

Bio-Inspired and Probabilistic Algorithms for Distributed Odor Source Localization using Mobile Robots

Thomas Lochmatter
thomas.lochmatter@epfl.ch

THIS PAGE IS REPLACED BY THE TITLE PAGE

February 18, 2010

Abstract

We compare six different algorithms for localizing odor sources with mobile robots. Three algorithms are bio-inspired and mimic the behavior of insects when exposed to airborne pheromones. Two algorithms are based on probability and information theory, and infer the source location by probabilistically merging concentration measurements at different positions in the environment. The last algorithm is a multi-robot algorithm based on a crosswind line formation.

The algorithms are mainly compared with respect to their distance overhead — a metric directly related to the speed of an algorithm — and their success rate. The thesis also reports on the influence of various environmental and algorithmic parameters, and compares the algorithms' requirements regarding sensors, self-localization, maps, and other information.

Systematic experiments under laminar flow conditions were carried out with real robots in an 18 m long wind tunnel. The robots were thereby equipped with an ethanol sensor and a wind direction sensor, and could — if the algorithm required it — access their current position. Overall, more than 500 experimental runs with teams of up to 5 robots were carried out in this wind tunnel.

Similar experiments were also carried out in simulation. Over 5000 runs were carried out in a realistically calibrated multi-robot simulator. Odor was thereby simulated as set of filaments that are transported by advection, an approach that generates the intermittence and stochasticity of real plumes. Additional, more than 10000 runs were carried out using body-less simulators under various plume models. Simulation runs were mostly used to quantify the influence of various parameters on the performance of the algorithms.

Finally, the thesis also provides theoretical insights into the bio-inspired algorithms, and a general theoretical model for probabilistic odor source localization. For the latter, a number of potential real-world scenarios are discussed on the example of a simplified train station environment.

None of the algorithms is strictly superior to all other algorithms. While the probabilistic algorithms offer more flexibility and a slightly better performance, the bio-inspired algorithms are much less CPU and memory intensive, and could therefore be deployed on extremely small and limited robotic platforms.

Using multiple robots (with or without collaboration) for odor source localization was found to improve the performance under certain conditions only. The crosswind formation algorithm with 3 robots yielded excellent results, but the multi-robot experiments with the bio-inspired algorithms were hardly better than their single-robot counterparts. The thesis provides reasons for this, and discusses alternatives.

Keywords: Odor Source Localization, Distributed Mobile Robotics, Wind Tunnel Experiments, Chemical Sensors, Anemometry

Zusammenfassung

Wir vergleichen sechs Algorithmen zum Lokalisieren einer Geruchsquelle mit mobilen Robotern. Drei dieser Algorithmen wurden dem Verhalten von Insekten im Geruch abgeschaut. Zwei weitere Algorithmen versuchen die Position der Geruchsquelle durch probabilistisches Kombinieren von Geruchskonzentrationsmessungen zu bestimmen. Beim sechsten Algorithmus arbeiten mehrere Roboter zusammen und bewegen sich in Linienformation gemeinsam auf die Geruchsquelle zu.

Als Vergleichskriterien dienen vor allem der Umweg welcher ein Roboter gegenüber der Ideallinie macht sowie die Erfolgsquote beim Auffinden der Geruchsquelle. Da der Umweg in direkter Beziehung zur Geschwindigkeit eines Algorithmus steht ist dies ein wichtiges Vergleichskriterium. Die Algorithmen werden aber auch in Bezug auf notwendige Sensoren, Selbstlokalisierung und weitere Informationen verglichen. Ausserdem studieren wir den Einfluss von verschiedenen Parametern auf die Leistungsfähigkeit der Algorithmen.

Die Algorithmen wurden anhand von systematischen Experimenten unter laminarer Strömung in einem Windkanal getestet. Jeder Roboter war mit einem Alkoholsensor und einem Windrichtungssensor ausgerüstet und konnte — falls der Algorithmus dies benötigte — Informationen über seine eigene Position im Raum abrufen. Insgesamt wurden mehr als 500 solcher Experimente mit bis zu 5 Robotern gleichzeitig durchgeführt.

Ähnliche Experimente wurden auch in Simulation ausgetragen. Dabei haben wir mehr als 5000 Experimente in einem realistischen Robotersimulator durchgeführt, der dafür mit einer Geruchssimulation ausgestattet wurde. Weitere 10000 Experimente wurden in Simulationen vorgenommen in welchen der Roboter lediglich als Punkt dargestellt wird. Solche Simulationsexperimente dienen vor allem dazu, den Einfluss von verschiedenen Parametern zu quantifizieren.

