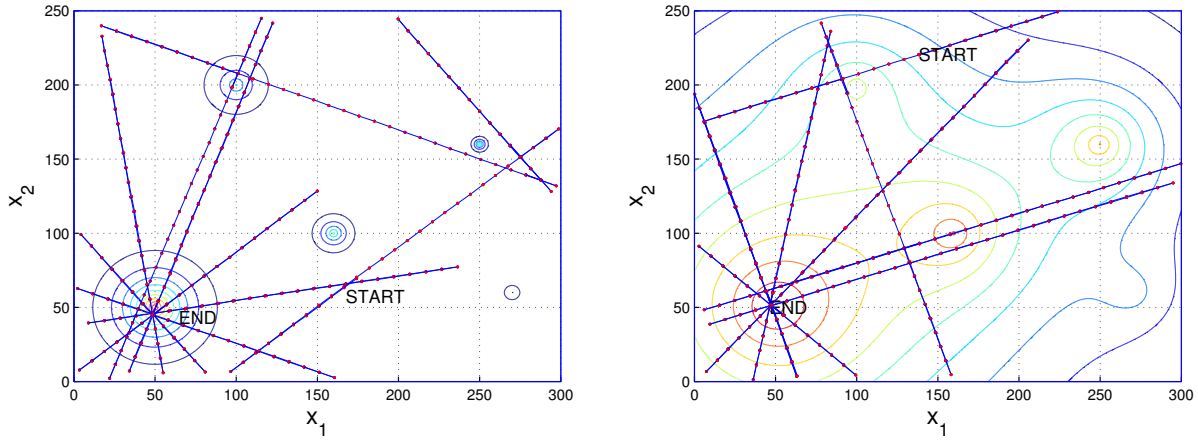
Table 3.1: Success rate and average mission time for Grid Search.

3.1.2 Line Search

We will not discuss parameter selection in full detail, but a few things have to be noted. The leg length should at least be the diagonal of the space, to give the vehicle a chance to travel through the whole space at the beginning. Parameter N has little influence if chosen above 3 since the algorithms normally terminates when there is no new search axis left. If γ is chosen large, the mission time is elongated, but the success rate is also slightly increased. The turning angle ϕ shows best results for $\phi \in [\pi/8, \pi/3]$. We choose the following parameters:

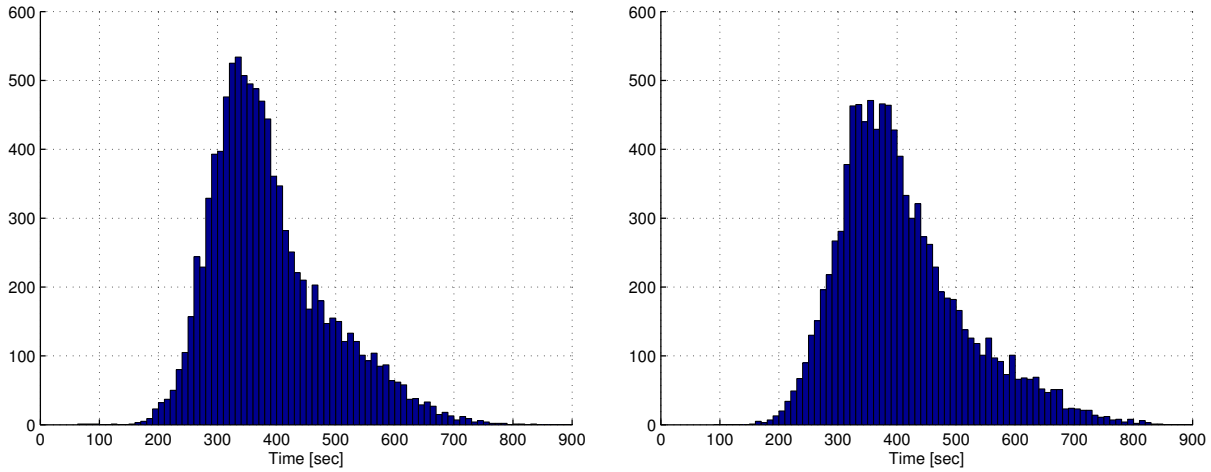
$$l = 380\,\text{cm},\ \gamma = 0.8,\ N = 4,\ \phi = \pi/6. \tag{3.7}$$

As for grid search, we selected the step length to be 10 cm (i.e. $\overline{v} = 10\,\text{cm/s}$, $T_s = 1\,\text{s}$). Figure 3.8 below deptics one sample path for each function, Figure 3.9 shows the mission times for TF1 and TF2 and the distances in Figure 3.10. From Table 3.2 we note that Line Search shows better performance for test field 2 than test field 1, which is due to the fact that the global maximum has a larger zone of attraction.



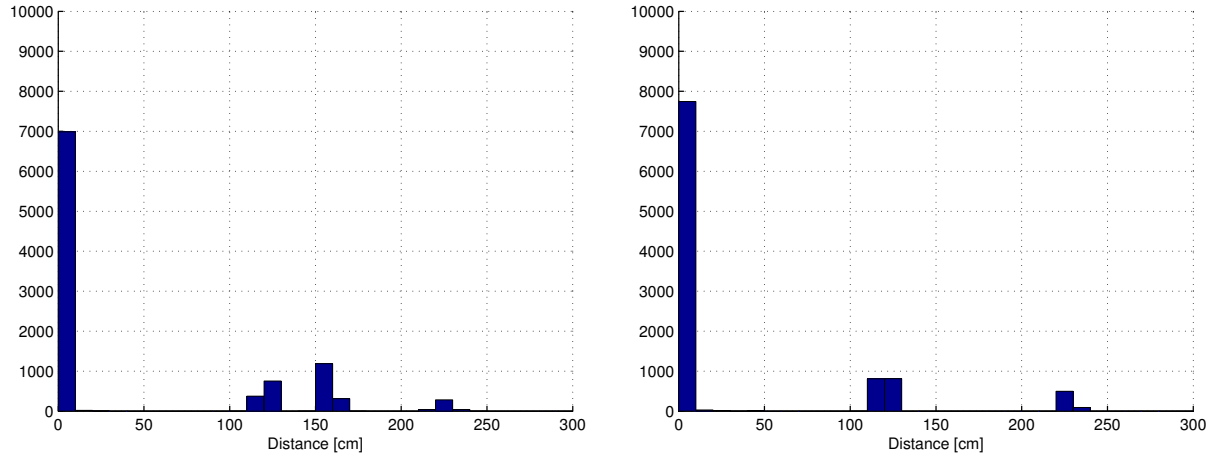
(a) TF1 **(b)** TF2

Figure 3.8: Simulated sample paths of Line Search. The vehicles path is indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. Also the contour of the concentration field is indicated.



(a) M_1 **(b)** M_2

Figure 3.9: Distribution of the mission time ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$).



(a) D_1 **(b)** D_2

Figure 3.10: Distance from estimated to global maximum ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$). The clusters at 120, 160 and 230 represent local maxima.

	TF1	TF2
Success rate	69.9 %	77.4 %
Average mission time	388 s	408 s

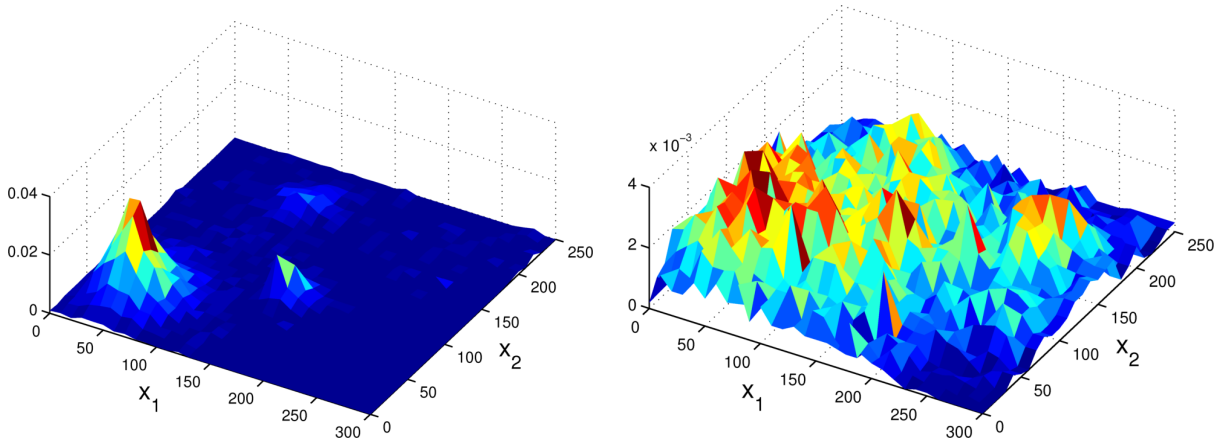
Table 3.2: Success rate and average mission time for Line Search

3.1.3 Metropolis-Hastings

The most important parameters are the bin size and ϵ. From the definition of the success rate in (3.1) it becomes clear that the larger the bin size, the higher the success rate. Also obvious, the longer the Markov chain runs, the more accurate is the result. This means a small ϵ yields a higher success rate but at the same time the mission time is elongated. The variance of the proposal distribution has a small influence. It should not be chosen too small, since then the Markov chain is slowly mixing. But if we choose it too large, the mission time is elongated due to the implemented traveling between rejected states. We select the following parameters:

$$b = 10\,\text{cm},\ \epsilon = 0.0001,\ \sigma^2 = 400\,\text{cm}^2. \tag{3.8}$$

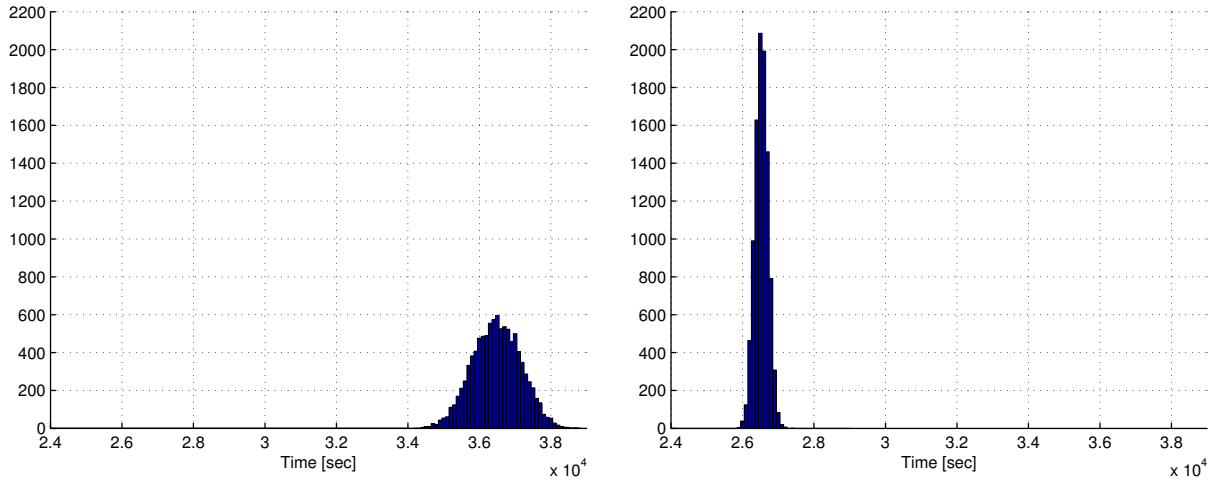
The particular choice of ϵ corresponds to about $N = 10\,000$ samples of the Markov chain. For these parameters we plotted two sample distributions of visited bins in Figure 3.11. We already see that for TF1 the sampled distribution represents the test field better than for TF2 (compare to Figure 3.2).



(a) TF1 **(b)** TF2

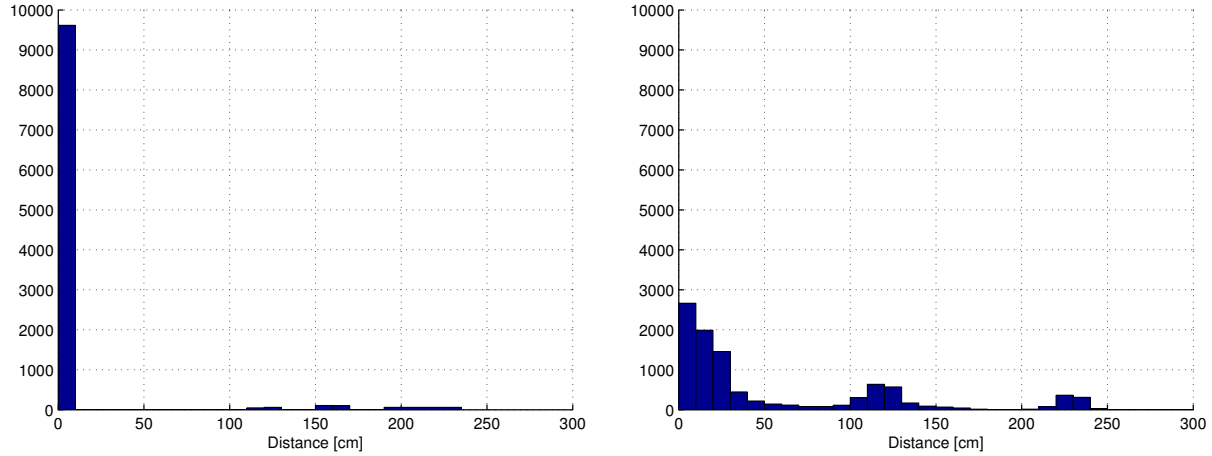
Figure 3.11: Simulated samples of Metropolis-Hastings. For TF1 the distribution seems to converge faster (spoken in number of chain transition steps) than for TF2.

For test field 1 with the steep peaks, the Metropolis-Hastings algorithm has a very high success rate of $r = 96.17\,\%$, but the mission time (the actual time the vehicle drives around) is extreme (Tab. 3.3). For the second test field, Metropolis-Hastings shows poor performance with a success rate of only 26.81 % and also an extremely high mission time. The mission times are normal distributed (Figure 3.12). For TF1 we have a high mean and high variance ($\sigma^2 = 765\,\text{cm}^2$), where for TF2 the variance ($\sigma^2 = 185\,\text{cm}^2$) is lower and the mean too. The higher mean for TF1 is due to the fact that it is more likely to produce rejected samples and thus the vehicle travels much more.



(a) M_1 **(b)** M_2

Figure 3.12: Distribution of the mission time ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$)[2].



(a) D_1 **(b)** D_2

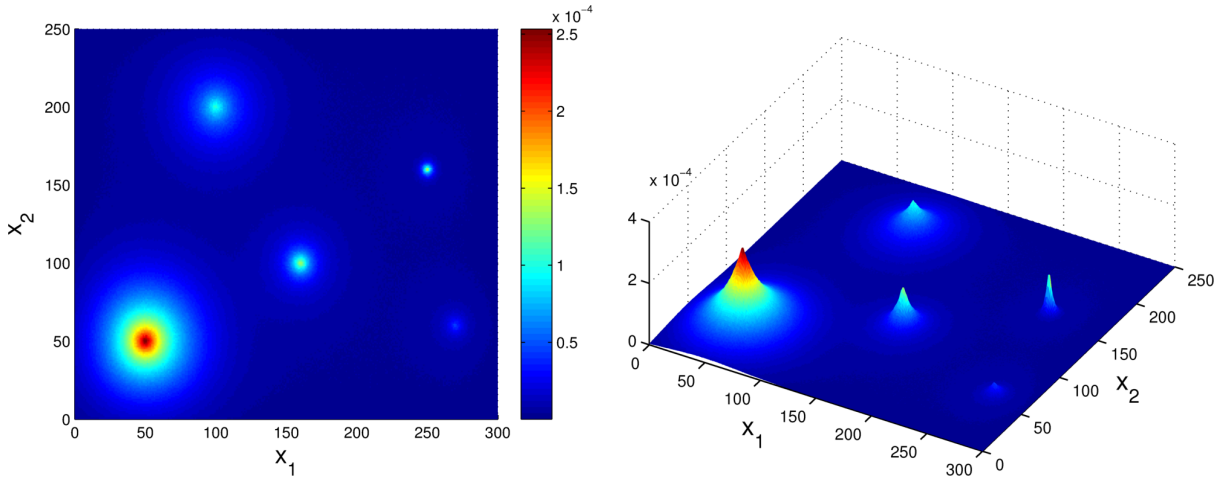
Figure 3.13: Distance from estimated to global maximum ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$). The peaks represent other local maxima.

	TF1	TF2
Success rate	96.2 %	26.8 %
Average mission time	36 477 s	26 532 s

Table 3.3: Success rate and average mission time for Metropolis-Hastings.

In Chapter 7 when we present the results from the real world robot, we only consider a TF1-like field. Thus for completeness we plotted the stationary distribution for TF1 in Figure 3.14. To obtain the converged Markov chain we let the vehicle drive for $N = 10^8$ iterations and used a bin size of 1 cm for higher accuracy. The stationary distribution represents the true function f_1 quite well.

[2]Note that N refers to the number of iterations as mentioned above, and not the number of transitions of the chains.



(a) Top view **(b)** 3D view

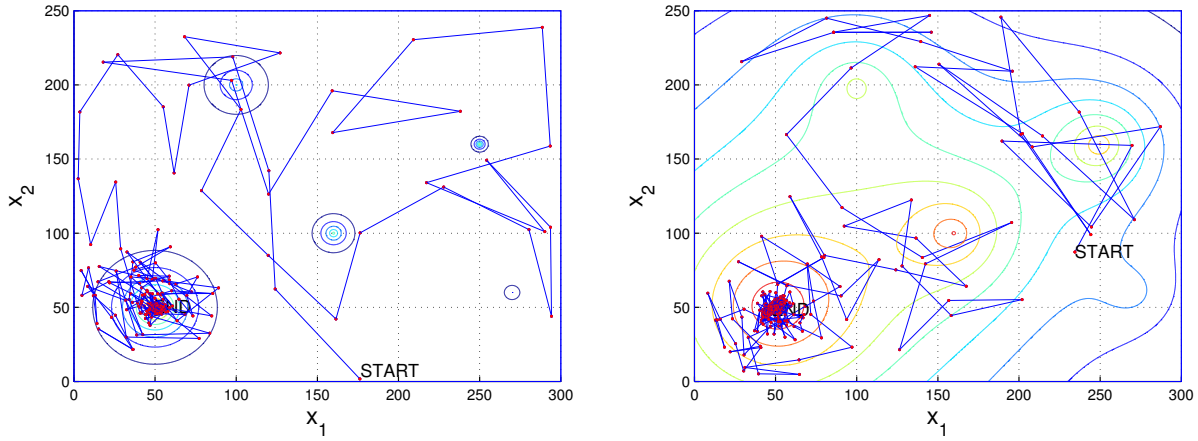
Figure 3.14: Stationary Distribution of Metropolis-Hastings of TF1. The converged distribution represents TF1 quite accurate.

3.1.4 Simulated Annealing

Since there are so many parameter, the selection becomes rather tedious. We quickly summarize the results from our simulations. The iterations per time period N_t has a significant influence on both the success rate and the mission time. If chosen too large the mission time increases drastically, while the success rate is high, but if chosen too low, the success rate decreases. Thus it is left to the designer to decide what is more important. The initial temperature T_0 needs to be above 1.5, otherwise the success rate drops drastically. Parameters γ and η should be chosen in the same range between 0.7–0.85. The initial step length needs to be long enough, but no too long. $R_0 = 80\,\text{cm}$ has shown good results. For the end condition, N_r should be large to achieve a high success rate, but the mission time is elongated. We make a trade-off between the two and choose $N_r = 8$. The last parameter left is the standard deviation of the bias on the angle of the proposal, σ_θ. The success rate shows a maximum between 0.4–0.8, hence we select $\sigma_\theta = 0.75$. The mission time is nearly unaffected by the bias. To avoid stability issues, we set a lower limit on the step radius $R_{min} = 2\,\text{cm}$. The parameter selection of Simulated Annealing exemplifies the 'Exploration versus Exploitation dilemma'. We found the following parameters to be appropriate:

$$N_t = 14,\ T_0 = 2,\ \gamma = 0.8,\ R_0 = 80\,\text{cm},\ \eta = 0.75,\ N_r = 8,\ \sigma_\theta = 0.75. \tag{3.9}$$

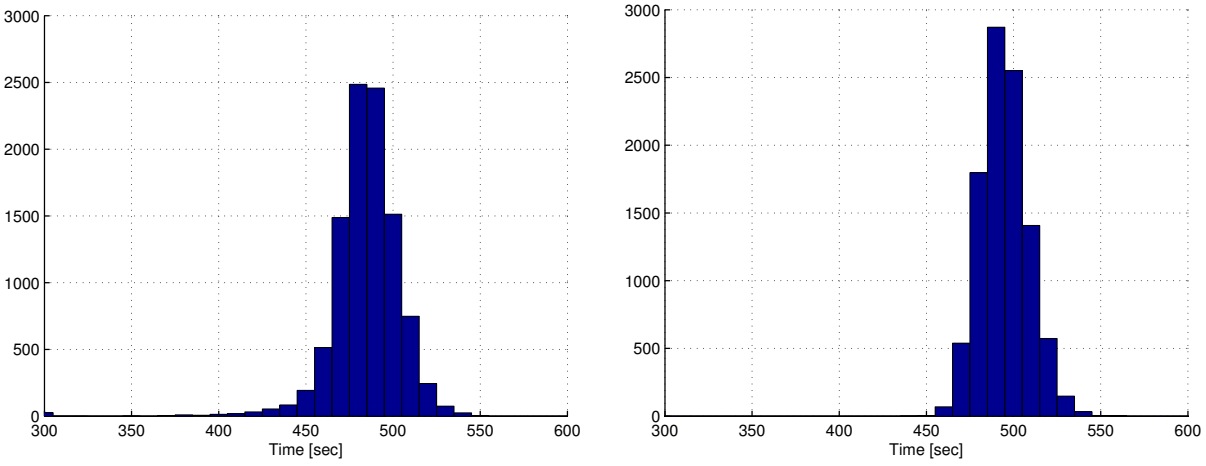
We plotted one sample path for both test fields in the Figure 3.15 below.



(a) TF1 (b) TF2

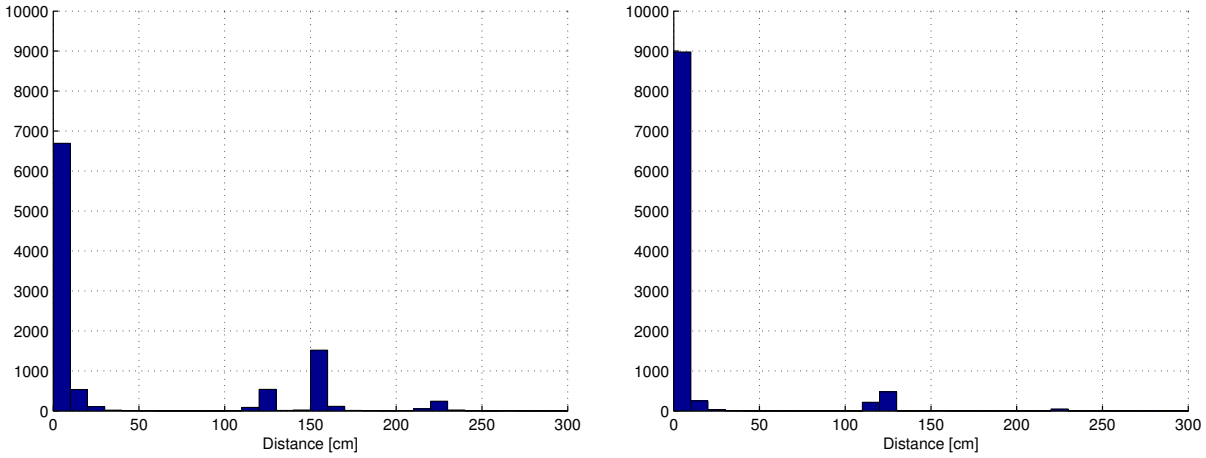
Figure 3.15: Simulated sample paths of Simulated Annealing. The vehicles path is indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. Also the contour of the concentration field is indicated. In the best cases the vehicle converges to the global maximum.

The distributions of the mission times and the distribution of the distances are plotted in Figures 3.16 respectively 3.17. The performance is better for the test field two, as the mission times are almost equal, but the success rate is significantly higher (Table 3.4). In test field 2 the vehicle is less likely to get trapped in a local maximum because it is more likely to escape it again.



(a) M_1 (b) M_2

Figure 3.16: Distribution of the mission time ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$).



(a) D_1 **(b)** D_2

Figure 3.17: Distance from estimated to global maximum ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$). The peaks represent other local maxima.

	TF1	TF2
Success rate	66.9 %	89.8 %
Average mission time	484 s	495 s

Table 3.4: Success rate and average mission time for Simulated Annealing.

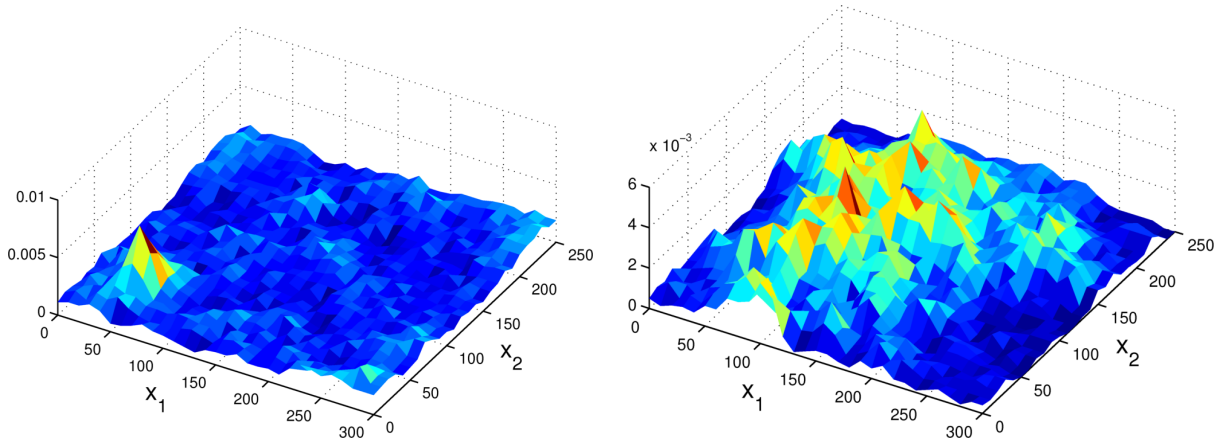
Alternatively the vehicle could return to the current maximum after each temperature period. Although this contradicts the idea of escaping a local minimum, the results are reasonable. We found that the success rate is slightly higher (2 %–5 %), but the mission time is also higher (ca. 500 s in average).

3.1.5 Stochastic Localization

To attain a comparable result to the Metropolis-Hastings, we choose the same bin size and the same ϵ. Also, the step length is again 10 cm (i.e. $\overline{v} = 10\,\text{cm/s}$, $T_s = 1\,\text{s}$). Therefore we only have to tune the parameters of the acceptance, J and K. Simulations have shown that $2.5 < J < 3$ and $3 < K < 3.5$ give the best performance for TF1, but they yield poor performance for TF2. Since it is difficult to find acceptable parameters for both test fields, we only focus on TF1 for the time being. We select the following parameters:

$$b = 10\,\text{cm}, \epsilon = 0.0001,\ J = 2.8,\ K = 3.25. \tag{3.10}$$

From Figure 3.18 we note that the distribution for TF2 has little resemblance to the actual test function while for TF1 the global maximum is clearly distinguishable. From this example it becomes clear that the distributions depend on the function and the selected parameters.

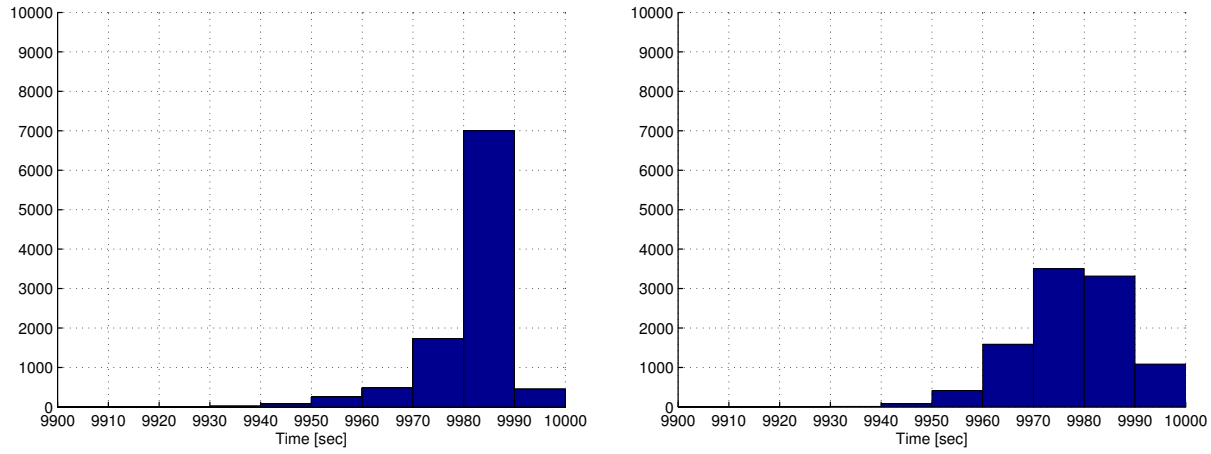


(a) TF1 **(b)** TF2

Figure 3.18: Simulated sample distribution of Stochastic Localization. For the given parameters, the distribution of the chain of TF2 represents TF2 quite poorly.

The distribution of the mission time is depicted in Figure 3.19 and the distribution of the distance in Figure 3.20. The average mission time and the success rate are summarized in Table 3.5. For TF2 the chosen set of parameters shows very bad performance. With a different choice of $J = 5$ and $K = 1.2$ we can increase the success rate to about 55 %. We found that the mission time correlates to the inverse of ϵ,

$$t \approx \frac{1}{\epsilon}. \tag{3.11}$$

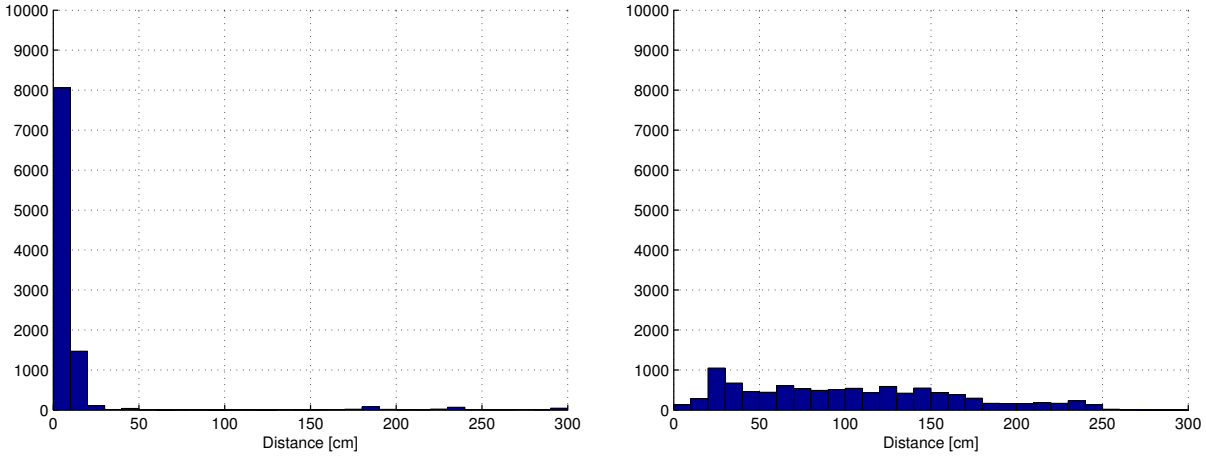


(a) M_1 **(b)** M_2

Figure 3.19: Distribution of the mission time ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$).

	TF1	TF2
Success rate	80.6 % (0.2 %)	1.3 % (55 %)
Average mission time	9982 s	9978 s

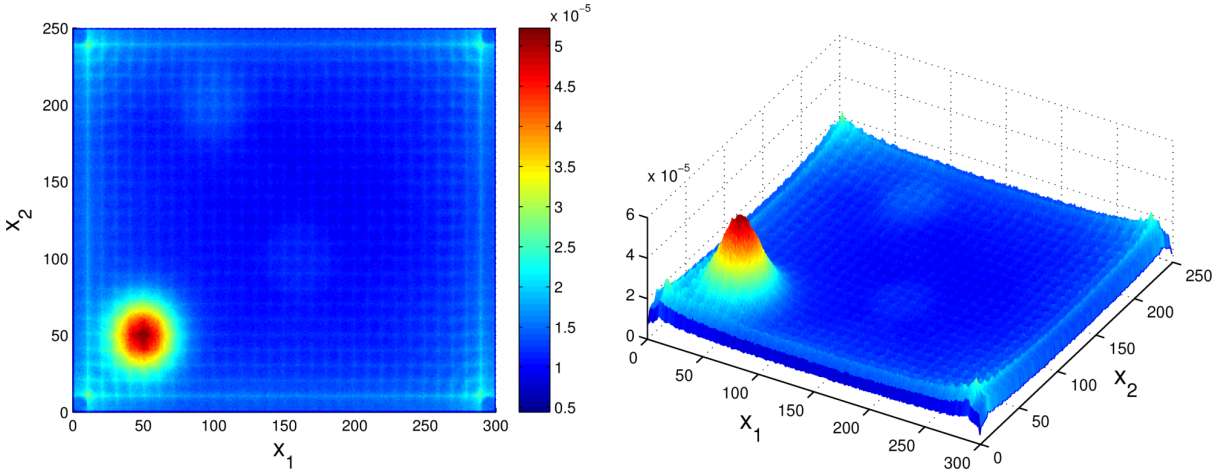
Table 3.5: Success rate and average mission time for Stochastic Localization.



(a) D_1 (b) D_2

Figure 3.20: Distance from estimated to global maximum ($N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$).

For completeness we have plotted the stationary distribution for TF1 in Figure 3.21. To obtain the converged Markov chain we let the vehicle drive until $N = 10^8$ iterations. The bin size was chosen to be equal to 1 cm for higher accuracy and visibility.

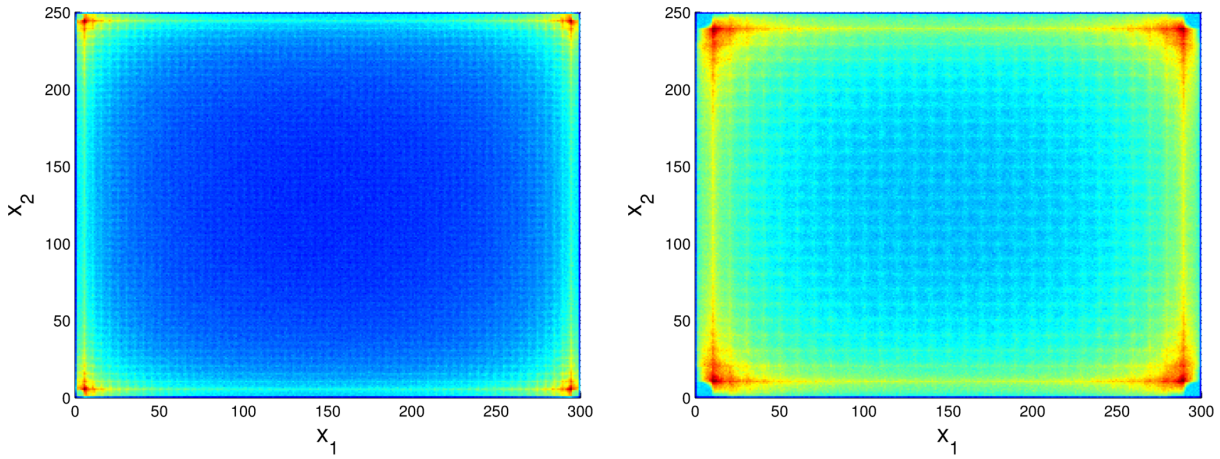


(a) Top view (b) 3D view

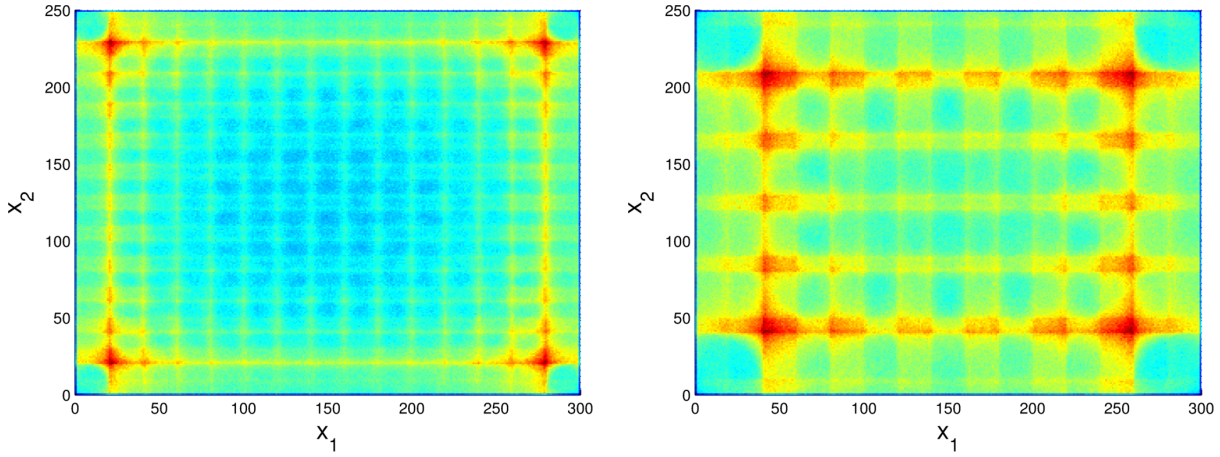
Figure 3.21: Stationary Distribution of Stochastic Localization for TF1. Apart form the global maximum and two local maxima, we observe a grid-like pattern leading to small peaks at the corners.

There are three main observations we make. First, the five peaks, except the global maximum, are almost not identifiable. This behavior is heavily influenced by the parameters J and K. Since we are only interested in the global maximum we chose the parameters in such a way that the global maximum is clearly detectable and thus this distribution is expected. Second, the distribution is not zero where it actually should be. However, this is not that surprising, since the vehicle needs to drive through the regions with low densities and thus adds to the bins, because the rejected state are also counted. Third, and not obvious, is the grid structure we see. Each border creates a pattern of lines in the same direction as the border. The magnitude of each line decreases with increasing distance from the border. On the four corners this effect overlaps from both adjacent borders and creates small peaks. To further investigate on this effect, we set $f(x) = 0$ for all $x \in \mathcal{D}$ and simulate the Markov chain again for 10^8 iterations and different step

lengths. As a result we get the distributions depicted in Figure 3.22. The grid is now clearly visible, the gap between two consecutive grid lines is exactly equal to the step length. Where the step length is a divider of the dimensions of $\mathcal{D}$, the lines from two opposite borders overlap each other creating a symmetric grid (Figures 3.22a, 3.22b), whereas the step length is not a divider the individual lines are visible (Figure 3.22c, 3.22d). In the latter case, the lines do not interfere out. We have to admit that we do not fully understand why this effect exists. Clearly it is dependent on the step size and the geometry of $\mathcal{D}$, but the the proposal orientations and therefore lines of movement are continuous and not aligned with the border. To understand the effect theoretically one should discretize $\mathcal{D}$, because in the discrete space the transition matrices can be calculated exactly. One then would see that a state on a line really has higher probability of being visited. However, this lies beyond the scope of the thesis and the magnitude of these patterns is small compared to the peaks around the global maxima and therefore negligible.



(a) l_s= 5 cm **(b)** l_s= 10 cm



(c) l_s= 20 cm **(d)** l_s= 40 cm

Figure 3.22: Sampled distributions for $f(x) = 0$, $N = 10^8$ and four different step lengths. The distributions show clearly a gridded pattern depending on the step length.

At this point we would like to mention that the authors from [2] have simulated this algorithm with more than one vehicle. We will not discuss this setup here, but we will implement the algorithm for several robots later. The implementation will be discussed in Chapter 5 in general and in Chapter 6 in more detail for Stochastic Localization.

3.1.6 Comparison

Let us quickly summarize our findings in Table 3.6.

		TF1	TF2	Influence to Parameters
Grid Search	$\overline{r}$	100%	99.4%	little
	$\overline{t}$	779 s	779 s	
Line Search	$\overline{r}$	69.90%	77.4%	medium
	$\overline{t}$	388 s	408 s	
Metropolis-Hasting	$\overline{r}$	96.20%	26.8%	little
	$\overline{t}$	36'477 s	26'532 s	
Simulated Annealing	$\overline{r}$	66.90%	89.8%	large
	$\overline{t}$	483 s	495 s	
Stochastic Localization	$\overline{r}$	80.60%	1.3 % (50 %)	large
	$\overline{t}$	9'982 s	9'978 s	

Table 3.6: Results from the simulations

The most obvious fact is that Metropolis-Hastings and Stochastic Localization require much more time to terminate than the other three algorithms. This is due to their totally different strategy of locating the maximum. What these algorithms actually do is to try to reshape the whole form of $f(\cdot)$. In a later section, we will thus set a time limit on the algorithms and terminate them after it has elapsed. Furthermore, we note that Simulated Annealing and Stochastic Localization are very sensitive to their parameters in terms of success rate. The main problem is that the parameters depend heavily on the function itself, which is assumed to be unknown. Therefore the optimal parameters are also unknown. We have seen that Metropolis-Hastings and Stochastic Localization perform better on a field with steep peaks than smooth peaks, for Simulated Annealing and Line Search the converse is true. Also, the parameters depend on the preference of the designer, if a higher success rate or a lower mission time is desired. If we only compare Metropolis-Hastings and Stochastic Localization, we observe that after stationarity has been reached, Metropolis-Hastings (Figure 3.14) maps the concentration function quite exactly, whereas Stochastic Localization (Figure 3.21) is influenced by the grid pattern noise, rendering the local extrema almost unobservable. We see for TF1 Metropolis-Hastings has a very high success rate at the cost of a high mission time, while the success rate of Stochastic Localization is still high, but the time is considerably lower.

The fact that Grid Search shows such good performance (100 % success rate at $\bar{t} \approx 800\,\mathrm{s}$), is due to the small test board we are using. For larger regions, Grid Search will most probably still have 100 % success, but the mission time increases disproportional. If we are forced to make a general statement, we would say for TF2 that Simulated Annealing shows the best performance, whereas for TF1 Grid Search provides the best performance. Note that for the case of small region, Grid Search is actually a good choice since it is independent of $f(\cdot)$.

3.2 Robustness Analysis

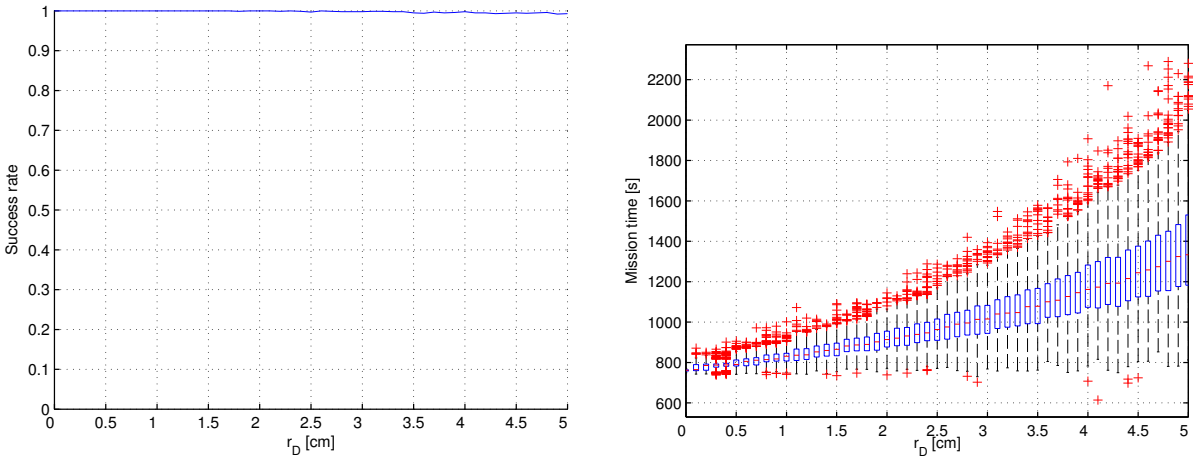
To test the algorithms on robustness we simulate them again with different values for the position noise and measurement noise. For these simulations we only use the function from test field 1.

First we vary the position noise r_D from 0 cm to 5 cm with a constant $\sigma_\nu = 0.02$, then we vary the standard deviation of the measurement noise σ_ν between 0 and 1 while $r_D = 0.2\,\mathrm{cm}$ is

held constant, for each algorithm. The plots of the success rates and mission times are shown below in Fig. 3.23 to 3.32. Just as a reminder, the condition $\overline{v}\,T_s > 2\,r_D$ must always be met. To prevent the vehicle from leaving $\mathcal{D}$, no algorithm allows a new state not to be in $\mathcal{D}_r$. Due to the position noise the vehicle eventually will enter the border region ($\mathcal{D}_B = \{x \in \mathcal{D} \,|\, x \notin \mathcal{D}_r\}$), but not leave $\mathcal{D}$.

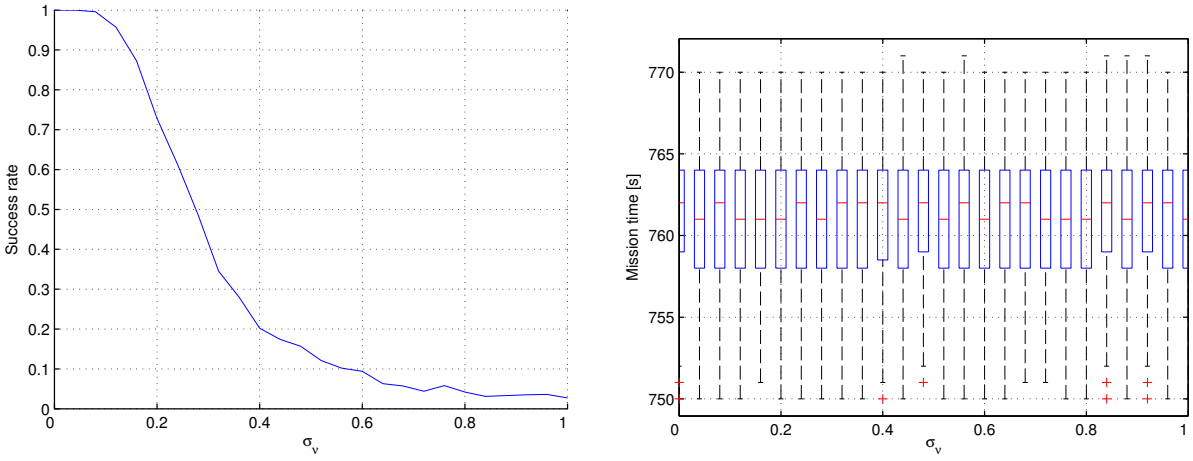
Grid Search

The success rate is nearly constant equal to 1 under position noise, whereas the mean and variance of the mission time increase (Figure 3.23). The success rate under heavy measurement noise drops rapidly to zero, and as expected the mission time is not influenced in any way by the measurement noise (Figure 3.24).