Diese Arbeit liefert ausserdem theoretische Resultate zur Analyse der Effizienz von Algorithmen, sowie ein allgemeines Wahrscheinlichkeitsmodell zum Lokalisieren von Geruchsquellen. Das Potential dieses Wahrscheinlichkeitsmodells wird am Beispiel eines vereinfachten Bahnhofsszenarios demonstriert.

Einen absolut besten Algorithmus konnten wir dabei nicht ausmachen. Während die probabilistischen Algorithmen eine grosse Flexibilität und eine recht gute Effizienz bieten benötigen die von Insekten inspirierten Algorithmen sehr viel weniger Rechenleistung und Speicher. Letztere könnten deshalb auf extrem kleinen Robotern zum Einsatz kommen.

Geruchslokalisierung mit mehreren Robotern gleichzeitig (mit oder ohne Zusammenarbeit zwischen den Robotern) zu betreiben verbessert die Effizienz nur bedingt. Exzellente Resultate lieferte der auf Linienformation basierende Algorithmus — die den Insekten abgeschauten Algorithmen hingegen konnten durch das Hinzufügen von Robotern nur unwesentlich verbessert werden. Wir haben Gründe dafür gesucht und diskutieren Alternativen.

Stichwörter: Geruchslokalisierung, Verteilte Mobile Robotersysteme, Elektrochemische Sensoren, Windmessung

Acknowledgments

First and foremost, I would like to thank my PhD advisor Prof. Alcherio Martinoli for his guidance and input throughout the whole PhD thesis. Working in his lab was a very motivating and enriching experience.

This work would not have been possible without the help of Xavier Raemy and Pascal Gilbert, two R&D engineers who were heavily involved in the development of the hardware tools for this project. Also involved in the electronic development were Peter Bruehlmeier and André Badertscher.

Special thanks go to Massimo Vergassola, with whom I was working during a 3-months internship at UCSB, CA, USA. During that time, I not only had several inspiring discussions on probabilistic odor source localization with him, but also shared experience and ideas with Jean-Baptiste Masson, a PhD student of him.

The idea of odor source localization with formations emerged from discussions with Iñaki Navarro, with whom I subsequently had a great collaboration on that topic.

The development of SwisTrack was a collaborative effort among a number of developers from different universities. While I developed the core part of version 4 of this software, I got substantial help and advice from Pierre Roudit, who implemented the majority of SwisTrack's core components. Basis for our development was SwisTrack 3 which had been implemented by Nikolaus Correll mainly.

I also express my gratitude to the following students, who carried out semester, summer or master projects related to this thesis:

Mehmet Erbas (2006)	Vedran Coralic (2006)
Laetitia Todesco (2006 – 2007)	Vincent Salzgeber (2006 – 2007)
Michael Devyver (2007)	Alexandre Herrmann (2007)
Saurabh Indra (2007)	Loïc Matthey (2007)
Marc-Olivier Fleury (2008)	Nicolas Heiniger (2008 – 2009)
Mitra Bahadorian (2008)	Hajir Roozbehani (2008)
Ebru Aydin (2009)	Pierre-François Laquerre (2009)

Some of these students helped exploring various aspects of the problem, while others were involved in the development. All of them contributed to the outcome of this thesis, even if not all their results are reprinted here.

Finally, I would like to thank all colleagues and friends who shared their ideas with me, or otherwise contributed with inspiring discussions about odor source localization.