(a) Success rate **(b)** Mission time

Figure 3.23: Success rate and mission time for different r_D ($N = 1000$, σ_ν=0.02).

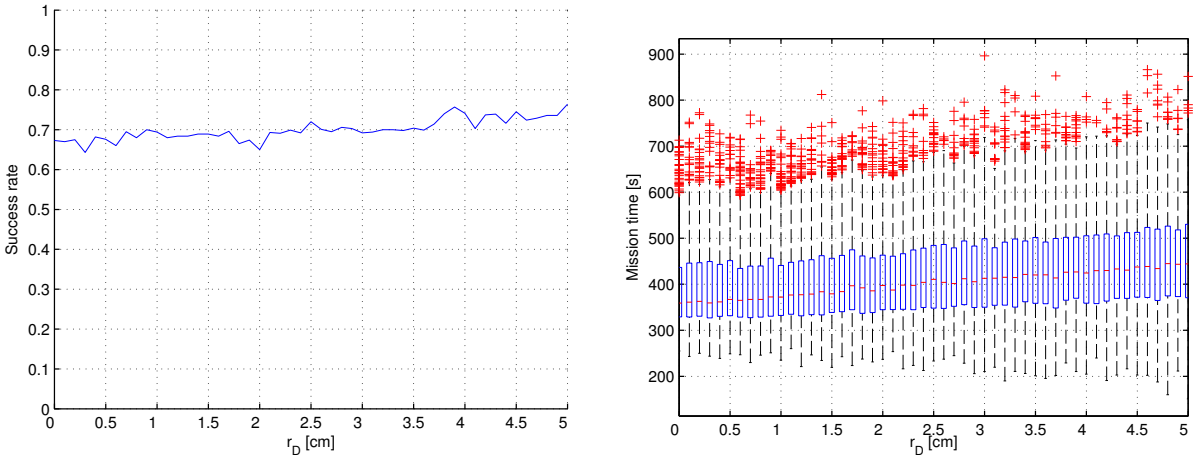


(a) Success rate **(b)** Mission time

Figure 3.24: Success rate and mission time for different σ_ν ($N = 1000$, $r_D = 0.2\,\text{cm}$).

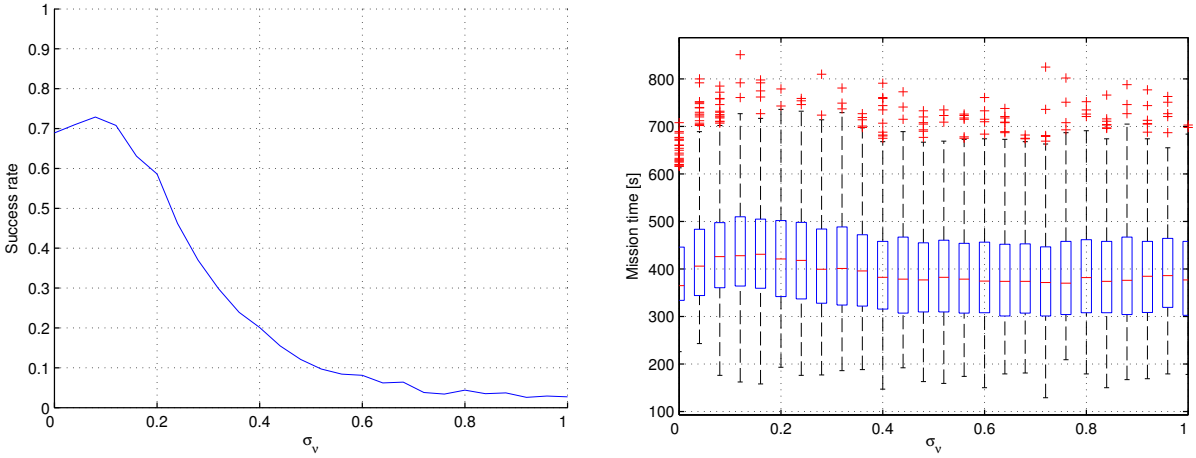
Line Search

Heavy position noise has a small positive effect on the success rate and a small negative influence on the mission time (Figure 3.25). The overall movement is still line shaped, but the points are randomly distributed along a line which leads to a better exploration of the space. For different measurement noise variances the success rate increases shortly before dropping to zero, while the mission time also declines (Figure 3.26).



(a) Success rate **(b)** Mission time

Figure 3.25: Success rate and mission time for different r_D ($N = 1000$, σ_ν=0.02).

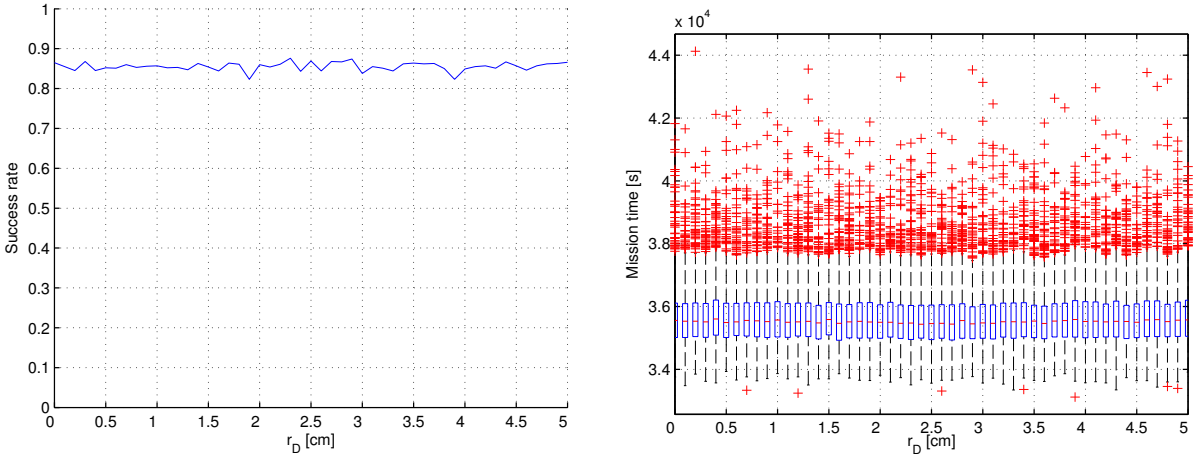


(a) Success rate **(b)** Mission time

Figure 3.26: Success rate and mission time for different σ_ν ($N = 1000$, $r_D = 0.2\,\text{cm}$).

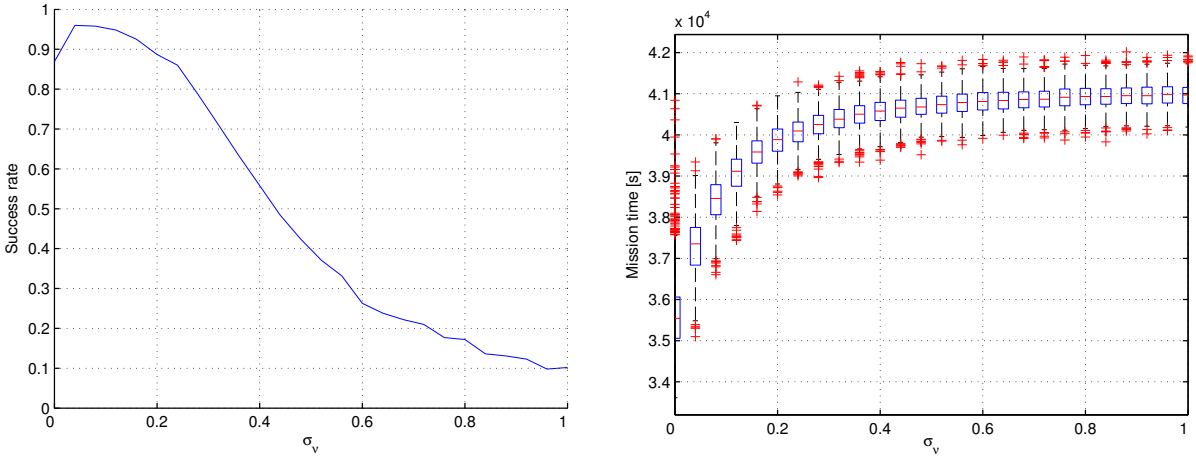
Metropolis-Hastings

The Metropolis-Hastings algorithm is completely independent of position noise (Figure 3.27). This result is obvious, since position noise acts as a deformation on the proposal distribution, but Metropolis-Hastings operates with any proposal distribution. Also, the algorithm is less sensitive to measurement noise than the preceding two but the curve has a similar shape as the curve for Line Search (Figure 3.28).



(a) Success rate **(b)** Mission time

Figure 3.27: Success rate and mission time for different r_D ($N = 1000$, σ_ν=0.02).

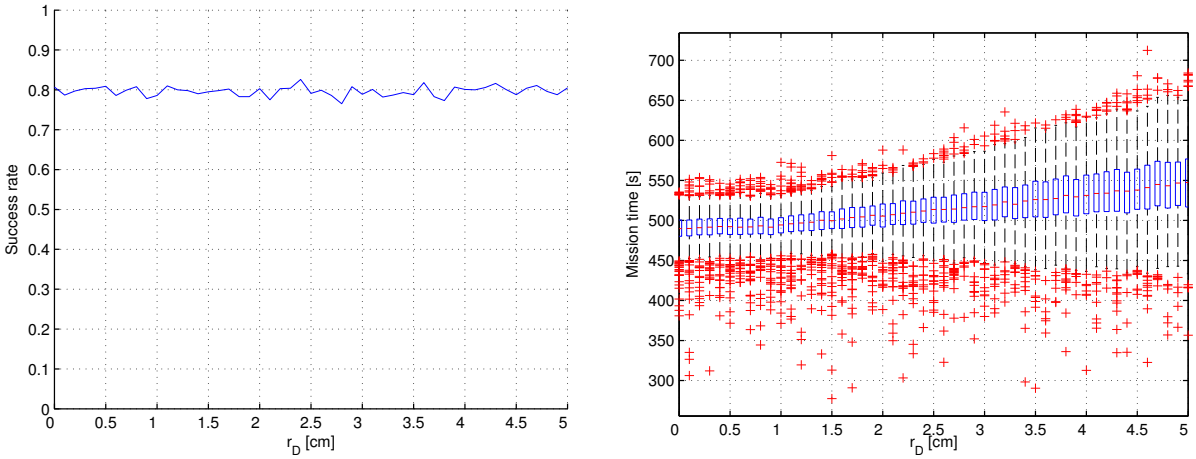


(a) Success rate **(b)** Mission time

Figure 3.28: Success rate and mission time for different σ_ν ($N = 1000$, $r_D = 0.2\,\text{cm}$).

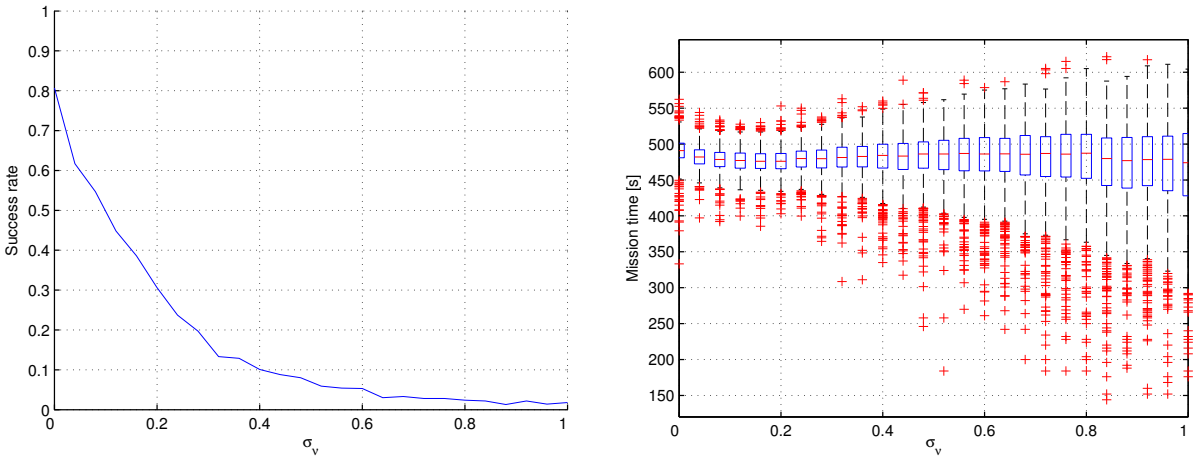
Simulated Annealing

For Simulated Annealing we introduced a second lower bound on the step length, which is $2 \cdot r_D$, to prevent problems close to the border of $\mathcal{D}$. Since Simulated Annealing is also a MCMC-type algorithm, the success rate is also independent to position noise, but the mean and variance of the mission time slightly increase (Figure 3.29). On the other hand it is very sensitive to measurement noise as the success rate drops rapidly to zero with increasing σ_ν (Figure 3.30).



(a) Success rate **(b)** Mission time

Figure 3.29: Success rate and mission time for different r_D ($N = 1000$, σ_ν=0.02).

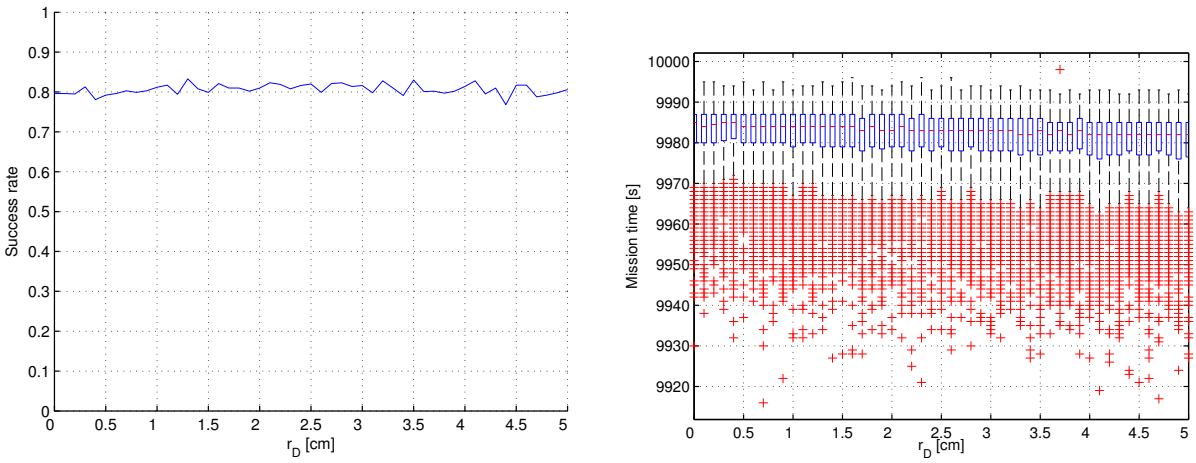


(a) Success rate **(b)** Mission time

Figure 3.30: Success rate and mission time for different σ_ν ($N = 1000$, $r_D = 0.2\,\text{cm}$).

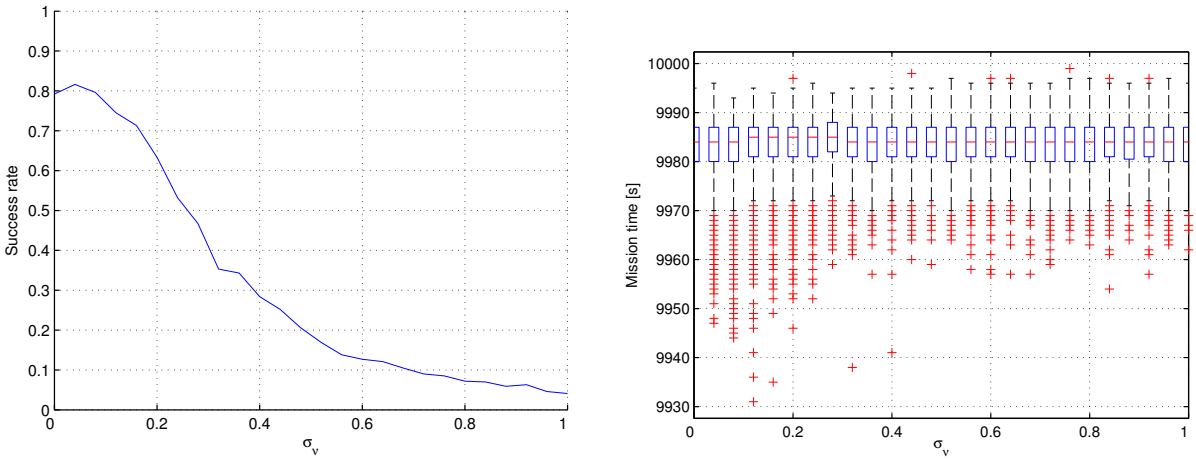
Stochastic Localization

As the other two MCMC algorithms, Stochastic Localization is also invariant to position noise (Figure 3.31). The success rate under measurement noise drops slower than for Simulated Annealing, but faster than for Metropolis-Hastings (Figure 3.32). The underlying functional form seems to be the same for all three MCMC algorithms, expect the offsets and the small increase in the beginning differ. The mission time is anyway independent of the noise, since it is approximately equal to the inverse of ϵ (3.11).



(a) Success rate **(b)** Mission time

Figure 3.31: Success rate and mission time for different r_D ($N = 1000$, σ_ν=0.02).



(a) Success rate **(b)** Mission time

Figure 3.32: Success rate and mission time for different σ_ν ($N = 1000$, $r_D = 0.2\,\text{cm}$).

One important conclusion we draw from these findings is that all algorithms are not sensitive to positions noise in view of the success rate. This comes in very handy when we implement this algorithm on the hardware.

3.3 Comparison in Large Region

We are also interested in the large scale behavior of the five algorithms. Therefore we multiply each dimension of $\mathcal{D}$ by a factor of 10, and we obtain a new $\mathcal{D}' = [0, 3000] \times [0, 2500]$. We construct a third test field based on test field 2. The coordinates of the extrema of $f_2(\cdot)$ are also multiplied 10 times, and we obtain $f_3(\cdot)$, whose graph is shown below in Figure 3.33.

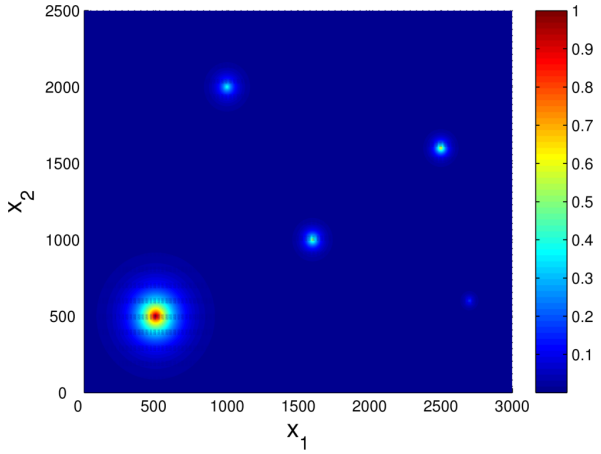


Figure 3.33: Graph of the scaled function, TF3. Both dimensions and the positions of the maxima are multiplied by a factor of 10.

We need to adapt some parameters of the algorithms according to the new setting (Table 3.7). Furthermore, we set the radius for a success to 100 cm. The velocity and the sampling time stay the same, $\overline{v} = 10\,\text{cm/s}$ respectively $T_s = 1\,\text{s}$.

Line Search	**Metropolis-Hastings**	**Simulated Annealing**	**Stochastic Localization**
$l = 3800\,\text{cm}$	$\sigma^2 = 4000\,\text{cm}^2$	$R_0 = 800\,\text{cm}$	$b = 70\,\text{cm}$
	$b = 70\,\text{cm}$		$\epsilon = 10^{-5}$

Table 3.7: New parameters to adjust to the large region

Table 3.8 below presents the success rates and average mission times. As expected, Grid Search has its mission time multiplied by factor 100, while Line Search and Simulated Annealing only have mission times approximately ten times larger. Also the time of Stochastic Localization is increased tenfold, but this is due to the choice of ϵ. Considering the success rates we note that Grid Search maintains its success guaranty, while the other algorithms decrease in performance. The highest decrease has the Metropolis-Hastings, since the success rate halves.

Please not that we did not change ϵ for Metropolis-Hastings, because it would require even more time (approx 10 times) than it already does. To have at least a similar value of the mission time as Stochastic Localization we left ϵ unchanged.

Grid Search	
Success rate	99.6 %
Average mission time	76 851 s
Line Search	
Success rate	54.9 %
Average mission time	4442 s
Metropolis-Hastings	
Success rate	49.2 %
Average mission time	118 890 s
Simulated Annealing	
Success rate	54 %
Average mission time	4622 s
Stochastic Localization	
Success rate	76.1 %
Average mission time	99 902 s

Table 3.8: Success rate and average mission times, for $N = 10\,000$, $r_D = 0.2\,\text{cm}$, $\sigma_\nu = 0.02$.

3.4 Simulation with Time Limit

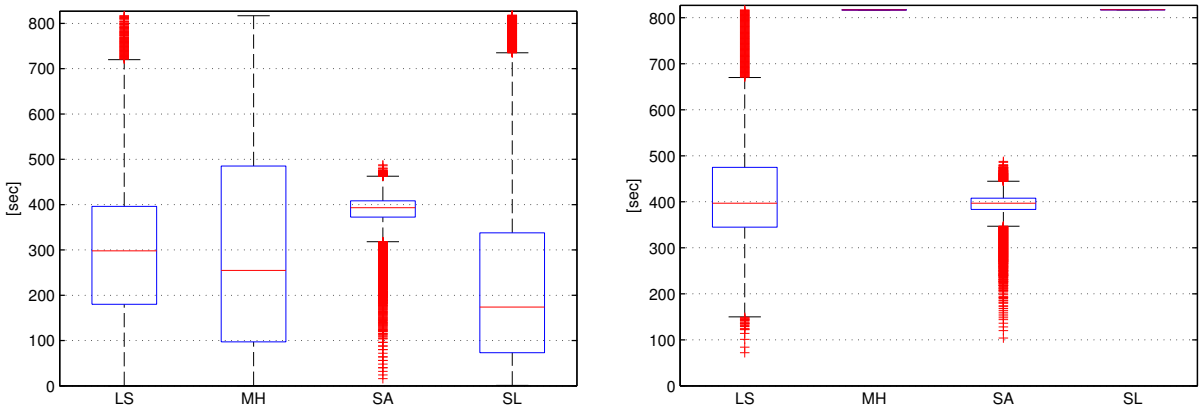
In this section, we revert back to the scenario with test field 1. As we have seen in the previous Sections, Metropolis-Hastings and Stochastic Localization have a huge mission time and thus seem rather useless. As mentioned earlier, we thus set a limit on the mission time and look directly at the measured concentrations and not on the distribution of visited positions. However, for Grid Search, Line Search and Simulated Annealing the built-in terminal condition is still active, e.g. Simulated Annealing can terminate earlier than the time limit (one would need to readjust the parameters in order to use the full time that is given). Following the idea that Grid Search explores the whole region and thus sets an upper limit to the mission time T_{limit}, we stop the algorithms after T_{limit} has expired. Furthermore, we record the time when the vehicle hits the global maximum for the first time. Note that the first hit time only exists when the run itself was successful. Naturally the global maximum is unknown, but to calculate the first hit time we assume it is known. Also, the setup is different than in the previous section. Instead of looking at the two test fields, we will look at $N_f = 20$ different randomized fields. The fields share the number, weight and steepness of the peaks as TF1 but their positions is randomly drawn over $\mathcal{D}' \subset \mathcal{D}$. $\mathcal{D}'$ has all its border 20 cm away from the borders of $\mathcal{D}$, because all algorithms except Grid Search perform poorly when the global maximum is close to the border. All parameters except N_r for Simulated Annealing ($N_r = 5$, to reduce the exploitation phase) stay the same as in the initial setting.

As mentioned, the time limit is the largest possible time Grid Search needs, which is $T_{limit} = 817\,\text{s}$. From Table 3.9, where the success rates and first hit time averages are shown, we read that Stochastic Localization has the highest success rate and the shortest average first hit time. On the other hand, Metropolis-Hastings still has a poor success rate. In the following Figure 3.34, on the left is the distribution of the first hit time and on the right the distribution of the total mission time plotted. As expected Metropolis-Hastings and Stochastic Localization need the full mission time, but rather surprisingly Stochastic Localization is the one with the lowest average first hit time. This is due to the fact that Stochastic Localization lets the vehicle explore the space quite fast. Furthermore also as expected Line Search and Simulated Annealing terminate before T_{limit} has expired. For Simulated Annealing both distributions almost coincide, because

the built-in terminal condition terminates the algorithm also when the vehicle drove to a local maximum, whereas Line Search has has a higher average on the mission time than on the first hit time.

Line Search	
Success rate	76.1 %
Average first hit time	298 s
Metropolis-Hastings	
Success rate	49.6 %
Average first hit time	305 s
Simulated Annealing	
Success rate	67.7 %
Average first hit time	380 s
Stochastic Localization	
Success rate	95.2 %
Average first hit time	228 s

Table 3.9: Success rates and average first hit times



(a) Distribution of the first hit time

(b) Distribution of the total mission time

Figure 3.34: Results when setting a mission time limit. The medians of each group are marked by the red line inside the boxes. The edges are the 25th and 75th percentiles, the whiskers extend to the most extreme data points, while outliers are plotted in a red cross. Note that Line Search and Simulated Annealing still can terminate before the time has expired.

We are now interested in what happens with the success rates when we further reduce the time limit. For this comparison we also include Grid Search, the grid size is adjusted to the time limit such that the vehicle covers the whole space given the velocity of 10 cm/s. In Figure 3.35 below we have on the x-Axis the decreasing time limit and on the y-axis the success rates. We note that Stochastic Localization shows good performance and is only topped by Grid Search for high time limits. Line Search and Simulated Annealing are not influenced by the time limits first, because they naturally terminated before the time has elapsed. For $T_{limit} < 440\,\text{s}$ the success rate of Simulated Annealing drops very rapid. Alternatively, one could readjust the parameters such that these two algorithms are also terminated by the time limit. We expect then the success rate to be higher for high time limits than it is now, because the exploration phase is elongated. Furthermore, Metropolis-Hastings performs very poorly. However, these findings must be treated with caution. The parameters of the algorithms were not adjusted to the new setting (i.e. the

time limit), so there might be a parameter setup that leads better performance at a given time limit. For example, we think that Simulated Annealing could be improved for high and low time limits (also Line Search, Metropolis-Hastings and Stochastic Localization could be improved). In order to have a fair comparison, one should pick a number of time limits and then adjust the parameters of the algorithm for each limit such that the success rate is optimized. Then we could produce a clear statement about the algorithms, and not the parameters of the algorithms.

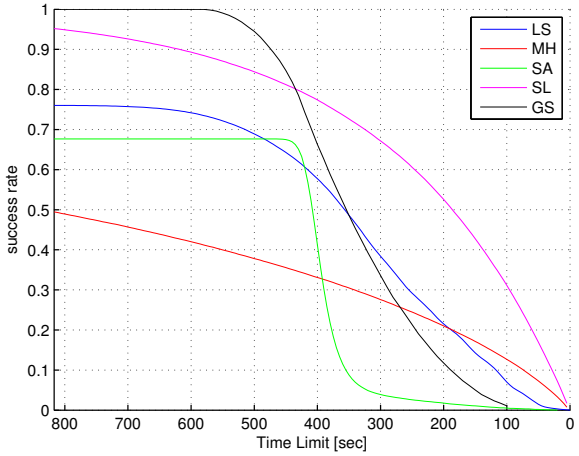
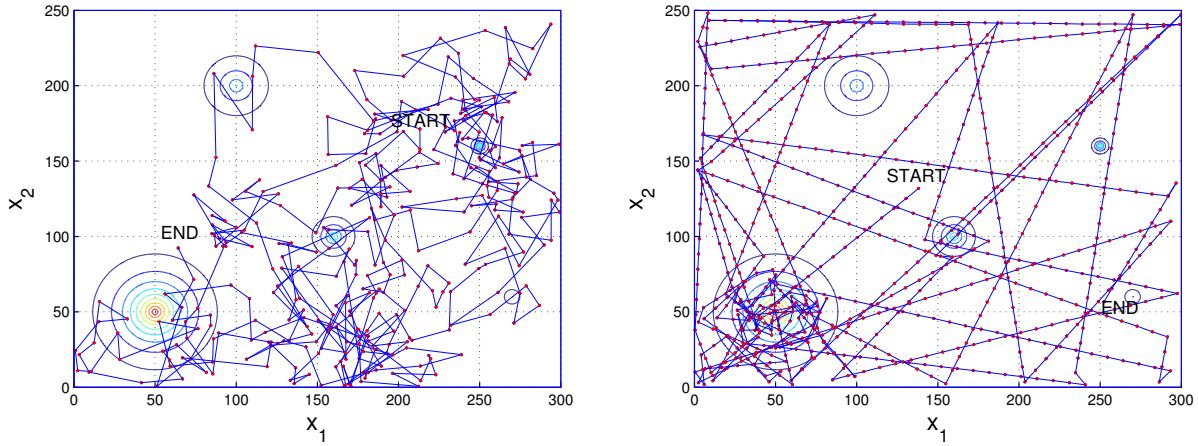


Figure 3.35: Succes rates over decreasing time limit for the five algorithms. LS = Line Search, MH = Metropolis-Hastings, SA = Simulated Annealing, SL = Stochastic Localization, GS = Grid Search. For low time limits Stochastic Localization seems to perform best.

For completeness, we plotted one sample path for Metropolis-Hastings and Stochastic Localization in Figure 3.36 below. Interestingly, Stochastic Localization reminds us of Line Search (see Figure 3.8, because the vehicle usually drives straight on a line. This observations is supported by the fact that Stochastic Localization and Line Search show the best performance in Table 3.9.



(a) Metropolis-Hastings **(b)** Stochastic Localization

Figure 3.36: Sample paths when set a time limit of $T_{limit} = 817\,\mathrm{s}$. The vehicles path is indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. Also the contour of the concentration field is indicated. While the movement in Metropolis-Hastings seems purely random, the vehicle turns in Stochastic Localization only in regions of high concentration or at the border.

Chapter 4

Mobile Robot Test Bed

In this chapter we present the experimental setup, the robot we utilize as vehicle an we introduce the model of it.

4.1 Setup

The robots move on a ground plane of 300 cm × 250 cm. In order to detect the robots, we use a Logitech HD Pro Webcam C920 which is mounted 2.6 m above the ground plane approximately in the middle of it. Unfortunately, the resolution properties of the camera do not allow us to use the full space of the plane and we leave a safety margin of 10 cm to the border. Thus the usable space decreases to

$$\mathcal{D} = [0, 240] \times [0, 175]. \tag{4.1}$$

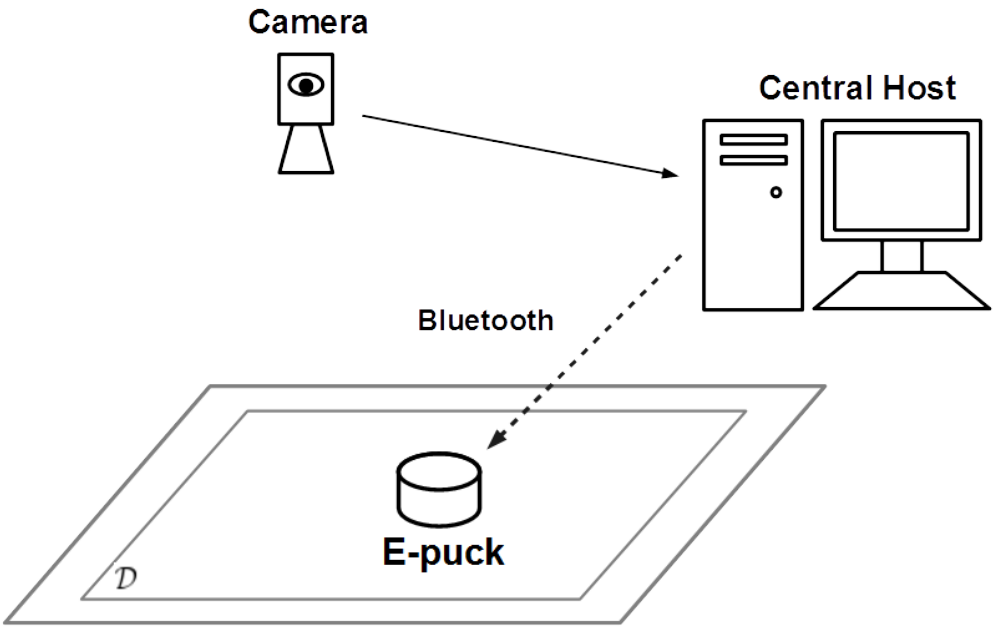


Figure 4.1: Setup of the project. The robots (E-puck) move on the plane $\mathcal{D}$ and are connected via bluetooth to the central host. A camera is used to detect the robots. A central host is used to process the images of the camera and the algorithms.

The images from the camera are processed by the central host, then it sends navigation commands via bluetooth to the E-puck. A schema of the setup is drafted in the Figure 4.1 above. With the central host we create the function f over $\mathcal{D}$. Since the robot cannot actually measure the function, we simulate the measurements on the central host.

4.2 Hardware

The experimental test bed (described above) utilizes the two wheeled E-puck robot. It is an educational robot developed by EPF Lausanne and now distributed by GCtronic [20], an image is shown in Figure 4.2 [21]. The E-puck includes many features, such as 2 individual motors, three microphones, a speaker, proximity sensors, a bluetooth adapter, a camera, a 3D accelerometer and many more. For our project we need only the wheel motors for motion and the bluetooth adapter for communication with the host.

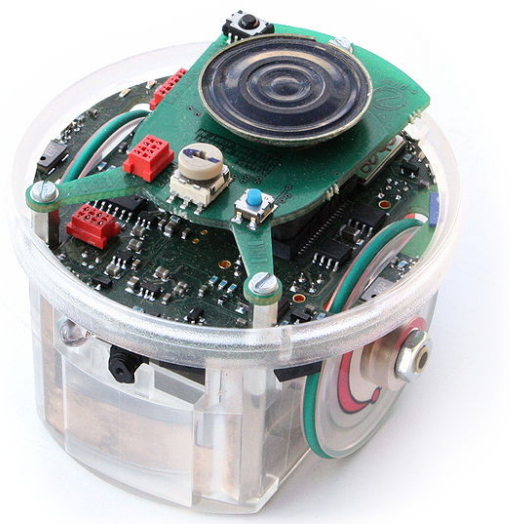


Figure 4.2: Image [21] of the robot, called E-puck.

The E-puck is equipped with a Microchip dsPIC30F6014A digital signal controller. The main specifications of this chip are: 49 kB flash memory, 8 kB data memory, 30 MHz processor [20]. So, the capabilities on the E-puck itself are very limited, but sufficient for our project. The two wheels are operated by two step motors, which can be controlled separately. For one revolution the motors need 1000 steps, thus allowing very precise odometry. Furthermore, the response of the motors to input changes is very quick and the mass of the E-puck is low (200 g). These two properties allow us to use the assumption that we can directly control the velocities of the wheels by setting the steps per second of the motors. In other words, we assume that the robot has infinite acceleration. The maximum step per second we can set is $|u_{max}| = 1000/\text{s}$. A standard library of basic functionality for the E-puck is provided by the distributer or freely downloadable on the EPFL site of the E-puck [24]. Many of these functions were used for our software implementations on the E-puck.

4.3 Model of the E-puck

We model the E-puck as a unicycle located at the mass center (i.e. the middle) of the robot. A rudimentary model of the robot is drawn in Figure 4.3. The weel velocities are denoted as v_L respectively v_R, the longitudinal speed of the center is $\overline{v}$ and the angular velocity ω.

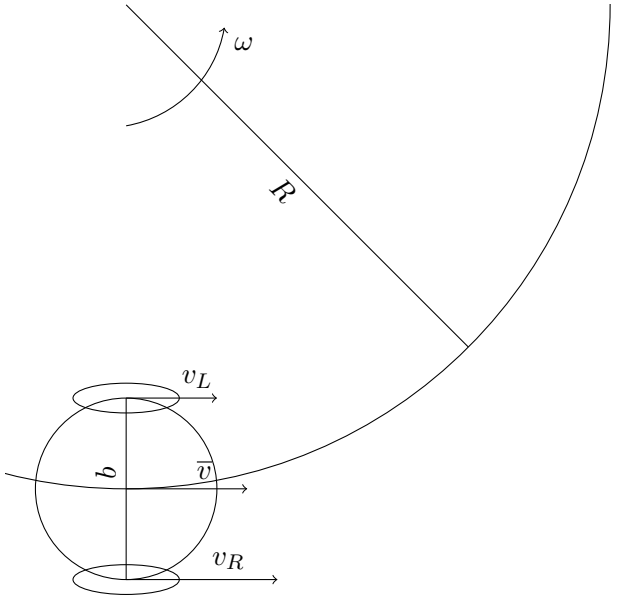


Figure 4.3: Model of the E-puck. The wheel speeds v_L and v_R are translated into longitudinal and angular velocity for the unicycle model.

We can easily convert the steps per second u into velocity with the formula below

$$v = \frac{u}{1000}\,\pi\, d_{wheel}, \tag{4.2}$$

where $d_{wheel} = 4.1\,\text{cmis}$ the wheel diameter. The maximum speed is thus

$$v_{max} = \frac{u_{max}}{1000}\, d_{wheel}\, \pi = 12.88\,\frac{\text{cm}}{\text{s}}. \tag{4.3}$$

With the wheel speeds in $[cm/sec]$ we compute the velocity of the middle point $\overline{v}$ to be

$$\overline{v} = \frac{v_R + v_L}{2}. \tag{4.4}$$

The angular velocity ω is calculated as

$$\omega = \frac{v_R - v_L}{b}, \tag{4.5}$$

where $b = 5.3\,\text{cm}$ is the distance between the wheels. The relationship between $\overline{v}$ and ω is

$$\overline{v} = \omega\, R. \tag{4.6}$$

Given the longitudinal velocity and the angular velocity we calculate the motor speeds to be

$$v_L = \overline{v}\,(1 - \frac{b}{2R}) = \overline{v} - \frac{\omega\, b}{2}, \tag{4.7}$$

$$v_R = \overline{v}\,(1 + \frac{b}{2R}) = \overline{v} + \frac{\omega\, b}{2}. \tag{4.8}$$

The state s of the robot consists of the position $(x, y) \in \mathbb{R}^2$ and the heading angle $\alpha \in [0, 2\pi]$. The dynamics of the robot are given in the formula below. The inputs are the velocity $\overline{v}$ and the angular velocity ω.[1]

$$s = \begin{pmatrix} x \\ y \\ \alpha \end{pmatrix} \qquad \dot{s} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\alpha} \end{pmatrix} = \begin{pmatrix} \overline{v}\cos\alpha \\ \overline{v}\sin\alpha \\ \omega \end{pmatrix} = f(s, \overline{v}, \omega) \tag{4.9}$$

In discrete time, the dynamics become

$$s_{k+1} = s_k + \begin{pmatrix} \overline{v}_k \cos(\alpha_k) \\ \overline{v}_k \sin(\alpha_k) \\ \omega_k \end{pmatrix} T_{ep}, \tag{4.10}$$

where T_{ep} is the sampling time.

Since we have three degrees of freedom but only two controllable degrees of freedom the system is nonholonomic. This fact complicates the design of a controller (see Section 5.3). Furthermore, we set some artificial constraints on the capabilities of the robot. First, we assume that the wheels cannot rotate backwards and that the mean speed has a lower limit $v_{min} = \frac{v_{max}}{5}$. This means that the robot cannot stop. Also, the robot turning rate is limited by $\omega_{max} = 2.4302\,\mathrm{rad/s}$,

$$\begin{aligned} v_{max} \geq v_L \geq 0\,\frac{\mathrm{cm}}{\mathrm{s}}, \\ v_{max} \geq v_R \geq 0\,\frac{\mathrm{cm}}{\mathrm{s}}, \\ \overline{v} \geq v_{min}, \\ |\omega| \leq \omega_{max}. \end{aligned} \tag{4.11}$$

To turn the robot, there must be a difference in the individual wheel speeds (4.5). If we want to steer the robot with minimal turning radius R_{min}, we must employ the maximal possible ω. In order to calculate R_{min} we first need to compute the maximal longitudinal velocity while turning with $\omega = \omega_{max}$ (denoted as v'_{max}). We know for sure that $v_R = v_{max}$, thus (4.8) becomes (with $\overline{v} = v'_{max}$),

$$v_{max} = v'_{max} + \frac{\omega_{max}\, b}{2}, \tag{4.12}$$

solving for v'_{max} yields

$$v'_{max} = v_{max} - \frac{\omega_{max}\, b}{2} = 6.44\,\frac{\mathrm{cm}}{\mathrm{s}}. \tag{4.13}$$

The minimum circle radius imposed by (4.13) and (4.6) is thus

$$R_{min} = \frac{v'_{max}}{\omega_{max}} = 2.65\,\mathrm{cm}. \tag{4.14}$$

The minimum circle radius is exactly half of the robot's wheel distance b. In this case the right wheel spins with full speed while the left wheel stands still. (Of course, the derivation could have been done for the opposite case with $\omega = -\omega_{max}$) We set these constraints to make our setup a bit more realistic, for example when those algorithms would be applied to vehicles with non-stopping constraints, e.g. jet-propelled drones.

[1] For the robot the velocities $\overline{v}$ and ω are time-dependant, these actual velocities are not to be confused with the velocity of the algorithm given in (2.2) and (2.4).

Chapter 5

Control

This chapter deals with the controller design, describing the main control structure and a more in depth analysis of its different parts. The whole control structure is drawn in Figure 5.1. It consists of three sub-systems indicated by the dashed boxes. They are distinguished by their functionality. The algorithms described in Chapter 2 form the upper level controller. The second sub-system is the robot detection part, consisting of the camera, the low-level controller and an extended Kalman filter (EKF). Last, the third sub-system is the robot itself with the same low-level controller.

The three sub-systems also differ in their sampling times. The algorithm sampling time T_s is chosen by the designer with some constraints, which we will see later in this chapter. The same low level controller is implemented once on the robot to generate the wheel inputs and once on the central host to generate the inputs for the extended Kalman filter. We choose this type of structure, because it would be too expensive (e.g. communication time lag) to send the inputs from the robot back to the central host. Therefore we have only one way communication from the central host to the robots. The lower level controller receives at ever time step k from the algorithms the estimate of the current state $\hat{s}_k$ and the next state s_{k+1}. It then translates this information into motor inputs signal. The sampling time for the low level controller is $T_{ep} = 20\,\text{ms}$. Since it does not receive any state updates during the time interval $[k \cdot T_s, (k+1) \cdot T_s]$, this controller is open loop. The low level controller keeps track of the state using the model of the E-puck from (4.9). To generate state estimates we use a camera and an extended Kalman filter since the system (4.10) is nonlinear. The camera generates state measurements z_i at time instances $i \cdot T_{ekf}$. This sampling time is again different than the two mentioned above. Furthermore, T_{ekf} is not constant due to our implementation. It varies slightly depending mostly on the image processing and measurement generation part, then the instantaneous available computing power on the central host and on the resolution of the camera. For our chosen resolution we obtain a mean sampling time of $\overline{T}_{ekf} = 47\,\text{ms}$.[1]

In the following three sections the E-puck detection, the Kalman filter and the low level controller are described in more detail, and in the last section we take a look at the performance of the whole control and measurement part.

[1] The distribution of T_{ekf} is not Gaussian, because the maximum frame rate of the camera is 30. This leads to a minimal $T_{ekf,min} = 33\,\text{ms}$. However, we can calculate the standard deviation, it is equal to $11.2\,\text{ms}$. These values of the mean an standard deviation are valid only if there is only one robot. Also note that the sampling time of an algorithm T_s does not vary in the sense T_{ekf} does, because the whole algorithm sub-system is implemented in a separate thread than the robot detection sub-system.

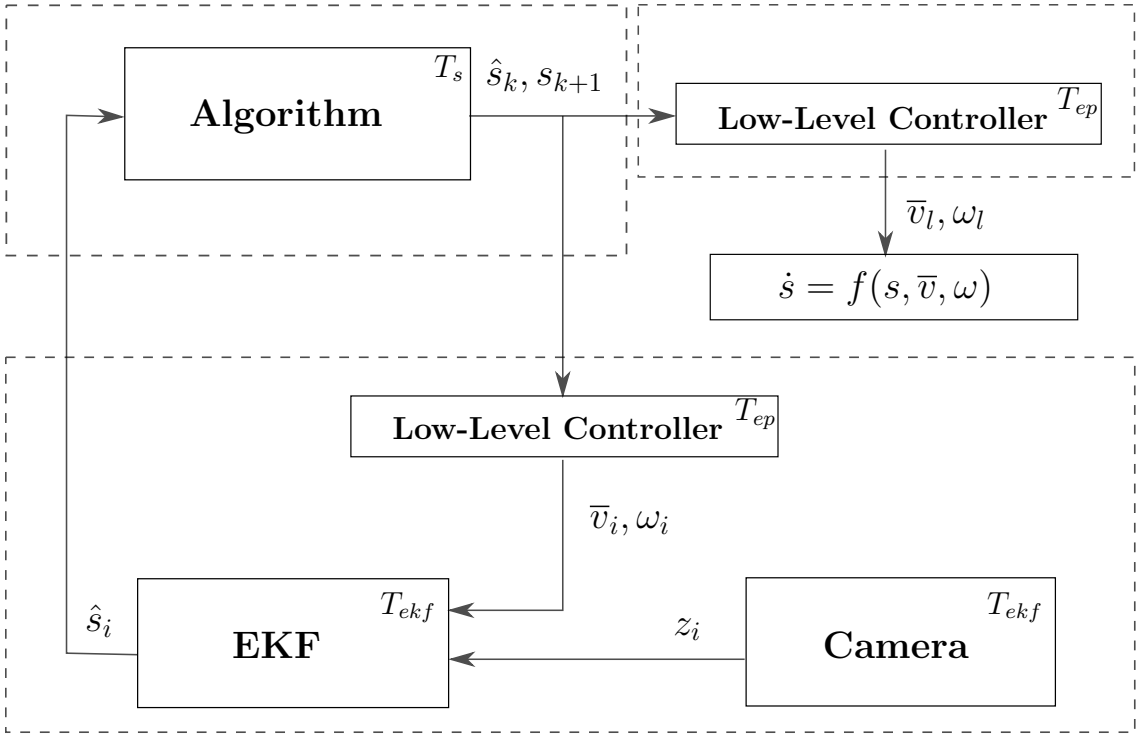


Figure 5.1: Controller schema. The dashed boxes denote the tree sub-systems. The algorithms (top left) and the robot detection (bottom) sub-system are implemented on the central host, while a low-level controller (top left) operates on the E-puck. The algorithm receives state estimates from an extended Kalman filter and tells the low-level controller where to drive the e-puck next. The EKF receives state measurements from the camera and inputs from the low-level controller in order to estimate the state. The low-level controller for the EKF is implemented on the central host too.