Thomas Lochmatter, Lausanne, January 2010

Contents

1	Introduction	9
1.1	Odor Source Localization	10
1.2	Challenge	11
1.3	Problem Structure	12
1.4	Environments	13
1.5	Information Sources	13
1.6	Fundamental Approaches and Existing Algorithms	14
1.6.1	Gradient Ascent Methods	14
1.6.2	Bio-inspired Algorithms	16
1.6.3	Algorithms Based on Probabilistic Inference	16
1.6.4	Map-Based Techniques	17
1.6.5	Other Approaches	17
1.7	Multi-Robot Collaboration Schemes	18
1.8	Adjacent Research Fields	18
1.9	Research Questions	18
1.10	Methodology	19
1.11	Metrics	20
1.12	Contributions	21
1.13	Structure of This Thesis	21
2	Development and Setup	23
2.1	Real-Robot Setup	23
2.1.1	Odor	23
2.1.2	Khepera III Robot	24
2.1.3	Odor Sensor Board	25
2.1.4	Wind Sensor Board	28
2.1.5	Odor Source	31
2.1.6	Wind Tunnel	32
2.1.7	Wind and Odor Profile	32
2.1.8	Camera System	32
2.2	Simulation Setup	37
2.2.1	Experimental Arena	37
2.2.2	Advection Model	37
2.2.3	Odor Propagation Model	37
2.2.4	Odor Sensor Model	38
2.2.5	Wind Direction Sensor Model	38
2.3	Summary and Conclusion	39

3	Bio-Inspired Algorithms	41
3.1	Algorithms	41
3.1.1	The Casting Algorithm	42
3.1.2	Surge-Spiral	42
3.1.3	The Surge-Cast Algorithm	43
3.2	Results in Laminar Flow	44
3.2.1	Real Robots	44
3.2.2	Robotic Simulation	46
3.3	Theoretical Analysis of the Laminar Flow Performance	51
3.3.1	Wind and Plume Model	51
3.3.2	Wind Direction Sensor Model	52
3.3.3	Performance of the Casting Algorithm	53
3.3.4	Performance of the Surge-Spiral Algorithm	58
3.3.5	Performance of the Surge-Cast Algorithm	59
3.4	Results in Non-Laminar Flow and with Obstacles	61
3.4.1	Experimental Setup	62
3.4.2	Algorithms	63
3.4.3	Results	64
3.4.4	Discussion	67
3.5	Multi-Robot Experiments without Collaboration	67
3.5.1	Expected Performance of Multi-Robot Experiments	67
3.5.2	Experiments	69
3.5.3	Casting	69
3.5.4	Surge-Spiral	70
3.5.5	Surge-Cast	72
3.6	Thoughts on Collaboration	72
3.7	Summary and Conclusion	73
4	Probabilistic Algorithms	75
4.1	General Model	75
4.1.1	The Basic Model	75
4.1.2	Extensions to the Model	78
4.1.3	Optimality	83
4.2	Map-Centric Implementation	84
4.2.1	Results	86
4.3	Train Station Scenario	88
4.3.1	Modeling the Plume Propagation	89
4.3.2	Experiments	91
4.3.3	Plume Tracking with 1 and 2 Robots	91
4.3.4	Unreachable Source	91
4.3.5	Non-Uniform Prior	91
4.3.6	Surveillance and Area Coverage	93
4.3.7	Analyzing the Optimality of a Given Trajectory	95
4.3.8	Placing Static Sensors	95
4.3.9	Surveillance with Static Sensors	97
4.3.10	Source Localization with Static Sensors and Mobile Robots	98
4.3.11	Conclusion of the Train Station Scenario	99
4.4	Lightweight Robot-Centric Implementation	99

4.4.1	Complexity and Execution Time	103
4.4.2	Abstract Simulation	103
4.4.3	Robotic Simulation (Webots)	106
4.4.4	Real Robots	108
4.4.5	Similarity to Insect Trajectories	111
4.5	Summary and Conclusion	111
5	Algorithms Based on Robot Formations	113
5.1	The Crosswind Formation Algorithm	113
5.2	Experiments with 3 and 5 Robots	115
5.3	Experiments with a Moving Source	118
5.4	Summary and Conclusion	118
6	Comparison	119
7	Conclusion	123
7.1	The Robots vs. The Dogs	124
7.2	Outlook and Future Work	125

1 Introduction

Everyone of us has seen dogs sniffing around in a park to find a piece of meat, or simply to locate an item that the holder has thrown for the dog to fetch. Dogs indeed have excellent noses. With over 1000 different types of receptors, they are able to distinguish subtle differences in smell, and as their nose is orders of magnitude more sensitive than a human nose, dogs excel in finding odor sources.

That's why dogs are used in a variety of safety and security tasks. Policemen use dogs to track down drug dealers. Security staff on airports and public infrastructure use dogs to sniff for bombs or explosive material. In criminal inspection, dogs are able to follow traces of odor on inspection sites, and thereby help reconstructing what happened. And finally, dogs are also used in disaster areas (avalanches, earthquakes, and so on) to find victims or dangerous substances.

Dogs are by far not the only animals with good noses. Swines, rats, and even bees are being used to track down sources of odor [1].