5.1 E-puck Detection

As mentioned before, we use a camera to detect the state of an E-puck. The camera creates images of the ground plane and everything on it. We need a way to extract the E-puck from these images. Fortunately, other people have faced similar problems and large libraries for computer vision have been created. We will use the open source libraries opencv [16] and cvblobslib [17].

In computer vision the coordinate system is usually different. The camera measures from the top left corner which is the origin. A schematic of the camera coordinate system is shown below in Fig. 5.2. Also the extended Kalman filter, which will be described in Section 5.2, operates with this coordinate system.

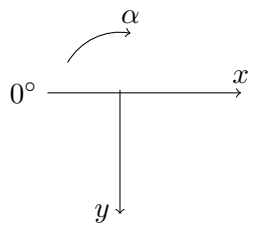


Figure 5.2: Coordinate system of the camera and extended Kalman filter.

Extracting different colors on an image is a well understood process and implemented in opencv. However, we only use two colors, blue and black, because they are especially robust to detect in terms of environmental condition (e.g. lighting conditions). On every Robot we stick a blue paper circle with a black dot indicating its head direction. The blue circle is used for (x, y) measurements and in combination with the black dot we can determine the heading angle α of the robot. To ensure good distinguishability we painted the background (i.e. the ground plate) with white color. Our Robot detection program proceeds as follows. First, all correct sized blue blobs are detected and if there is the black dot on the blue blob, we create a new valid measurement candidate. Second, if two E-puck are stuck together we have a special routine that is still able to recognize the two individual robots, rather than filtering out the oversized blue blobs. However, encounters with more than two robots are not handled by the code, but in our project this case never happened. Then, after all blobs from the camera are evaluated, we have n measurement candidates. Consider the case where we have m E-pucks. Please note that n does not have to be equal to m, as we shall now explain. If $n > m$ we have to many measurements, this can happen when two or more E-pucks are close together. If $n < m$ we have not enough measurements, which happens when one ore more Robots drove outside the scope of the camera or were removed manually. Furthermore, none of the above effects could have happened ($n = m$) or both at the same time or multiple times. Thus we have a variety of scenarios leading to the particular values of n. This implies that we need an intelligent matching of the new measurement candidates to the already existing blobs from the previous time step.[2] The objective is to match the measurement candidates to the existing blobs with the scenario that is the most probable. By scenario we mean one particular assignment possibility out of all possibilities. The most probable scenario j^* is defined where the weighted norm of the sum of errors is minimized

$$||s||_{w2} = x^2 + y^2 + 100\,\alpha^2 \tag{5.1}$$

$$j^* = \arg\min_j \sum_{i=1}^{m} ||\hat{s}_i - \overline{s}_{i,j}||_{w2} \ , \tag{5.2}$$

where $\hat{s}_i$ is the predicted new state (since the measurements are for the next time step) from robot i and $\overline{s}_{i,j}$ denotes the measurement at scenario j assigned to robot i. The question remains, how are the scenarios generated? Theoretically we could consider the maximum case where any measurement could be assigned to any blob. We know though that the robot cannot drive too far in a time period of T_{ekf}, so we set a filter radius of 30 cm for each robot. Only measurements inside this circle are considered. 30 cm seems quite large, but think of the case when the robot drove outside or there weren't any measurements of it for a couple of seconds. In that time, the robot could have easily driven around 30 cm. This brings us to another point. On one hand, if for a blob there is for some reason no measurement (i.e. $\overline{s}_{i,j}$ does not exist), we must ensure that no measurement will be assigned to it. On the other hand, we need to penalize the case of not having a measurement. In practice we achieve this by setting a penalty to this case of $||\overline{s}_{i,j}||_{w2}$ = 999. Thus we do not accidentally have a no measurement of a blob, but at the same time a ghost blob[3] is created. If we have found the most probable scenario, the measurement candidates are assigned to the blobs and treated as true measurements of that respective blob. If in the end, there are still valid measurement candidates[4] left, they must be newly added robots and therefore m will be increased. To make the blob matching more stable, a blob is only considered lost and deleted, when this blob did not receive any new measurements for N consecutive time

[2] At the initialization, no blobs exist yet. They are created according to where the measurements are and assigned to the robots in ascending y-order.

[3] A ghost blob is a blob, where no robot exists.

[4] A valid measurement candidate is a measurement that is physically possible, e.g. if two measurements are closer than 2 cm one measurement is not valid.

steps. This prevents deleting blobs when for example the robot drove shortly outside the scope of the camera but again back inside.

This blob matching function is one reason for the varying sampling time T_{ekf}. The time required for this function depends on how many scenarios are possible. This is seen in the fact that when let several robots run the mean of the sampling time is increase (for seven robots, $\overline{T}_{ekf} = 63\,\text{ms}$).

Now, we have successfully assigned the measurements to the correct blobs. Unfortunately, these measurements are influenced by noise, mostly caused by the camera resolution which allows us only to measure discrete values of x, y and α. Consequently we need a filter for the measurements, which is describe in the section below.

5.2 Extended Kalman filter

We will not restate the main formulas of the Kalman filter, as we assume the reader is already familiar with Kalman filtering. For a refresher on the (extended) Kalman filter we refer to [18].

We need to slightly adjust the state space equations of the E-puck (4.10) to the camera coordinate system, leading to changes in the sign.

Furthermore, we add some process noise $\xi_k \in \mathbb{R}^3$ with the process covariance $Q \in \mathbb{R}^{3\times3}$ to the state equation and measurement noise $\eta_k \in \mathbb{R}^3$ with measurement covariance $R \in \mathbb{R}^{3\times3}$ to the output equation. Both noises are assumed to be zero mean multivariate Gaussian distributions.

The dynamics with noise then become (note that some signs have changed due to the different coordinate system)

$$s_{k+1} = s_k + \begin{pmatrix} -\overline{v}_k \cos(\alpha_k) \\ -\overline{v}_k \sin(\alpha_k) \\ \omega_k \end{pmatrix} T_{ekf} + \xi_k, \tag{5.3}$$

the observation model is

$$z_k = \begin{pmatrix} x_k \\ y_k \\ \alpha_k \end{pmatrix} + \eta_k. \tag{5.4}$$

While running tests we observed that the EKF itself operates for the camera resolution (640 $\times$ 480, 30 fps) as well as for a higher resolution (1920 $\times$ 1080, 5 fps). The lower limit of sampling time T_{ekf} depends on the frames per second (fps) of the camera and the fps decrease with increasing resolution. For the low resolution we have $\overline{T}_{ekf} = 47\,\text{ms}$ and for the high resolution $\overline{T}_{ekf} = 200\,\text{ms}$. Using a low resolution leads to a faster updating of the state, but the measurement is more noisy. As a drawback we are not able to cover the whole platform and hence have a artificial reduction of the region D. On the other hand, the high resolution yields measurements with higher precision on a larger region, but at the cost of reduced sampling time. We decided to use the low resolution, because otherwise we also need to increase the sampling time of the algorithms, which implies a longer mission time.

For the extended Kalman filter we need to linearize the model (5.3) without the noise around the current state and we obtain the state transition matrix as

$$F_k = \begin{pmatrix} 1 & 0 & \overline{v}_k\, T_{ekf} \sin(\alpha_k) \\ 0 & 1 & -\overline{v}_k\, T_{ekf} \cos(\alpha_k) \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.5}$$

The same is done for the observation matrix and we obtain

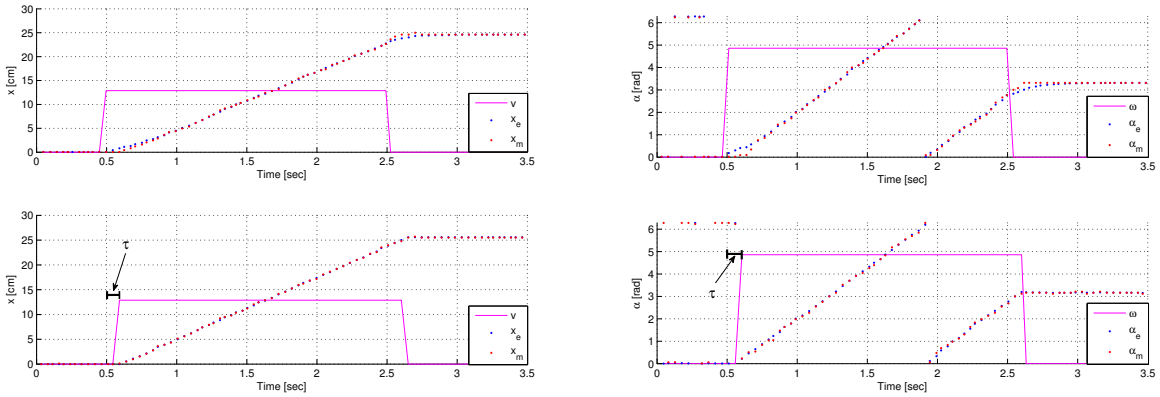
$$H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.6}$$

Usually the true values of the two covariance matrices are unknown and thus can be used as tuning parameters. We found the following values to yield satisfactory performance.

$$Q = \begin{pmatrix} 0.02 & 0 & 0 \\ 0 & 0.02 & 0 \\ 0 & 0 & 0.02 \end{pmatrix} \tag{5.7}$$

$$R = \begin{pmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.2 \end{pmatrix} \tag{5.8}$$

We will now show the behavior of the EKF with an example of a step change in $\overline{v}$ and ω on a robot in Figure 5.3. If you look closely at the output of the Kalman filter in the upper two images, you note that the estimate in both x and α react before the measurement does. This leads to small distortions when we try to read out the state estimate after an input change. The reason for this is that the input is send via bluetooth to the robot and there further processed, which of course requires time. Therefore we should take this delay into account in the Kalman filter. In the two lower images you see the performance of the filter with the delayed inputs. The input command is still sent at the same time as before (at 0.5 s), but now the inputs is applied after $\tau = 120$ ms to the Kalman filter, as they are applied to the robot's motors. We can read from the image that the estimate follows the measurement much nicer, thus eliminating (or at least drastically decrease) a source of errors.



(a) Step change in $\overline{v}$ **(b)** Step change in ω

Figure 5.3: Response of the EKF step changes to in $\overline{v}$ and ω . The magenta lines denote the inputs to the EKF, the red dots are the measurements x_m and the blue dots are the estimates x_e. In the lower two images the input to the Kalman filter is delayed by $\tau = 120$ ms. The delay is introduced to match the delay of the bluetooth signal. Consequently, without the delay the estimate changes before the measurement does at the input changes. This fact is seen in the two upper images at the times 0.5 and 2.5 s in the deviation of the blue dots to the red dots. The introduced delay removes this systematic deviation.

At this point we would like to mention that we first implemented a different Kalman filter than the above presented. In this first version, we did not have the velocities as inputs, but we treated them as unknown state variables which are also to be estimated. This filter worked fine, but the estimate (especially of the angle) was heavier inflicted by noise. However, all algorithms with one robot worked fine with this filter. We then tried to run Stochastic Localization with seven robots with the collision avoidance strategy discussed in Chapter 6. Due to the noisy angle estimates the robots did not proceed as calculated by the host. In consequence the robots crashed into each other quite often. We thus saw the need to improve the Kalman filter by feeding it with the inputs. A proper analysis of the whole system is provided in Section 5.4.

5.3 Low Level Controller

As already mentioned in the head of this section, the low level controller receives the current state $\hat{s}_k$ of the E-puck and next state s_{k+1} from the currently running algorithm. Using this approach, we obtain an open loop controller. That is, the controller does not receive any new state update until the next time step $k+1$. Therefore we need a trajectory strategy on the E-puck, which is described in the following subsection. Fortunately, the E-puck has a lot of built in functionalities and we can use odometry as internal state estimation. This internal estimation operates with a sampling time of $T_{ep} = 20\,\text{ms}$. The fact that we need inputs for the Kalman filter as well implies that we must also implement the low level controller on the host. One probably wonders now why we do not recalculate these inputs every time a new state estimation is available and effectively close the loop? The main problem is that the communication rate between host and robot is limited by the bluetooth connection, so first, we are never able to achieve a sampling time of 20 ms and second we would require another controller for path tracking. Last but no least, our algorithms do not require prefect path tracking, they also work under position noise, which is created by the open loop low level controller. In any case, the overall system is still closed loop as the algorithms receive new state information at every time step k.

5.3.1 Robot Motion Strategy

The firmware of the robot must ensure that it will reach the target without obtaining new state updates. Of course, this leads to some noise on the position and the angle. To determine its internal state, the robot uses its wheel speeds and the equations from Section 4.3.

The strategy we choose to move from s_k to s_{k+1} is first described in [19] by L.E. Dubins in 1957. For the remainder of this subsection the start state is denoted as $s_0 = \hat{s}_k$ and the terminal state as $s_N = s_{k+1}$. Basically, the robot drives first on a circle A with radius R, then a straight line, and then again on a circle B with radius R, where we set $R = R_{min}$ from (4.14). The two circles are chosen perpendicular to the starting state s_0 respectively the terminal state s_N. This leaves us with a total of four circles as depicted in Figure 5.4. The straight line is the tangent that connects the chosen two circles.

$$s_0 = \begin{pmatrix} x_0 \\ y_0 \\ \alpha_0 \end{pmatrix} \qquad s_N = \begin{pmatrix} x_N \\ y_N \\ \alpha_N \end{pmatrix} \tag{5.9}$$

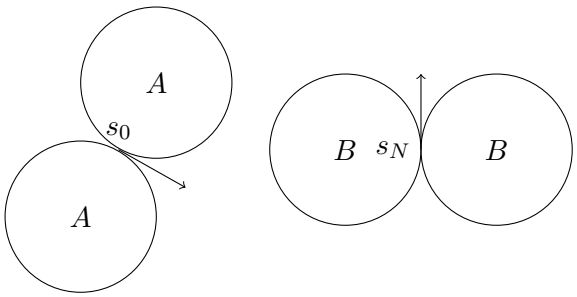


Figure 5.4: Start and terminal states have both a left and right rotating circle.

For simplicity, the robot's coordinate system is the normal Cartesian system. To drive a left circle, we need $\omega = \bar{v}/R$, where for a right circle we need $\omega = -\bar{v}/R$. In order to calculate the actual path of the robot we need some geometry. Remember, we have twice two circles to choose from, leading to four cases. First we can choose the circles both to the left (left-left), second both to the right (right-right), third A to the left and B to the right (left-right) and fourth A to the right and B to the left (right-left). These four cases are described in the following paragraphs. In the end the controller chooses the case where the total length is the shortest to be the actual path. We will later refer to this strategy as the Dubins path or the Dubins strategy.

Case left-left The drawing for this case is shown in Figure 5.5. What we need to describe the movement of the robot (depicted by the blue line) is the angle γ and the point Q. The movement is split in three parts, first rotate on the circle A until the $\alpha = \gamma$, then drive straight until $(x, y) = Q$ and last, rotate on circle B until $\alpha = \theta_N$. The points A, B, Q, P and the angle γ are extracted using basic geometry. First, we calculate the centers of the circles, A and B:

$$\begin{pmatrix} A_x \\ A_y \end{pmatrix} = A = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + R \begin{pmatrix} -\sin(\alpha_0) \\ \cos(\alpha_0) \end{pmatrix}, \tag{5.10}$$

$$\begin{pmatrix} B_x \\ B_y \end{pmatrix} = B = \begin{pmatrix} x_N \\ y_N \end{pmatrix} + R \begin{pmatrix} -\sin(\alpha_N) \\ \cos(\alpha_N) \end{pmatrix}. \tag{5.11}$$

Determining the distance $\overline{PQ} = L$ is easy, since it is equal to $\overline{AB}$. We get

$$\Delta x = B_x - A_x, \tag{5.12}$$
$$\Delta y = B_y - A_y, \tag{5.13}$$
$$L = \sqrt{\Delta x^2 + \Delta y^2}. \tag{5.14}$$

Also we can directly calculate γ as

$$\tan\gamma = \frac{\Delta y}{\Delta x}. \tag{5.15}$$

Further, we calculate

$$P = A + \frac{R}{L} \begin{pmatrix} \Delta y \\ -\Delta x \end{pmatrix}, \tag{5.16}$$

$$Q = B + \frac{R}{L} \begin{pmatrix} \Delta y \\ -\Delta x \end{pmatrix}. \tag{5.17}$$

The angular velocities are $\omega_A = \omega_B = \bar{v}/R$.

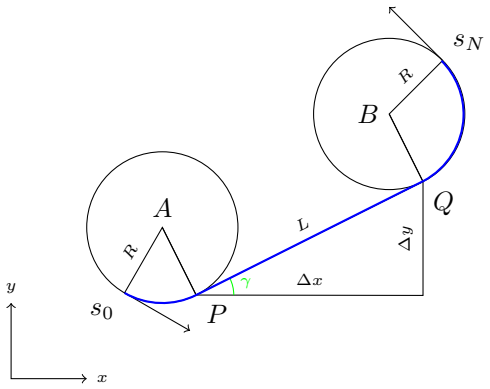


Figure 5.5: Geometry for the case left-left. The blue line depicts the Dubins path.

Case right-right The case with two right circles is very similar to the left-left case. The formula for L and γ essentially stay the same. The only difference lies in the location of A, B, P and Q. We get,

$$A = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + R \begin{pmatrix} \sin(\alpha_0) \\ -\cos(\alpha_0) \end{pmatrix}, \tag{5.18}$$

$$B = \begin{pmatrix} x_N \\ y_N \end{pmatrix} + R \begin{pmatrix} \sin(\alpha_N) \\ -\cos(\alpha_N) \end{pmatrix}, \tag{5.19}$$

$$P = A + \frac{R}{L} \begin{pmatrix} -\Delta y \\ \Delta x \end{pmatrix}, \tag{5.20}$$

$$Q = B + \frac{R}{L} \begin{pmatrix} -\Delta y \\ \Delta x \end{pmatrix}. \tag{5.21}$$

The angular velocities are $\omega_A = \omega_B = -\bar{v}/R$.

Case left-right This case differs from the other two above. The setup is illustrated in Figure 5.6. We calculate for the centers of the circles

$$A = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + R \begin{pmatrix} -\sin(\alpha_0) \\ \cos(\alpha_0) \end{pmatrix}, \tag{5.22}$$

$$B = \begin{pmatrix} x_N \\ y_N \end{pmatrix} + R \begin{pmatrix} \sin(\alpha_N) \\ -\cos(\alpha_N) \end{pmatrix}. \tag{5.23}$$

The distance $\overline{PQ} = D$ is calculated as

$$D = \sqrt{L^2 - 4R^2}, \tag{5.24}$$

where we need the constraint $L^2 \geq 4\,R^2$. The physical meaning of the above constraint is that the circles do not overlap. We will assume that this is always the case. The angle is given by

$$\sin(\gamma) = \frac{2\,R\,\Delta x + \Delta y\,D}{L^2}. \tag{5.25}$$

For the points P and Q we compute

$$P = A + R \begin{pmatrix} \sin(\gamma) \\ -\cos(\gamma) \end{pmatrix}, \tag{5.26}$$

$$Q = B + R \begin{pmatrix} -\sin(\gamma) \\ \cos(\gamma) \end{pmatrix}. \tag{5.27}$$

The angular velocities are $\omega_A = \bar{v}/R$ and $\omega_B = -\bar{v}/R$.

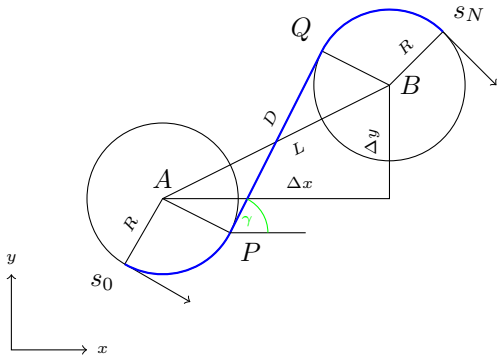


Figure 5.6: Geometry for the case left-right. The blue line depicts the Dubins path.

Case right-left The similarities to the case left-right are obvious. For A and B we get

$$A = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + R \begin{pmatrix} \sin(\alpha_0) \\ -\cos(\alpha_0) \end{pmatrix}, \tag{5.28}$$

$$B = \begin{pmatrix} x_N \\ y_N \end{pmatrix} + R \begin{pmatrix} -\sin(\alpha_N) \\ \cos(\alpha_N) \end{pmatrix}. \tag{5.29}$$

Also the angle changes to

$$\sin(\gamma) = \frac{-2\,R\,\Delta x + \Delta y\,D}{L^2}, \tag{5.30}$$

and

$$P = A + R \begin{pmatrix} -\sin(\gamma) \\ \cos(\gamma) \end{pmatrix}, \tag{5.31}$$

$$Q = B + R \begin{pmatrix} \sin(\gamma) \\ -\cos(\gamma) \end{pmatrix}. \tag{5.32}$$

The angular velocities are $\omega_A = -\bar{v}/R$ and $\omega_B = \bar{v}/R$.

5.3.2 Calculating the Inputs

We know now the path that the robot drives and it can be shown by geometrical considerations that for any two combinations of start state and terminal this path exists. We must ensure that the robot reaches its target in the given algorithm sampling time T_s. Since there is a maximum speed on the straight v_{max} and on the circles v'_{max}, we have a lower limit on T_s, which needs to be set for the used algorithms. The sampling time needs to be chosen such that the robot can achieve all possible orientations with a given speed and step length. This is especially important for Stochastic Localization. To find the minimum allowed sampling time with a given step length we must consider the worst case. The worst case occurs when the length of the trajectory is maximal, which is the case when the angle of the terminal state is shifted by 180 degrees from the starting state. Figure 5.7 illustrates the worst case.

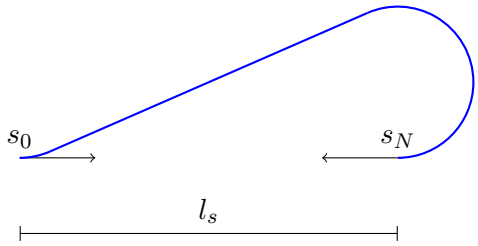


Figure 5.7: For Stochastic Localization, the longest Dubins path occurs when two states face each others.

The cases left-right and right-left which to the same total path length in the worst case. We will now calculate the path length using the left-right case. For simplicity, we assume $s_0 = (0, 0, 0)^T$ and $s_N = (l_s, 0, \pi)$ where l_s is the step length specified from the algorithm. Thus we have $\Delta_x = l_s$ and $\Delta_y = 0$ and consequently $L = l_s$. With equations (5.24) and (5.25), we calculate

$$\sin\gamma = \frac{2R}{l_s}, \tag{5.33}$$

$$D = \sqrt{l_s^2 - 4R^2}. \tag{5.34}$$

The length on the circle is equal to $c = c_1 + c_2 = R\gamma + R(\gamma + \pi)$. For example, let $R = 2.65\,\text{cm}$ and $l_s = 10\,\text{cm}$, then we have $c = 11.28\,\text{cm}$ and $D = 8.48\,\text{cm}$. The minimal time is achieved when the robot drives with maximal speed,

$$T_{s,min} = \frac{c}{v'_{max}} + \frac{D}{v_{max}}. \tag{5.35}$$

with the afore given values we calculate $T_{s,min} = 2.41\,\text{s}$. The controller calculates the circle and straight distances ahead and determines the inputs for the three sections of the path. This lower limit is only important for Grid Search, Line Search and Stochastic Localization, since for Metropolis-Hastings and Simulated Annealing the sampling time will always be choose such that the robot reaches its target with the given speed of the algorithm.