This PhD thesis is not about animals, however. We instead seek to engineer a system to *replace* animals in such tasks. More precisely, we seek to build a robot (or a robotic system) that is able to find a source of odor, such as a victim buried under debris after an earthquake, or a bomb placed on an airport. Such a robotic system — if it reaches similar capabilities as a dog — would have a number of advantages over trained animals. Robots can be produced in large quantities once a model has been designed. Robots do not get tired, but run as long as their battery lasts. Robots do not have moods, but can be deployed at any time, even on very short notice. Robots do not need lengthy training (lasting several months or years with dogs), but only an initial setup and calibration procedure. Robots do not need as much care as dogs when they are not deployed anywhere for some time. And finally, if a bomb blows off and destroys a robot, this is far less displeasing as compared to a dog (and potentially its guide) being killed.

These are clear advantages which would cut the cost in existing applications, save lives in high-risk applications, and potentially allow for even more applications.

But let us foreclose one fact right away: current state-of-the-art technology is far behind the capabilities of a dog. The mobility and agility of a dog (or other animals) has not yet been achieved with engineered mobile platforms. Dogs can jump over obstacles, walk on very uneven or highly vegetated terrain, and sneak in narrow openings. They can combine visual, aural, olfactive and tactile cues to plan complicated paths through difficult terrain, and recover from almost any situation. No mobile robot to date can compete with similar properties.

In addition, even the best chemical sensing systems (electronic noses) available nowadays are still far away from the capabilities of a dog's nose. Some sensors reach the sensitivity of dog's nose [2] [3], but for a very limited range of substances. Other sensors [4] allow for discrimination of various odors, but are far less sensitive.

In both these areas (robot locomotion and electro-chemical sensing), there is a lot of research going on, and we might see interesting new technologies coming up in the next decades.

In addition to a mobile platform and an electronic nose, yet another component is needed to allow for odor source localization: an algorithm (a set of rules) that describes how the robot should move through the environment in order to find the odor source. When compared to the dog, this

would be its brain, or the “wiring” therein, and it is needless to say that algorithms are a crucial factor for finding odor sources. This PhD thesis is about such odor source localization algorithms. We compare existing algorithms — mainly in terms of speed and success rate — and present new ones that are faster, more energy-efficient, or easier to integrate into mobile robotic systems.

Besides single-robot algorithms, we thereby also explore algorithms for multi-robot systems, that is, a fleet of mobile robots collaborating to find an odor source. Collaboration, or teamwork, is an interesting aspect in our life, and in the life of many animals. Working together allows us to finish a given task in shorter time (e. g., carrying a large amount of small pieces of wood to another place), or accomplish it at all (e. g., carrying one heavy piece of wood to another place). In a similar fashion, we can make robots collaborate to decrease the time to find the source. Some algorithms even require multiple robots to work together.

Speed is indeed an important criterion in many of the applications mentioned above. In some cases, the time to find a source of odor is even more important than finding the source at all. Imagine, for instance, that you are looking for a victim buried in an avalanche. Finding the victim within 10 minutes may save his or her life — finding it after 6 hours or not finding it at all does not make a big difference. When looking for bombs (e. g., after bomb alerts), speed is even more important: once the bomb blows off, its original location is not of interest any more, or has to be determined with other means. Hence, using multiple robots can be advantageous even if the total energy spent and the total system cost are higher. Put in different words, using multiple robots allows us to “buy in” time for energy and material.

Just as with humans, working in teams has yet other advantages. If somebody falls sick (or a robot breaks down), others can jump in and take over their work. Hence, the system is more robust with respect to failure or malfunction of an individual. Furthermore, just as different people have different opinions or ways of achieving a task, different robots may use different algorithms or different sensors. This diversity makes the system more robust with respect to the environment, or changes therein.

In this thesis, we do not focus on any particular application, but compare different algorithms and comment on their performance and peculiarities. Such information is valuable for robotic system designers if their system includes odor source localization tasks. We do that by carrying out carrying out experiments with real robots in a wind tunnel, by running simulation experiments, and through theoretical models.

Let us now introduce odor source localization from a scientific point of view.

1.1 Odor Source Localization

Odor source localization¹ with mobile robots is a search problem, with the goal of finding a source that releases some chemical substance which can be measured by an appropriate sensor. Substances can range from molecules (e. g., O_3), to droplets (e. g., water vapor) or particles (e. g., smoke).