Unfortunately, Grid Search, Line Search and Stochastic Localization usually tell the vehicle to go straight. If the vehicle drives straight, the length of the path is minimal thus the velocity will also be small. In fact, continuing with the above example, the speed when driving straight is $v = \frac{l_s}{T_s} \approx 4.14\,\text{cm/s}$.

5.4 Error Analysis

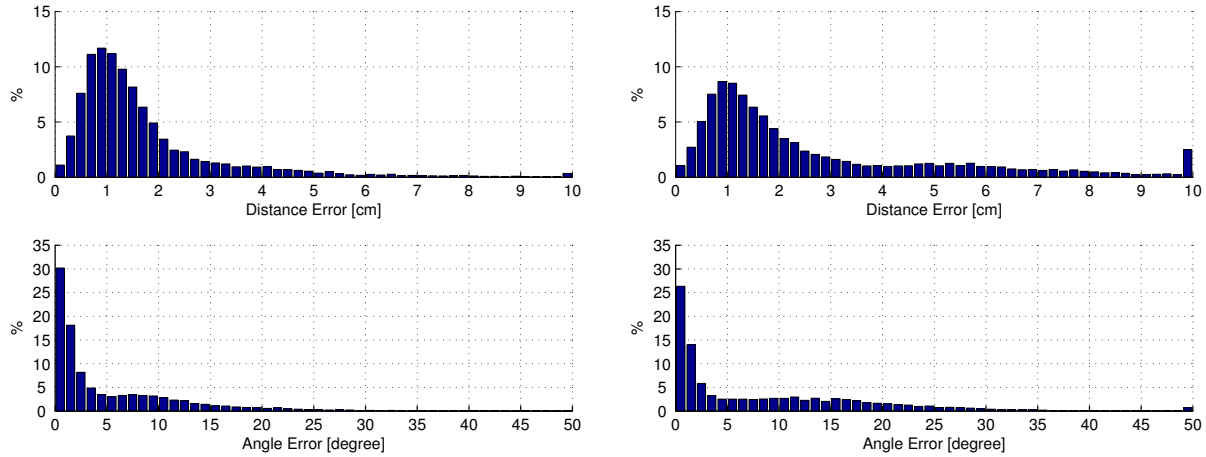
Having implemented the whole system according to Figure 5.1 we are naturally interested of how well it performs. We rate the performance in terms of position deviation and angle deviation between the calculated state s_k and the estimated state $\hat{s}_k$ after the robot has driven there. We make the distinction between position error e_d and angle error e_α.

$$e_d = \left|\left| \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} - \begin{pmatrix} x \\ y \end{pmatrix} \right|\right|_2$$
$$e_\alpha = ||\hat{\alpha} - \alpha||_2$$

Figure 5.8 below depicts the distributions of e_d and e_α for two scenarios. On the left side we have the controller with the delayed inputs for the Kalman filter whereas on the right the inputs are not delayed. To produce this data we used Stochastic Localization with a step length of 20 cm. The means of the errors are given in Table 5.1 and the distributions of the errors plotted in Figure 5.8 below. For both errors, the Kalman filter without delayed inputs shows clearly worse performance, as the tails of the distributions are considerably longer, thus leading to higher error means. By implementing a delay of 120 ms we are thus able to improve the performance to some extent. This is especially important when we let multiple robots drive at the same time for collision avoidance, which depends on accurate state estimations.

	Delay = 120 ms	No Delay
e_d	1.7 cm	2.8 cm
e_α	5.3°	8.5°

Table 5.1: Means of the errors



(a) Errors with delayed Kalman filter inputs

(b) Errors without delayed Kalman filter inputs

Figure 5.8: Error analysis of the system, with and without delayed input. The data depicts the position and angle errors of the combined system (EKF estimation and low level controller). The shape of the error distributions are similar, but the peak is higher for the EKF with the delay, thus this EKF input shows less estimation errors in total. Please note that this represent both the noise from the open-loop controller as well as from the estimation.

We can think of three main error sources that explain the above results. First, since the robot drives open loop we naturally expect it to produce some error. Second, besides the noise of the measurement, there exists some systematic error in the measurement process. All the camera measurements are taken in pixels which are discrete quantities and hence cannot map the continuous state exactly. Using higher resolution would yield more precise measurements. Also, we used a linear mapping between pixels and centimeters, which is incorrect because the camera lens is round leading to the so called "Fish-eye effect". However, our camera has a special wide angle lens and the effect is barley noticeable, thus we omitted a calibration. Furthermore, the Kalman filter and algorithm sampling times are not synchronized. Consequently, it could happen that the algorithm reads out an estimate that is slightly outdated. This effect can be quantified, as we now demonstrate. Assume that the highest sampling time that occurs on the Kalman filter is $T_{ekf,max} = 300\,\text{ms}$ and the robot drives with maximum speed or maximum angular velocity. Assume further that the algorithm request a new state estimate just before the filter has produced one, then we calculate the deviation

$$\left|\left|\begin{pmatrix}\hat{x}\\ \hat{y}\end{pmatrix} - \begin{pmatrix}x\\ y\end{pmatrix}\right|\right|_2 = v_{max}\, T_{ekf,max} = 3.86\,\text{cm},$$
$$||\hat{\alpha} - \alpha||_2 = \omega_{max}\, T_{ekf,max} = 0.73\,\text{rad} = 41.77^\circ.$$

While the error on the position is tolerable, the error on the angle seems rather high. Especially if the next position depends on the current angle, this leads to a huge deviation of the predicted next position and the actual next position. This fact complicates collision avoidance when using several robots.

The above error analysis is done specifically for the hardware and software implemented on the experimental setup, an analysis of the effect on the search results is needed. Since the underlying models (2.2) and (2.4) of the algorithms in Chapter 2 already considers noise, we expect only small influence of these errors on the algorithms, which will be confirmed in Chapter 7.

Chapter 6

Stochastic Localization with Several Vehicles

So far we have only considered one vehicle executing a particular algorithm. For some algorithms one may want to use multiple robots to reduce the mission time. Especially for Stochastic Localization and Metropolis Hastings the use of several vehicles will reduce the mission time drastically, spoken in absolute terms. Furthermore, the authors of [1] and [2] had this scenario in mind and they proved that mission time can be substituted with number of vehicles to some extent. Therefore we will implement the algorithm Stochastic Localization with several robots.

In the simulations of Chapter 3 we assumed that the vehicle is a point mass that is able to turn on the spot. This simplification may be great in simulations and theoretical considerations, but it is of little value in a realistic environment. We must therefore develop some sort of collision avoidance. In the first section we describe the strategy we chose and in the second section we show how it could be further improved by selecting a different path strategy than the Dubins path.

6.1 Collision Avoidance

Each robot must reach its predefined target state from its current state in a given time T_s, while not colliding with any other vehicle. A straight forward idea would be that whenever two robots i and j need to cross their path, robot i would drive faster in the beginning and slower in the end, while the robot j does the opposite. However, this solution is not applicable to our setting. Since we first had a step length of $l_s = 10\,\text{cm}$ for the algorithm in mind, we encounter a problem: There is simply not enough space to perform this maneuver. The robot itself has a diameter of $R_{robot} = 4\,\text{cm}$ (the robot is slightly larger than its wheels are away from each other). Therefore we could not let drive robot i with the Dubins path (see Section **??**) because the other robot j would block its path, hence we must develop a different mechanism.

First, we must think of the target states. Obviously, two target states must not be closer than $2\,R_{robot}$ to each other. Moreover, the robot must turn on circle B to its final state. This requires additional space. Remember that we specify at the current state s_k where the robot will be in two steps. (The next state s_{k+1} is already defined by θ_k and we choose the next angle θ_{k+1}, hence determining the position x_{k+2}). But, the most important fact is that we do not yet know the angle θ_{k+2}. Since we do not interfere with the paths of the robots from s_k to s_{k+1}, our collision avoidance routine must take place in the step from s_{k+1} to s_{k+2}, thus we must choose

θ_{k+1} in a favorable way. The actual paths from s_{k+1} to s_{k+2} are unknown, hence we must assume a worst case scenario. The worst case occurs when two robots facing each other must do turns by 180 degrees. To ensure that they do not crash into each other, the target states must have a distance of at least $d_{target} = 4\,R + 2\,R_{robot}$, where R is the radius of the Dubins path. The first condition is thus

$$\|x^i_{k+2} - x^j_{k+2}\|_2 \geq d_{target} \qquad \forall\, k, \forall\, i \neq\ j,\ i, j \in N, \tag{6.1}$$

where k is the current step and N is the set of all robots. This is a very strong condition, especially when R is large and l small. For our robots, we made the assumption that the individual wheels cannot drive backwards, leading to $R = 2.65\,\text{cm}$. This gives us a target distance of $d_{target} = 18.6\,\text{cm}$.

The second requirement is that no robot shall cross the path of another one. By construction of (6.1) the robots cannot cross at the beginning or the end of their trajectories, since there they are at least d_{target} from each other away. Thus, we must develop a routine that ensure that there are no crossings in the middle part of the paths. We do not completely rule out the possibility that two paths cross, because we also need to think about the time. That is, we allow crossings of paths if they cross at the beginning of one path and the end of the other path. This is fine because the two robots will not be there at the same time. Since we do not know the actual path yet, we consider a straight line from x_{k+1} to x_{k+2} for the crossings. The idea is that the circular movement is contained in condition (6.1). For this time-reason, we do not consider the full line, we leave out the first quarter.[1] Please note that we did not exactly calculate which line segment at the beginning could be left out. The quarter we use is an empirical value, of course it would work too the full line, but then more positions would be considered invalid. Again we need to take into account the dimension of the robots, that is instead of looking at a straight line, we need to consider a rectangle E with height $h = 2\,R_{robot}$ and width $w = 3/4\,l_s$. Therefore the second condition is

$$E_i \cap E_j = \emptyset \qquad \forall\, i \neq\ j,\ i, j \in N. \tag{6.2}$$

Figure 6.1 demonstrates our collision avoidance strategy. The dashed lines depict the rectangles and the circles with radius $r = \frac{d_{target}}{2}$. In the scenario from the figure, we can either say that the states x^i_{k+2} and x^j_{k+2} are not valid, or that the angles θ^i_{k+1} and θ^j_{k+1} are not valid.

[1] In addition, some line segment at the end could be left out too, because this area will be taken care of the first condition.

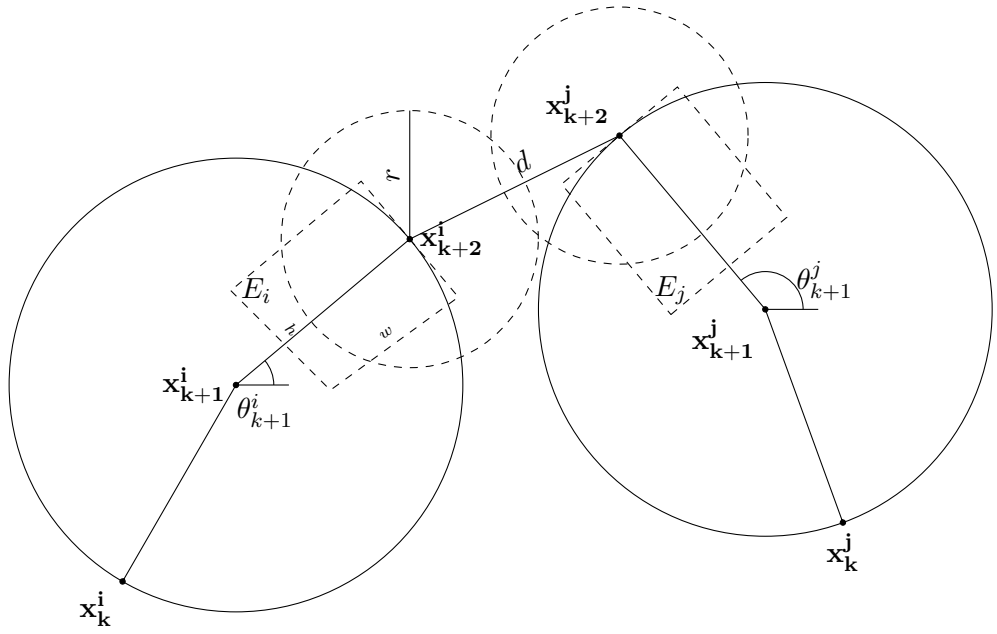


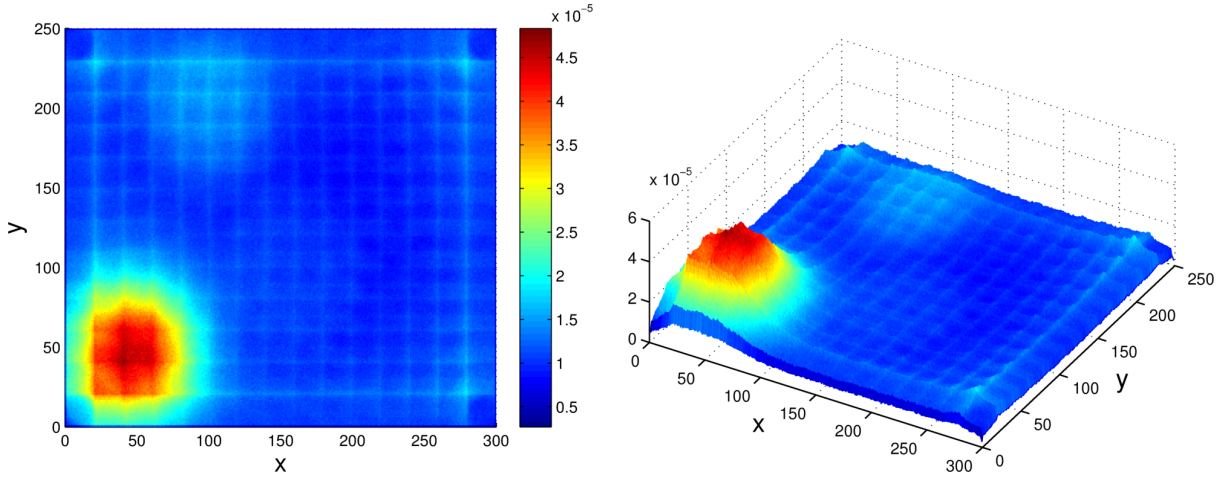
Figure 6.1: Explanation of the collision avoidance strategy for Stochastic Localization. The routine takes place in the step from x_{k+1} to x_{k+2}. The two end positions x^i_{k+2} and x^j_{k+2} should not be closer than $2r$ to allow any type of rotation of the robot, and the two rectangles E_i and E_j need to be disjoint to prevent path crossings of two robots. In the depicted case the combination of the angles θ^i_{k+1} and θ^j_{k+1} lead to invalid end positions and are therefore also invalid.

Please note that condition (6.1) is usually determining if two states are valid. However, condition (6.2) also has its use. For example, consider the case when $s^i_{k+1} = (0, 0, 0)^T$ and $s^j_{k+1} = (18.6, 0, \pi)^T$ and the step length is 20 cm, thus the second next positions are $x^i_{k+2} = (20, 0)^T$ and $x^j_{k+2} = (-1.4, 0)^T$. Condition (6.1) allows these states, because $\|(20, 0)^T - (-1.4, 0)^T\|_2 = 21.4 > 18.6$. If both robots do not turn, i.e. $\theta_{k+2} = \theta_{k+1}$, then they obviously would crash. Therefore we included the second condition which prevents this case.

What are the implications of this strategy on Stochastic Localization? Remember that we intended to use a step length of 10 cm. The target distance of 18.6 cm is then considerable larger than the step length, which will have a negative effect on the algorithm. To counteract this effect we need to increase the step length to $l_s = 20$ cm, so that we have $d_{target} \approx l$ instead of $d_{target} \approx 2l$.

First, we increase only the step length to 20 cm. Simulations with $J = 2.8$ and $K = 3.25$ have shown poor stationary distribution and success rates, so we had to adjust them to the new step length. We found $J = 4$ and $K = 10$ to yield good results, in Figure 6.2 below the stationary distribution with a bin size of 1 cm for higher accuracy is shown. If we compare this to the original simulation in Figure 3.21 notice several changes. Most obvious, the shape of the maximum peak has changed while the magnitude is of the same range. It is now less steeper than before, because we had to adjust J and K. More important though it is still clearly identifiable. Also some local maxima can still be detected. Unfortunately, the grid effect leads to a distortion of the global maximum. The true global maximum lies at $(50, 50)^T$ and the grid effect creates a local maximum at $(40, 40)^T$. These two effects overlap and thus create a global maximum at $(40, 40)^T$. However, since we have a bin size of 10 cm in the implementation and simulation we are given a tolerance of $\sqrt{2} \cdot 10$ cm from the true maximum to the found maximum. Hence we

will still consider this results as a successful result.

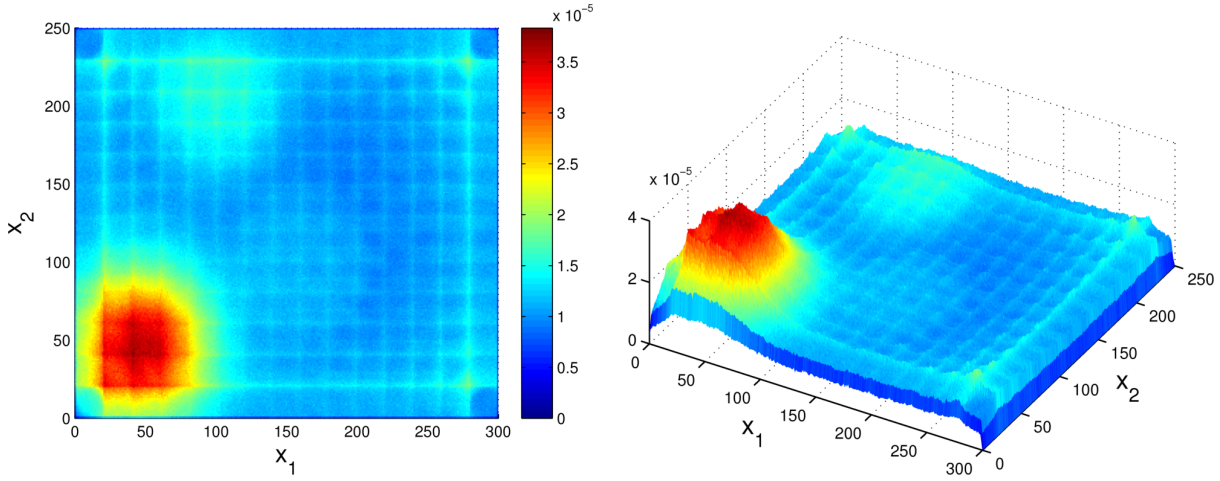


(a) Plane View **(b)** 3D View

Figure 6.2: Stationary distribution of Stochastic Localization with 1 robot, (l_s = 20 cm, J = 4, K=10, N = 10^8, b = 1 cm).

Second, for the real implementation we must adjust the algorithm to conditions (6.1) and (6.2). Right after we accept or reject the next angel θ_{k+1} the adapted algorithm checks whether condition (6.1) or condition (6.2) is violated. If there is a pair of robots i, j where the conditions are violated, the angles θ^i_{k+1} and θ^j_{k+1} are changed until the conditions are met, no matter if one proposal was initially accepted or rejected. If we would only alter the angles that were accepted, we could create a deadlock for a robot where it would not have a valid angle, since all positions would be blocked by the robots that rejected. We expect this strategy to not alter the results, because one could model this strategy into a different acceptance criterion, which still satisfies the constraints given in Section 2.7.

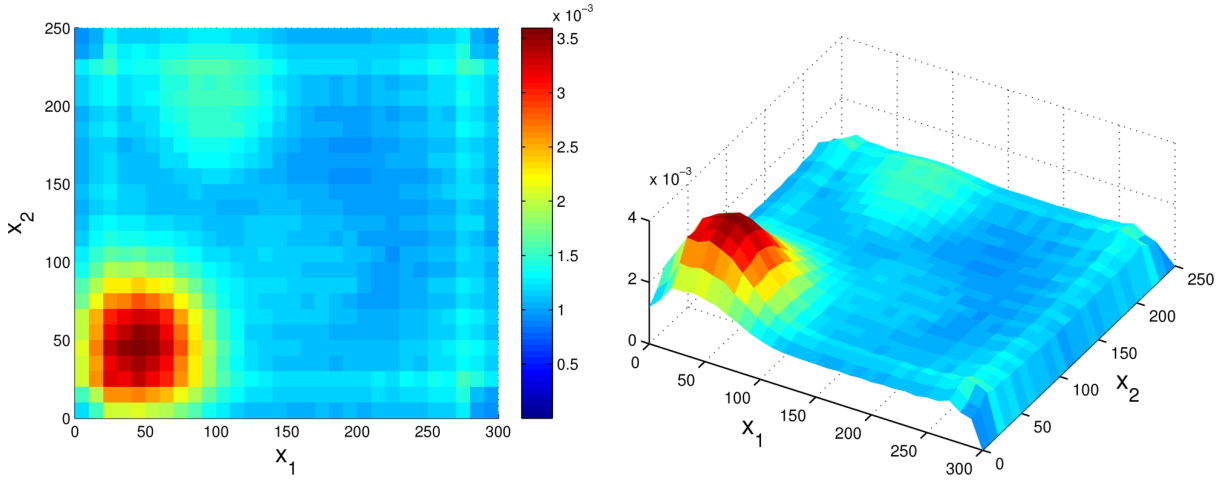
We plotted again the stationary distribution of the adapted algorithm (Figure 6.3). The distribution looks similar to the pure distribution above, with the sole difference that the noise level is higher and therefore the maximum peak is lower but still clearly identifiable. Thus we conclude that these adaptions are valid.



(a) Plane view **(b)** 3D View

Figure 6.3: Stationary distribution of adapted Stochastic Localization to 7 robots (l_s = 20 cm, J = 4, K=10, N = 10^8, b = 1 cm). The level of the maximum is lower and therefore the distributions seems to be noisier than the one produced with one robot.

In order to further prove that the adapted algorithm converges and to show that the grid effects becomes negligible when using a bin size of 10 *cm*, we increased the bin size for the same data as in Figure 6.3 to 10 cm and plotted the resulting distribution in Figure 6.4 below.



(a) Plane view **(b)** 3D View

Figure 6.4: Stationary distribution of adapted Stochastic Localization to 7 robots (l_s = 20 cm, J = 4, K=10, N = 10^8, b = 10 cm).

A remark: Normally the distribution of an proposed angle is uniform. With the above technique, this is not longer true if two or more robots get close to each other. In particular, if two robots are closer than 58.6 cm (= $2 \cdot 20$ cm + 18.6 cm) they interfere with each others angle distribution. In a more general perspective we confirmed that the angle distribution need not be uniform, but an analysis of number of encounters would be necessary to judge the performance of the algorithm, for bigger a space the observed effect should tend to vanish.