In the framework of mobile robotics, search is the problem of finding of an object (or type of object) with some given properties, usually by means of taking measurements in the environment. In our case, these environmental cues are mainly the concentration of odor at various places, and information about the wind flow.

¹Some scientist refer to the same problem as *gas source localization* [5] [6], to stress the fact that any gas could be used, not only gases perceivable by humans or animals. We prefer *odor*, as it stands for *chemical compound that can be perceived by the sense of smell*, which applies very well to artificial systems, too: our robot can only track down chemical compounds for which it has a chemical sensing system (i. e., an artificial sense of smell). *Odor* is thereby understood from the perspective of the robot, not from the perspective of a human or animal.

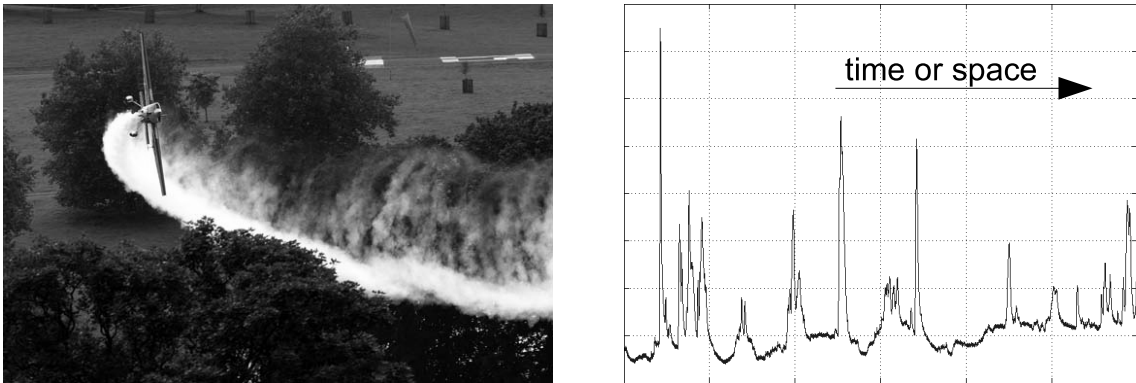


Figure 1.1: Illustration of the intermittent plume structure. **Left:** Picture of smoke and water vapor in the air, showing the packet-like structure created by turbulence. **Right:** Exemplary odor concentration in time (about 20 s) or space (about 50 cm).

Finding the source means *determining its location* and does not necessarily include *moving towards it*, even if the latter is interesting in several applications and a side effect of many algorithms. Since chemical sensing can only be done locally, measuring the concentration in proximity of the source most often gives the most precise indications of source position. Moving towards the source might not be possible in all cases, however, as sources might stand in places that are not reachable by the robot.

In some situations, the source is not even a clearly identifiable object in the environment. While a piece of luggage containing explosive material represents a relatively well-defined source object, the concept of “source” becomes fuzzy when odor is coming in through a little fissure or hole in a wall, for instance. From the perspective of somebody inside the room, one could say that this fissure is the source — indeed, if you pad the fissure, no odor will come in any more and you hence got rid of the “source”. The real source, however, is probably in another room behind the fissure.

Furthermore, a source is not a point in the environment, but can have a large physical extent. A freshly painted building is a source as a whole, for example, and the world’s oceans are the source of much of the water vapor in the air. A source therefore has a shape, and an intensity which is not necessarily uniform over the surface area.

For the remainder of this thesis, we however assume that the source is a well-defined object within the search area, and relatively small as compared to this area. Unless mentioned otherwise, we also assume that the source is reachable.

1.2 Challenge

The main challenge of odor source localization is the intermittent structure of plume in the air [7] [8]. Due to the turbulent structure (over large scales) of the air flow, plume has an irregular packet-like structure, whereby high and low concentrations are close in both time and space, as illustrated in Figure 1.1.

Often, the statistically highest odor concentrations are measured around the source. Over short time scales, however, this may not be true at all. For this reason, classical search algorithms based on the concentration gradient generally do not work unless the concentration is averaged over long times scales, which is impractical for many applications. Search algorithms from other fields (such as the localization of a sound or light source) can hardly be applied to odor sources.