6.2 Optimal Path Problem

Since the Dubins path determines the target distance, and thus takes influence on the results, another strategy of how to move the robot would be favorable if we want to reduce this distance. We can achieve this if we select the path with minimal distance. We investigated how much one would gain from solving the resulting optimal control problem instead of using the rather simple Dubins path approach.

First, let us state the optimal control problem in continuous time. The cost function to be minimized is the length L of the path

$$L(s, \overline{v}, \omega, T) = \int_0^T \left\| \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \end{pmatrix} \right\|_2 \mathrm{d}t \tag{6.3}$$

with the system dynamics

$$\dot{s} = f(s, \overline{v}, \omega) = \begin{pmatrix} \overline{v}(t) \cos(\alpha(t)) \\ \overline{v}(t) \sin(\alpha(t)) \\ \omega(t) \end{pmatrix}. \tag{6.4}$$

We can simplify (6.3)

$$\begin{aligned} L(s,\overline{v},\omega,T) &= \int_0^T \left\| \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \end{pmatrix} \right\|_2 \mathrm{d}t \\ &= \int_0^T \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2}\,\mathrm{d}t \\ &= \int_0^T \sqrt{[\overline{v}(t)\cos(\alpha(t))]^2 + [\overline{v}(t)\sin(\alpha(t))]^2}\,\mathrm{d}t \\ &= \int_0^T \overline{v}(t) \underbrace{\sqrt{\cos(\alpha(t))^2 +)\sin(\alpha(t))^2}}_{1}\,\mathrm{d}t \\ &= \int_0^T \overline{v}(t)\,\mathrm{d}t. \end{aligned} \tag{6.5}$$

The optimal control problem then becomes

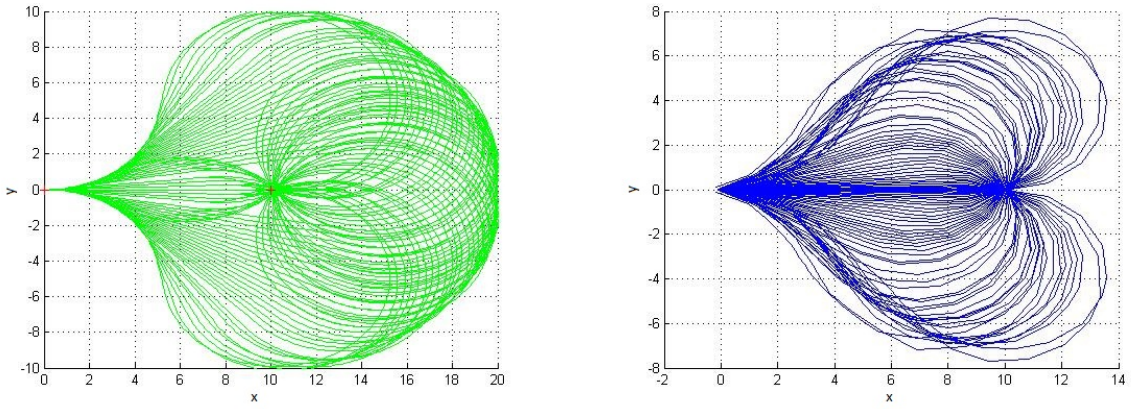
$$\min_u = \int_0^T \overline{v}(t)\,\mathrm{d}t \tag{6.6}$$

s.t.

$$\dot{s} = f(s,\overline{v},\omega), \tag{6.7}$$
$$s(0) = s_0, \tag{6.8}$$
$$s(T) = s_T, \tag{6.9}$$
$$v_{min} \leq \overline{v} \leq v_{max}, \tag{6.10}$$
$$-\omega_{max} \leq \omega \leq \omega_{max}, \tag{6.11}$$
$$0 \leq \overline{v} - \frac{b}{2}\omega \leq v_{max}, \tag{6.12}$$
$$0 \leq \overline{v} + \frac{b}{2}\omega \leq v_{max}, \tag{6.13}$$

where T is the fixed terminal time, b is the robot diameter, s_0 and s_T the initial respectively the terminal state given by the algorithm and $\overline{v}_{min}$, $\overline{v}_{max}$, ω_{max} constraints on the inputs. We have by physical limitations and designers choice that $v_{max} = 12.88\,\mathrm{cm/s}$, $v_{min} = v_{max}/5$ and $\omega_{max} = 1.683\,\mathrm{rad/s}$. The constraint on ω is chosen such that the minimal turning radius with maximal velocity is $r_{min} = 5\,\mathrm{cm}$. The last two equations specify that neither wheel can spin backwards.

We used the ACADO toolkit [25] to solve the optimal control problem. The toolkit requires a discretization of the time to solve the problem, therefore we chose 20 time steps. In order to determine the area a robot requires to turn we have plotted the paths for several terminal angles in Figure 6.5. On the left, you see the paths for the Dubins path strategy, on the right the paths for the length minimized approach. In each case, the robot starts at $(0,0,0)^T$ and drives to $(10,0,\theta)^T$ where $\theta \in [0,2\pi]$. We clearly see that the Dubins path by its definition, always turns with maximal speed and thus yields circles with a diameter of $2\,r_{min} = 10\,\mathrm{cm}$, thus needing a larger area than the optimized solution. Also we note that the optimized solution needs less space in the horizontal direction than in the vertical direction.



(a) Dubins paths **(b)** Shortest paths

Figure 6.5: Comparsion of the Dubins path and the optimized path. The robot drives from $(0, 0, 0)^T$ to $(10, 0, \phi)^T$, $\phi \in [0, 2\pi]$. The resulting paths are plotted as individual lines for different values of ϕ in order to see the area that is required for any ϕ. Please note the different scaling of the axes. The optimal shortest paths requires less space than the Dubins path.

In conclusion, to reduce the target distance a length optimized path is preferable to the Dubins path. However, solving the rather complex optimization problem needs computational resources. The E-puck hardware is limited and not able to solve this problem. This implies that we have to solve the problem on the host computer, then sending the commands. If we utilize one robot this might work fine, but if we increase the number of robots also the time consumed for the calculations increases. This leads to long time gaps between to sampling periods in which the robots drive with v_{min}, leading to huge deviations. All in all, we stick with the Dubins path strategy for the implementation on the robots.

Chapter 7

Results of the Algorithms Implemented on the E-puck

In this section we present the final results of the algorithms running on the robots. As mentioned in the previous section, we are restricted by our vision system in the choice of the region D. The largest area the camera supervises is 260 cm $\times$ 195 cm. For safety reasons we leave a small strip of 10 cm on the edges for turning maneuvers without loosing the robot. Thus we obtain the region

$$\mathcal{D} = [0, 240] \times [0, 175]. \tag{7.1}$$

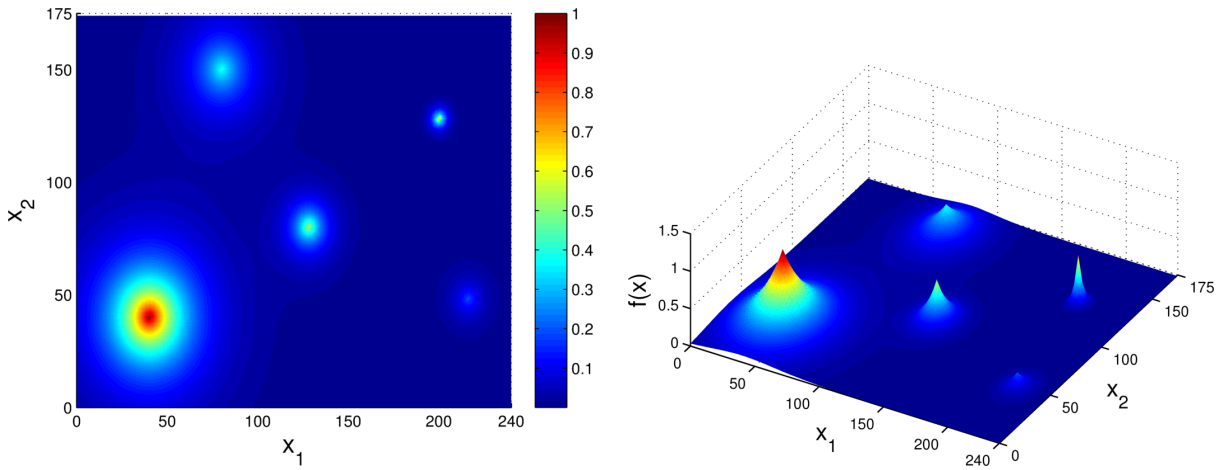
For the simulations we assume a bounded position noise of $r_D = 0.2$ cm which does not correspond to our real world scenario as we discussed in Section 5.4. To prevent the robots from leaving the supervised space, we have set the (hypothetical) position noise to 5 cm. Since the real region is smaller than the one used in the simulations, we also adapt the concentration function to

$$f(x) = \sum_{i=1}^{5} c_i e^{-\lambda_i \sqrt{(x_1 - X_i)^2 + (x_2 - Y_i)^2}} \qquad \text{for } x \in \mathcal{D}, \tag{7.2}$$

where

$$\begin{gathered} c_1 = 1,\ c_2 = 0.4,\ c_3 = 0.55,\ c_4 = 0.65,\ c_5 = 0.2, \\ X_1 = 40,\ X_2 = 80,\ X_3 = 128,\ X_4 = 200,\ X_5 = 216, \\ Y_1 = 40,\ Y_2 = 150,\ Y_3 = 80,\ Y_4 = 128,\ Y_5 = 48, \\ \lambda_{11} = 0.06,\ \lambda_{12} = 0.07,\ \lambda_{13} = 0.13,\ \lambda_{14} = 0.35,\ \lambda_{15} = 0.145. \end{gathered}$$

The plot (Figure 7.1) looks similar to TF1 (Figure 3.1a). We only compare the algorithms for concentration field (7.2).



(a) Top View **(b)** 3D View

Figure 7.1: Plots of the concentration field used for the real world implementation. The availabe search region $\mathcal{D}$ is smaller than we used in the simulations.

In addition, a small uncontrolled motion phase occurs just after the robot has reached its terminal state and before new navigation commands are sent from the host to the robots. Mainly this exists because of the time delay of the bluetooth connection, which is about 120 ms. Also before sending new commands, the host must carry out some calculations, for example evaluate the reject or accept criteria or drawing the next state from the proposal distribution. Additionally for Stochastic Localization the collision avoidance function needs to be executed. For some conditions this may take up to 500 ms (but this really is the absolute worst case). Keep in mind that we set a minimum velocity to the robot, with which the robot drives during the uncontrolled phase. This leads to a higher position noise.

7.1 Grid Search

As in the simulation the step length is $l_s = 10\,\text{cm}$ and as sampling time we choose $T_s = 2.5\,\text{s}$. (From (5.35) we calculated the minimal sampling time to be 2.41 s). When the robot drives straight, which it usually does in this case, this lead to a velocity of 4 cm/s (nr. nodes ≈ 408). In Figure 7.2 one sample path is plotted. With equation (2.9) we calculate the expected mission time to be 1122 s, but the true mission time is a bit lower (100 s) as we read from Table 7.1. This is because the vehicle did not drive all predicted grid lines because of its starting position (predicted nodes: 448, 40 nodes more than actually, $40 \cdot 2.5\,\text{s} = 100\,\text{s}$). Since Grid Search shows very similar behavior for different runs, we only made $N = 10$ runs.

Success rate	Average mission time	Average mission time when successful
100 %	1025 s	1025 s

Table 7.1

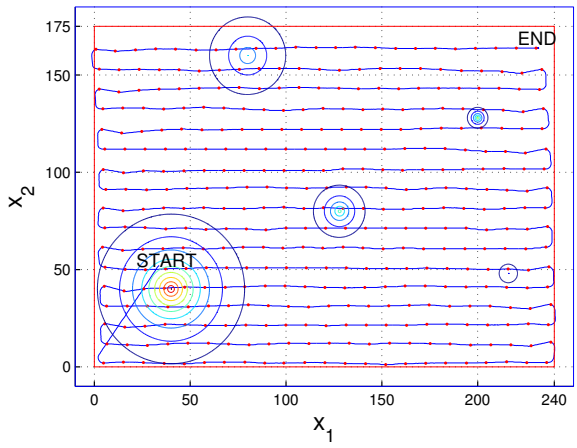


Figure 7.2: Sample path of Grid Search. Maximum: (36.52,40.38) and the mission time is 1027 s. The robots path is indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. The red box shows the edges of $\mathcal{D}$, the blue box the camera view. Also the contour of the concentration field is indicated.

7.2 Line Search

As for Grid Search, the sampling time is set to 2.5 s and the step length to 10 cm. We have adapted the parameters to the smaller region (compare to Equation (3.7)):

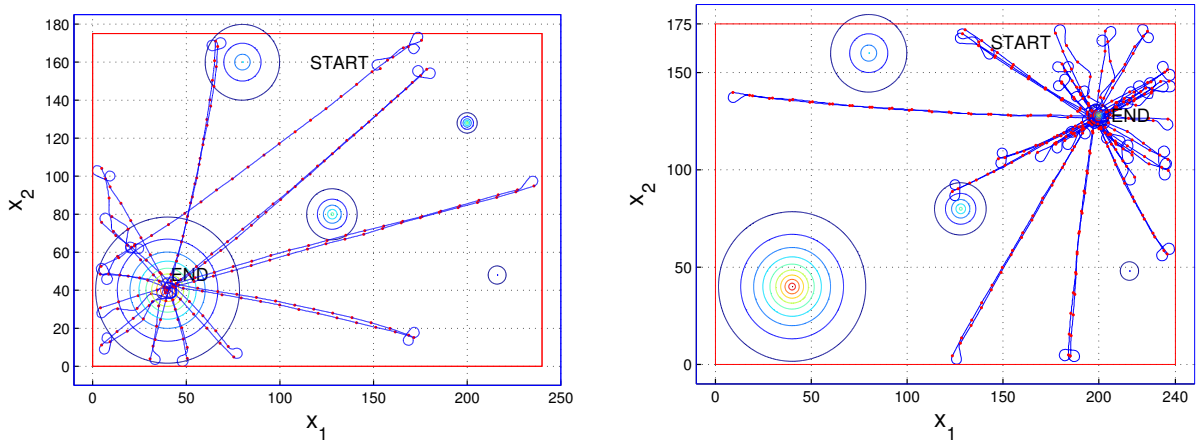
$$l_{=}300\,\text{cm},\ \gamma = 0.75^{1},\ N = 4,\ \phi = \pi/6. \tag{7.3}$$

In Figure 7.3 two sample paths out of $N = 30$ samples are depicted. On the left is a sample where the robot found the global maximum quite fast, on the right is a sample where the search was unsuccessful. Reading the values given in Table 7.2 we note that the success rate is close to the value obtain with the simulations in Section 3.1.2, but surprisingly the mission time is almost the same as for Grid Search. We think this is due to the higher (and real) position and angle noise, which leads to distortions on the angles that rotate the vehicle not exactly by 90° or 30°, which leads to more new search axes before the algorithm terminates. A small influence on the time have the additional turns that need to be made at the ends of each leg and at the new starting point. This claim is supported by a look at Figure 7.3b.

Success rate	Average mission time	Average mission time when successful
66.67 %	925 s	989 s

Table 7.2

[1] We also reduced γ because the side ratio of $\mathcal{D}$ has changed



(a) Successful run. Maximum: (40.27, 39.72), mission time: 604.4 s.

(b) Unsuccessful run. Maximum: (200.6, 127.8), mission time: 851 s.

Figure 7.3: Sample paths of Line Search. The robots path is indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. The red box shows the edges of $\mathcal{D}$, the blue box the camera view. Also the contour of the concentration field is indicated.

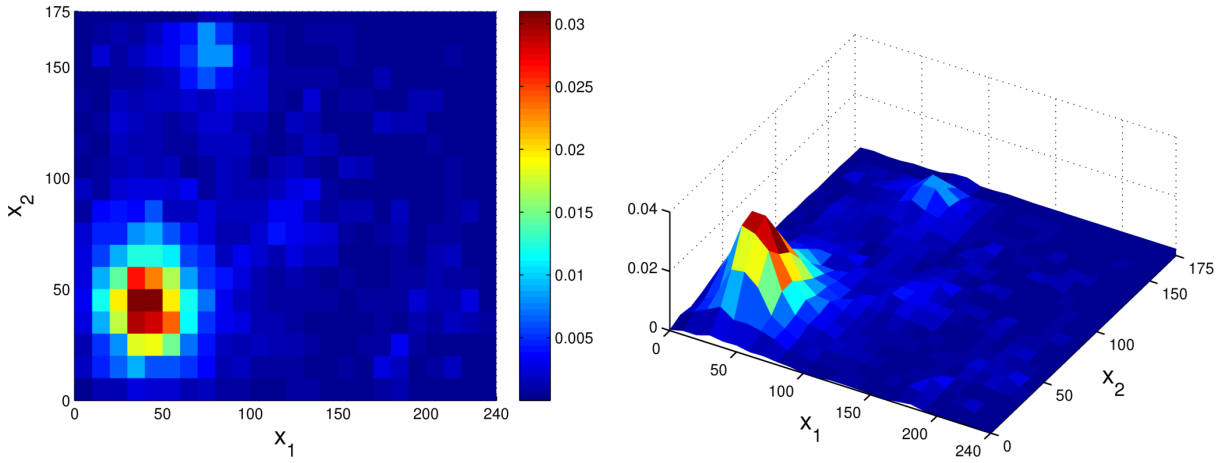
7.3 Metropolis-Hastings

As we have seen in the simulations, Metropolis-Hastings requires a long mission time. It is thus favorable if we, like for Stochastic Localization, employ multiple robots at the same time. Unfortunately, the collision avoidance strategy presented in the previous chapter is only applicable to SL, because it is based on model A (2.2). We could apply a similar strategy to Metropolis-Hastings, but we need to take into account that the step lengths (i.e. distance between current position and proposed position) are not fixed. To address this, we decided to adjust the sampling time, in contrast to adjust the velocity in order to keep similar velocities for all algorithms. Thus each robot would have its own sampling time and the number of chain iteration would begin to differ for each robot, hence the whole system would become asynchronous. To prevent the system from becoming asynchronous, every robot must operate with the same sampling time, but keep in mind that the sampling time still would vary from time step to time step according to the maximal distance from current position to proposed position of any robot. We think that in principle it would be possible to implement a multi-robot approach for Metropolis-Hastings based on the collision avoidance strategy, but maybe the proposal distribution need to be adjusted (to prevent too large step lengths).

We let the robot drive for 537 min ($\approx$ 9 h), producing a total of $N_a = 7391$ accepted moves (nr. of chain iterations $N = 17\,459$). To reduce the mission time, we let the robot always drive with maximum speed.[2] In Figure 7.4 the resulting distribution is shown. The global maximum (40,40) is clearly visible, as well as one local maxima (if you look carefully you see the maximum at (128,80) too). The other three maxima should become visible if we let the chain run for a longer time. If we are interested in the stationary distribution, we should let the chain run for $N = 10^8$, which leads to a mission time of about six years! For obvious reasons we could not produce such a run. In the simulations the chain ran for about $N = 10\,000$ iterations and we

[2] In the simulations we applied to all algorithms the speed of 10 cm/s

could generate multiple runs, but in the real world setting with the robots we only made this one run with $N = 17\,459$ for time reasons.



(a) Top view **(b)** 3D view

Figure 7.4: Sampled distribution of Metropolis-Hastings with one robot. The global maximum as well as one local maximum are identifiable. The drive time was 537 min.

In Figure 7.5 we have plotted the path of the first 400 s of the Markov chain.

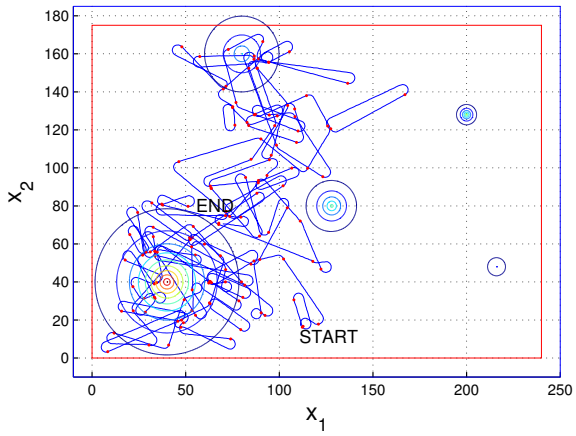


Figure 7.5: First 400 s of the above Markov chain of Metropolis-Hastings. The robots path is indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. The red box shows the edges of $\mathcal{D}$, the blue box the camera view. Also the contour of the concentration field is indicated.

7.4 Simulated Annealing

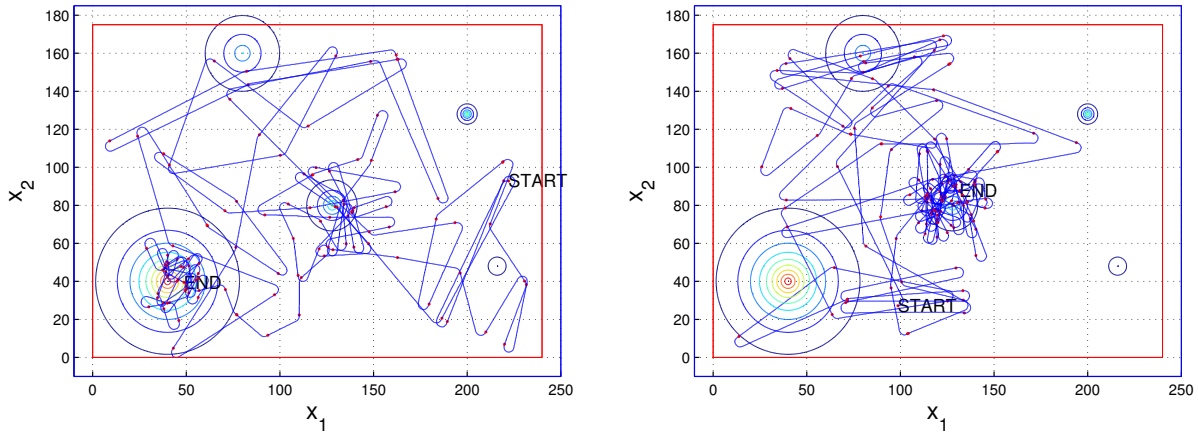
We first utilized the parameter set from (3.9), but then our trials have shown better result with the parameters

$$N_t = 12,\ T_0 = 1.8,\ \gamma = 0.7,\ R_0 = 80\,\text{cm},\ \eta = 0.75,\ N_r = 8,\ \sigma_\theta = 0.75, \tag{7.4}$$

where we have reduced the time periods N_t, lowered the initial temperature and the rate of change of the temperature γ. With these modifications we adapted the algorithm to the smaller region and further reduced the mission time. In Table 7.3 the numeric results are given and in Figure 7.6 two sample paths are plotted. We made a total of 30 runs to obtain these results. The success rate is acceptable, but the mission time is quite high, compared to the simulations and to that of Grid Search (Table 7.1). In the simulations we made the assumption that the vehicle turns on the spot in no time. Of course, this is not true and in fact, the robot wastes a lot of time in the end when exploring near a maximum with low temperature and low radius. Figure 7.6b illustrates this behavior. Another reason are the particular parameters we utilized. The mission time could be lowered on the expense of success rate if different parameters were used, for instance a lower R_0 or a lower N_t.

Success rate	Average mission time	Average mission time when successful
83.33 %	937 s	928 s

Table 7.3



(a) Successful run. Maximum: (40.09, 42.94), mission time: 935 s.

(b) Unsuccessful run. Maximum: (128.2, 80.23), mission time: 1035 s.