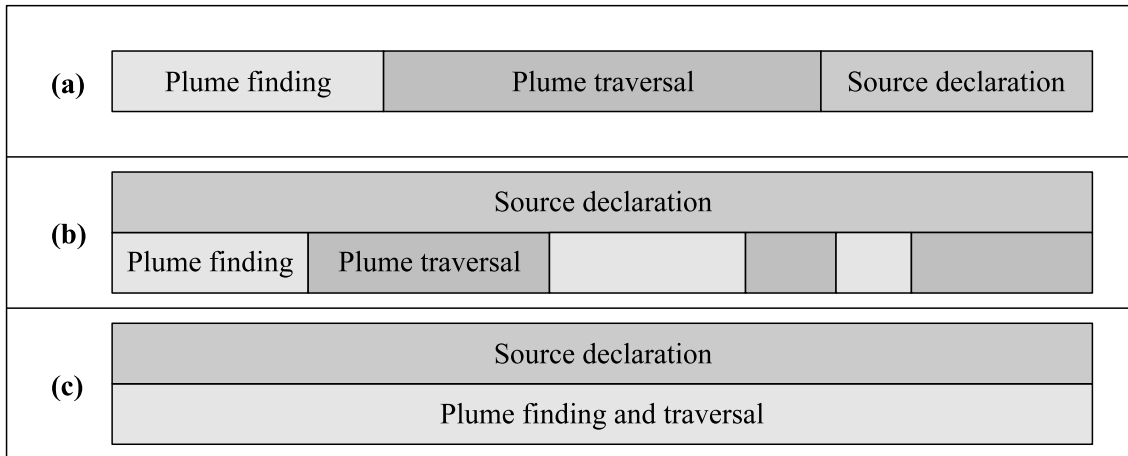


Figure 1.2: Problem structure as suggested by Hayes et al. [9] [10]. **(a)** Layman’s understanding. **(b)** Actual structure for approaches with a clear separation between plume finding and plume traversal. **(c)** Actual structure for solutions with algorithms that do not distinguish between plume finding and traversal.

1.3 Problem Structure

In simple terms, an odor source localization task consists of three parts [9] [10]:

1. *Plume finding* refers to the phase in which the robot is not in the plume, and has no (or not much) information about the location of the source. The main goal here is to find an initial cue, i. e., a first odor packet.
2. *Plume traversal* is the phase in which the robot is in the plume or close to it. In this phase, the robot tries to stay in the plume while approaching the source.
3. *Source declaration* is the decision process finding out whether the source has been reached, at which location it is, or which object exactly the source is.

While this model is very didactical and easy to understand by non-experts, it is a bit misleading as it suggests that these actions are running one after the other as shown in Figure 1.2 **(a)**. But in reality, the source declaration process is actually running all the time and not just at the end, even if it is just waiting for a potential source to show up (e. g., by comparing the measured concentration values against a threshold) for long periods of time. Furthermore, a robot may switch from plume traversal back to plume finding if it loses the plume and is unable to locally reacquire it. Hence, the diagram depicted in Figure 1.2 **(b)** is a more precise representation of the problem structure.

Yet, some plume traversal algorithms can be used for plume finding as well. An example for this is the *surge-spiral* algorithm, presented later in Chapter 3. This algorithm is primarily designed for plume traversal, but can — at least in some environments — be used for plume finding as well. Other algorithms, such as the probabilistic algorithms presented later in Chapter 4, do not even make a strict difference between plume finding and plume traversal. The same decision process is used throughout the whole tasks, no matter whether the robot has recently seen the plume or not. For solutions using such algorithms, Figure 1.2 **(c)** is an accurate sketch of the structure.

This thesis clearly focusses on the plume traversal phase, or on the combined plume finding and traversal phase. Source declaration is not covered at all, and actually a very different problem.

Recent attempts to tackle this problem have been made by Li et al. [11] [12] and Kowadlo et al. [13] and Lilienthal et al. [14], but this is clearly a domain requiring much more work.

For pure plume finding, most area coverage strategies — a field that is well understood [15] [16] [17] — can directly be applied. A more sophisticated coverage algorithm based on a plume propagation model is briefly discussed in Section 4.3.

1.4 Environments

Most work on odor source localization so far was concerned with airborne odor plumes, and so is this thesis. There has been work in other environments as well, however.

Tracking plumes in water [18] [19] [20] is similar to tracking plumes in the air. Despite the different physical properties of water as compared to air, the main transport phenomena are the same. As in air, plume in water is mainly dispersed by turbulence, and molecular diffusion only plays a marginal role. Hence, plumes in water show similar intermittency as plumes in the air. Chemical sensors as well as robotic platforms required for such applications are obviously very different.