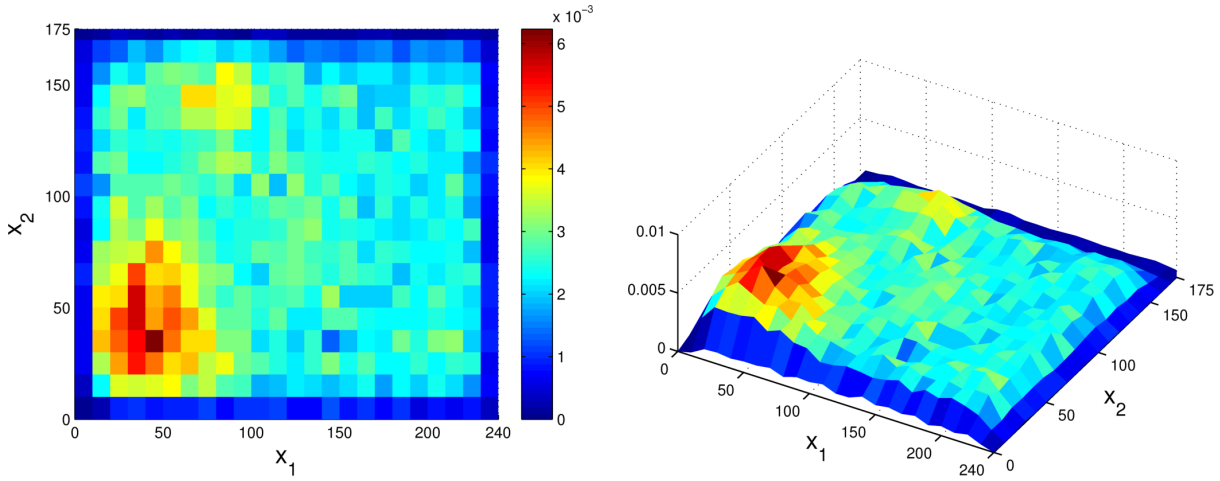
Figure 7.6: Sample paths of Simulated Annealing. The robots path is indicated by the blue line, the red dots mark the measurement instances. The starting location is marked with 'START', the terminal position with 'END'. The red box shows the edges of $\mathcal{D}$, the blue box the camera view. Also the contour of the concentration field is indicated.

7.5 Stochastic Localization

In the simulation we used one robot and we have seen that it drives for a quite long time (see Section 3.1.5). In the previous chapter we therefore adapted the algorithm to multiple robots. The number of robots we can use at the same time is limited to seven by the bluetooth technology. We have shown in the previous chapter that Stochastic Localization operates well with 7 vehicles, but the step length needs to be adjusted to $l_s = 20\,\text{cm}$ and thus also the parameters to $J = 4$, $K = 10$. With equation (5.35) we calculate the minimal sampling time to be $T_{s,min} = 3.0107\,\text{s}$. We choose a slightly higher sampling time of $T_s = 3.125\,\text{s}$, which leads to a velocity of $v = \frac{l_s}{T_s} = 6.4\,\text{cm/s}$. Additionally, we need to slightly adapt the parameter d_{target} of collision avoidance method, since in Chapter 6 we assume no position or angle noise. During our first trials with the robots and $d_{target} = 18.6\,\text{cm}$, we observed that they still occasionally bump into each other. To reduce the frequency of the crashes, we selected a larger d_{target} than calculated in Section 6.1. By setting the robot radius to $R_{robot} = 5.5\,\text{cm}$ thus $d_{target} = 21.6\,\text{cm}$, the robots crash very rarely. (In fact, for the run below, they crashed 6 times in 248 min. By even more increasing R_{robot} the crash frequency could be even lowered if not removed completely).

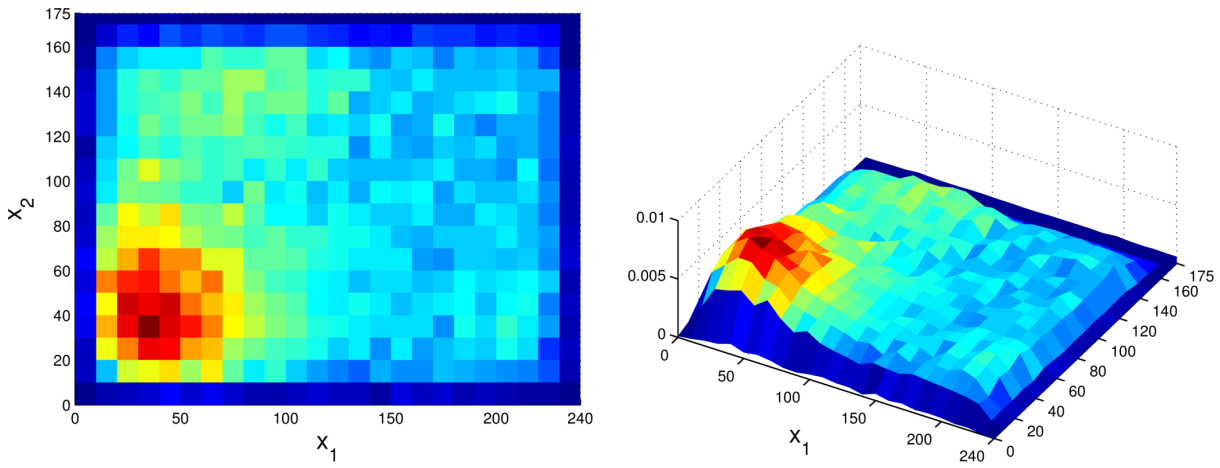
The second goal would be to produce the stationary distribution of the field. We have managed to let the Markov chain run for $N = 29\,547$ steps (4221 per robot). The total mission time is 248 min. To achieve $N = 10^8$ steps the mission time then would be around one and a halve year! The resulting distribution is plotted in Figure 7.7 and to compare it to the theory, we have plotted one simulated sample (with the same parameters and settings) in Figure 7.8. In the result from the robots, the global maximum is clearly visible and if you look closely also the second extrema is identifiable. Compared to the simulated sample, they look very similar.

With the same data we filter out each robots individual path, then we segment these paths into subpaths with a drive time of 1000 s. (i.e. we cut the Markov chains into smaller chains). Then we have a similar scenario as in Section 3.4. We obtain a success rate of $\overline{r} = 0.833$.



(a) Top view **(b)** 3D View

Figure 7.7: Simulated sample of Stochastic Localization applied to 7 vehicles. $N = 29\,547$, $J = 4$, $K = 10$, $N_{robots} = 7$,$r_D = 5\,\text{cm}$.



(a) Top view **(b)** 3D View

Figure 7.8: Simulated sample of Stochastic Localization applied to 7 vehicles. $N = 29\,547$, $J = 4$, $K = 10$, $N_{robots} = 7$,$r_D = 5\,\text{cm}$.

Note that due to the low convergence rate, the robots need to drive for a long time. The robots are powered by a rechargeable battery, which is good for about 60 to 80 min driving time. Thus we had to exchange the batteries quite often. Complicating that matter was that we only had 9 batteries and three recharge stations. Thus producing the chains was a rather tedious task, switching between driving the robots, exchanging the batteries and waiting for them to be recharged. This problem could be reduced if more batteries and recharge stations were available.

Chapter 8

Conclusion

The thesis set out to answer two main questions, 'how does Stochastic Localization perform compared to a selection of other stochastic and deterministic algorithms?' and 'is Stochastic Localization with multiple autonomous vehicles applicable to a real world setting?'.

We first carried out extensive simulations to compare Stochastic Localization to four different optimizing algorithms. Throughout the simulations we noticed that proper parameters are crucial for a reasonable performance. This is true for all algorithms. The parameters depend on the underlying function which is unknown to the operator. Thus finding optimal parameters is nearly impossible. However, we selected for each algorithm a parameter set which yields satisfactory performance with which we carried out the simulations. We used four scenarios to compare the algorithms. First, we considered two test fields to search for acceptable parameters and to compare the performance of the algorithms. We discovered that Stochastic Localization and Metropolis-Hasting have better performance in a field with steep peaks (TF1) rather than smooth peaks (TF2), while for the other three algorithms the converse is true (at least for the chosen parameters). We also noticed that these two algorithm posses an extremely long mission time, if evaluated based on the converged distributions, this making this usage not suitable for real time applications. Second, we investigated on the robustness of the algorithms. We found that all algorithms cope well with high position noise, but poorly with high measurement noise. The third scenario enlarged the search region. In the last scenario we set a time limit to the mission time and changed the method of how Stochastic Localization and Metropolis-Hastings produce results. Instead of looking at the distribution of the Markov chain, we look directly at the measured values at the visited positions. We have seen that Stochastic Localization performs better than Line Search, Simulated Annealing and Metropolis-Hastings with the limit set to be the time Grid Search requires. When decreasing the time limit, Stochastic Localization still performs best.

In a next step, we developed a robot detection software and implemented an extended Kalman filter on the host. Also we implemented a low level controller on the robots that moves the robot with a Dubins path strategy from the current state to any target state open loop. To apply Stochastic Localization to multiple robots, we developed a collision avoidance strategy. Finally, we implemented the five methods on an experimental test bed and produced samples from the algorithms on the robots.

If we had known what problems we would encounter during this project, we would change one or two aspects. First, the robot detection could be handled different. Instead of using several blobs of the same color, we could assign to each robot a different predefined color. The assignment from state measurements to robots then would be trivial, as there is only one possibility. Also,

the assignment from bluetooth communication ports to blobs would become trivial. This would be especially handy if a robot were to drive outside the camera view or if one were to be removed manually, for example to exchange the battery. One drawback is though that the colors and assignments must be hard-coded which leaves little flexibility if one were to use multiple vehicles. Already in our case finding seven different colors, that are clearly distinguishable in the software, proves difficult. Also it would fail in a different setup where GPS instead of a camera is used.

Second, Line Search, Simulated Annealing and Metropolis-Hastings could be further improved. For Line Search, it occurs that the vehicle drives multiple times along the same search axis. In a first step we already tried to reduce the occurrence of this behavior, but were not able to complete suppress it. Furthermore, the path of the vehicle could be shortened drastically, as the vehicle actually does not need to drive back to the current leg maximum to start he new leg, but it could just drive to the closer end point of the new leg. During this operation, it also could take measurements leading to a better exploration of the space. With these two adaptions, we expect the success rate to stay similar, but the mission time should be reduced. To reduce the mission time of Simulated Annealing a different terminal condition would be appropriate to prevent unnecessary exploitation at the maximum. Furthermore, as well as for Metropolis-Hastings, a different implementation of the vehicle traveling when a proposal was rejected would be appropriate. Instead of return to the last accepted position, the vehicle could immediately drive to a new proposal position.

As we implemented Stochastic Localization for seven robots, we saw the need to readjust the parameters J and K to find an appropriate set by executing simulations with different sets in a Line Search manner. This got us thinking if there is another more efficient way of optimizing the parameters. We thus tried to implement the KL-UCB algorithm from [22]. It is a form of Bandit Optimization [23]. However, we did not find an appropriate reward function to rate the performance of different parameter sets. We expected the reward function to depend on the variance of two (or more) executions of Stochastic Localization with a given parameter set. Since all chains, no matter what the initial conditions or selected parameters, converge to a unique distribution, the variance of two stationary distributions (with the same setting) is theoretically zero. What differs, is the rate of convergence. For example, a chain with a parameter setting that leads to a uniform stationary distribution converges faster than with a parameter setting that leads to a stationary distribution which reflects all peaks. Clearly, the first parameter setting is considered worse than the second setting. Since for the first parameter setting the chain converge faster, we expect the variance to be lower. Thus we cannot use the variance as a basis of the reward function.

Overall the project was successful in the sense that we compared the effectiveness of Stochastic Localization to other algorithms, and that we answered the question 'is Stochastic Localization with multiple autonomous vehicles applicable to a real world setting?' with yes.

Appendix A

Discrete Metropolis-Hastings

The real purpose of the Metropolis-Hastings algorithm is to draw random samples from a target distribution π. Following from Section 2.4, we adapt the equations to the discrete case. The subsequent two pages are an adaption from [8, Chapter 4], which we provide as background for our implementation.

Let

$$P(i,j) = \mathbb{P}(X_{t+1} = j | X_t = i)$$

be the transition kernel. The condition for reversibility becomes

$$\pi(i)P(i,j) = \pi(j)P(j,i). \tag{A.1}$$

For each pair $i < j$ we can choose either $P(i,j)$ or $P(j,i)$. The other value is then determined by (A.1). Remember that π is also known. Let us take a close look at (A.1). If $\pi(i)$ and $\pi(j)$ are both equal to zero, the condition is automatically met. If only $\pi(i)$ equals zero, then $P(j,i)$ must be zero. We are not allowed to have a positive transition probability to a state which has zero probability. If both $\pi(i) > 0$ and $\pi(j) > 0$, the transition in both ways must be possible, that means $P(i,j) > 0 \Leftrightarrow (j,i) > 0$. Furthermore we have the constraint that $\sum_{\forall j} P(i,j) = 1$. We start with an arbitrary transition matrix $Q(i,j)$ sucht that

$$\pi(i)Q(i,j) > 0 \Leftrightarrow \pi(j)Q(j,i) > 0. \tag{A.2}$$

For each pair we either set $P(i,j) = Q(i,j)$ or $P(j,i) = Q(j,i)$ and determine the other value from (A.1). We do this in such a way that both $P(i,j) \leq Q(i,j)$ and $P(j,i) = Q(j,i)$ are satisfied. Then it follows that

$$\sum_{j,j\neq i} P(i,j) \leq \sum_{j,j\neq i} Q(i,j) \leq 1,$$

and we get a valid transition matrix if we set

$$P(i,i) = 1 - \sum_{j,j\neq i} P(i,j).$$

How we choose which value to set for $P(i,j)$ depends on Q. If we set $P(i,j) = Q(i,j)$ then we must have $P(j,i) = \pi(i)Q(i,j)/\pi(j)$. This is less or equal to $Q(j,i)$ if and only if $\pi(i)Q(i,j) \leq \pi(j)Q(j,i)$. If this is not satisfied, we set $P(j,i) = Q(j,i)$ and thus $P(i,j) = \pi(j)Q(j,i)/\pi(i)$ to obtain the necessary properties. Written in compact form, we have for $i \neq j$ that

$$P(i,j) = \min(Q(i,j), \pi(j)Q(j,i)/\pi(i)) = Q(i,j)\alpha(i,j) \tag{A.3}$$

with

$$\alpha = \min\left(1, \frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)}\right). \tag{A.4}$$

Simulating according to P is easy, if we use the following algorithm. The assumption is that every new step in the chain is a sample of π.

Algorithm A.1 Discrete Metropolis-Hastings

1: Init s_0
2: Choose $j \sim Q(s_i, \sigma^2)$ and $u \sim \mathcal{U}[0,1]$
3: **if** $u \leq \alpha(s_i, j)$ **then**
4: $\quad s_{i+1} \leftarrow j$
5: **else**
6: $\quad s_{i+1} \leftarrow s_i$
7: **end if**
8: **repeat** from 2

The proposal distribution Q is of significant importance. Usually it is chosen as a Gaussian distribution with mean s_i and variance σ^2. Choosing σ^2 appropriate is crucial. A small value for σ^2 results in a slow-mixing chain, because the chain does not move fast enough through the space. A too high value also results in a slow-mixing, but because most of the proposed states will be rejected. A slow-mixing chain results in a slow convergence rate of the algorithm.

Since the chain may initially follow a very different distribution than π, because of an unfavorable starting state with low density, the first T iterations are neglected. T is called the burn-in time. Furthermore, the samples are correlated. Even though they follow π over the long run correctly, a set of neighboring samples will be correlated with each other. If we want a set of independent samples, this implies that we only take every n^{th} sample, for some value of n. The auto-correlation can be reduced by increasing the jumping width, which is equal to increasing the variance.

To demonstrate the influence of σ^2, we slightly adjust the algorithm to obtain an estimate of the target distribution. For example we choose a target distribution as shown in Figure A.1. The global maximum is at $(50, 50)$ and two local maxima at $(240, 120)$ and $(120, 200)$ exist.

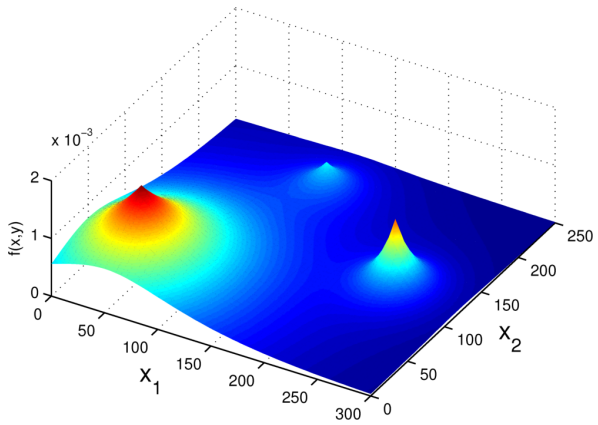
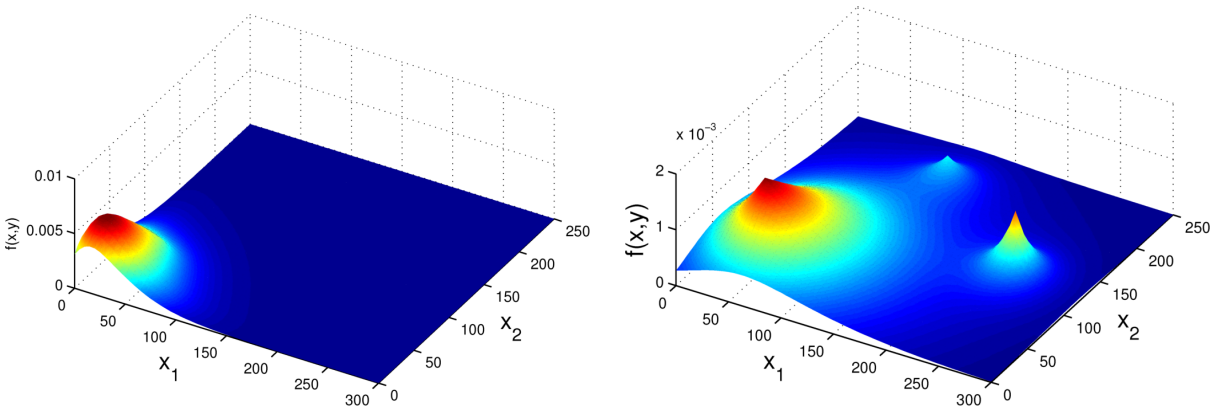


Figure A.1: Target distribution π

We choose two values for σ^2 and let the algorithm until run for $N = 10^8$ iterations were made. The space $\mathcal{D}$ is discretized in a grid with a edge length of 1. In Figure A.2 the results of

the simulation are plotted. Immediately we notice the differences. For $\sigma_1^2 = 200$ we obtain only one extrema around $(20, 20)$, whereas for $\sigma_2^2 = 5000$ all extrema exists at the correct positions. Therefore the convergence is better for a higher σ^2. In essence a high σ^2 means in this case a uniform distribution over $\mathcal{D}$. If we are only interested to locate the global maximum we can exploit the fact that a low σ^2 neglects the local maxima.



(a) $\sigma_1^2 = 200$ **(b)** $\sigma_2^2 = 5000$

Figure A.2: Sampled distributions for two different variances of the proposal distribution, $N = 10^8$.

Bibliography

[1] Stephan M. Huck, Peter Hokayem, Debasish Chatterjee, John Lygeros. *Stochastic localization of sources using autonomous underwater vehicles*. 2012 American Control Conference, 4192 - 4197, Fairmont Queen Elizabeth, Montréal, Canada, 2012.

[2] Stephan M. Huck, John Lygeros. *Stochastic localization of sources with convergence guarantees*. 2013 European Control Conference (ECC), Zurich, Switzerland, July 17-19, 2013.

[3] E. Burian, D. Yoerger, A. Braedy, H. Singh. *Gradien Search with Autonomous Underwater Vehicle Using Scalar Measurements*. Woods Hole Oceanographic Institution. Woods Hole.

[4] W.K. Hastings. *Monte Carlo Sampling Methods Using Markov Chains and Their Applications*. Biometrika, 1970, 57, 97-109.

[5] S. Krikpatrick, C.D. Gelatt, Jr., M.P. Vecchi. *Optimization by Simulated Annealing*. Science, New Series, Vol. 220, No. 4598. (May 13, 1983), pp. 671-680.

[6] D. Henderson, S.H. Jacobson, A.W. Johnson *The Theory and Practice of Simulated Annealing*.

[7] F. Romeo A. Sangiovanni-Vincenetlli. *A Theoretical Framework for Simulated Annealing*. Algorithmica (1991) 6: 302-345.

[8] H. Künsch. *Stochastic Simulation*. Seminiar für Statistik, ETH Zürich, June 2012.

[9] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller. *Equations of State Calculations by Fast Computing Machines*. Journal of Chemical Physics, 1953, 21, 1087-1092.

[10] S.P. Meyn, R.L. Tweedie. *Markov chains and stochastic stability*. Springer-Verlag, 1993, London. MR1287609

[11] S.P. Meyn and R.L. Tweedie. *Computable bounds for convergence rates of Markov chains*. Ann. Appl. Prob., 1994, 4, 9811011. MR1304770

[12] G.O. Roberts, Jeffrey S. Rosenthal. *General state space Markov chains and MCMC algorithms*. Probability Surveys, Vol. 1 (2004) 20-71.

[13] G.O. Roberts and J.S. Rosenthal. *Markov chain Monte Carlo: Some practical implications of theoretical results (with discussion)*. Canadian J.Stat. 1998, 26, 531. MR1624414

[14] G.O. Roberts and R.L. Tweedie. *Bounds on regeneration times and convergence rates for Markov chains*. Stoch. Proc. Appl. 1998,80, 211229. See also the corrigendum, Stoch. Proc. Appl. 91 (2001), 337338. MR1682243

[15] J.S. Rosenthal. *Convergence rates of Markov chains.* SIAM Review37, 1995, 387405. MR1355507

[16] OpenCV v2.4.3. *www.opencv.org*, viewed on 27. Sept. 2013.

[17] CvBlobsLib v83. *opencv.willowgarage.com/wiki/cvBlobsLib*, viewed on 6. June 2013. *http://cvblobslib.sourceforge.net/*, viewed on 28. Oct. 2013.

[18] G. Welch, G. Bishop. *An Introduction to the Kalman Filter.* Dpt. of Computer Science, Universitoy of Norht Carolina at Chapel Hill, Chapel Hill, NC 27599-3175.

[19] L.E. Dubins. *On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents.* American Journal of Mathematics 79(3):497-516, July 1957.

[20] GCtronic. *http://www.gctronic.com/doc/index.php/E-Puck*, viewed on 27. Sept. 2013.

[21] S. Magnenat. *http://en.wikipedia.org/wiki/File:E-puck-mobile-robot-photo.jpg*, 7. Oct. 2008, viewed on 27. Sept. 2013.

[22] A. Garivier, O. Cappé. *The KL-UCB Algorithm for Bounded Stochastic Bandits and Beyond.* JMLR: Workshop and Conference Proceedings vol (2011) 1-18, 25th Annual Conference on Learning Theory.

[23] A. Mahajan, D. Teneketzis. *Multi-Armed Bandit Problems.* University of Michigan, Ann Arbor, MI, USA.

[24] E-puck education robot. *http://www.e-puck.org/*, viewed on 3.Oct. 2013.

[25] ACADO toolkit. *http://sourceforge.net/p/acado/wiki/Home/*, viewed on 25. Oct. 2013.