Russell [21] [22] showed that tracking plumes is also possible in sand, where odor is mainly transported by diffusion and therefore creates a smooth concentration gradient up to the source. Hence, simple gradient-based algorithms can be used.

A somewhat similar problem is chemical trail following on a floor [23]. Some trail following algorithms are very similar to airborne plume traversal algorithms, and chemical trails on the floor are conceptually similar to static plumes (i. e., without meandering) in the air. In addition, similar chemical sensors can be used for trail following scenarios.

1.5 Information Sources

The primary information source for any odor source localization algorithm is a chemical sensor reporting the concentration of the substance that the source releases. Some algorithms require a robot to carry one such sensor, while some require at least two sensors. In one study [24], researchers even deployed 4 chemical sensors on the robot.

For airborne plume tracking, there are a number of other information sources that can support the odor source localization task:

- ▷ In environments with a main wind flow, information about this wind flow is very valuable. In the simplest case, this can be achieved by adding a wind direction sensor on top of the robot, allowing the robot to infer the direction (locally) from which the odor is coming. In more sophisticated setups, accurate information about the wind flow may be collected by external sensors, and/or simulated using CFD software [25].
- ▷ Visual information may allow a robot to come up with a list of potential sources [26] [13], or mark some areas as more probable than others (see Section 4.3.5) based on visual features. For the same purpose, laser rangefinders, depth cameras, or ultrasound range sensors could be of use.
- ▷ A map of the environment may help the robot navigate and find an optimal strategy for covering the area. In addition, maps can be used to infer information about the wind flow, and therefore the odor propagation [27] [28].

This thesis is entirely about airborne plume tracking, and focuses mainly on environments with a main wind flow. Our robots are therefore equipped with a wind direction sensor in addition to an odor sensor. The use of additional information is explored in Section 4.3.

1.6 Fundamental Approaches and Existing Algorithms

A number of specific odor source localization algorithms have been proposed in the last two decades. In 2008, Kowadlo et al. [29] published an excellent survey of the work in that area so far. They classified more than 25 algorithms with respect to several criteria in a Venn diagram. With the exception of infotaxis [30] which was published only shortly before the survey, the survey still retains its validity.

We suggest a slightly different classification of existing odor source localization algorithms here. An algorithm thereby belongs to one of the 9 approaches sketched in Figure 1.3. 7 of these approaches are further grouped into 4 main categories, which we present and discuss in the following sections.

Note that almost all works so far considered a 2D environment, and odor source localization experiments in 3D environments are still rare [31] [32] [33].

1.6.1 Gradient Ascent Methods

Despite the intermittent structure of the plume, many researchers have explored gradient ascent methods. Plume concentration is thereby seen as a noisy function over the explorable area or volume, and it is implicitly assumed that the odor source is located at the maximum of this function. The problem is then similar to optimization or function maximization, and robots are typically equipped with odor sensors only.

Techniques in this category mainly depend on the number of odor sensors:

- ▷ With a single odor sensor, biased random walk² [34] [35] [36] [37] or similar algorithms [22] [5] have been used. Such algorithms are extremely simple to implement and have very low requirements regarding sensors, CPU and memory. Their performance, however, is not particularly good.
- ▷ With two sensors on a single robot, the sensor difference can be used for robot navigation. Grasso et al. [38] and later Lilienthal et al. [39] [40] [41] carried out experiments with Braitenberg vehicles, in which both motor speeds of the differential-drive robot are functions of the instantaneous odor concentration measured by the two sensor. A similar approach is to modulate the curvature (rotational speed) of the robot as a function of the concentration difference measured by the two sensors [32] [31] [42]. Mathematically, both approaches are similar: the instantaneous wheel speeds are related to the instantaneous concentration sensor input through a (usually simple) function.
- ▷ Ishida et al. [24] applied the same concept on a robot with 4 sensors. However, only two sensors were used at a time.
- ▷ Marques et al. [37] as well as Jatmiko et al. [43] [44] [45] proposed odor source localization algorithms based on particle swarm optimization (PSO), or variants of it. Between 4 and 22 robots equipped with one sensor each were thereby collaboratively tracking down up to 5 sources in environments with and without obstacles. Experiments were carried out

²Some researchers refer to them as bacteria-inspired approaches.

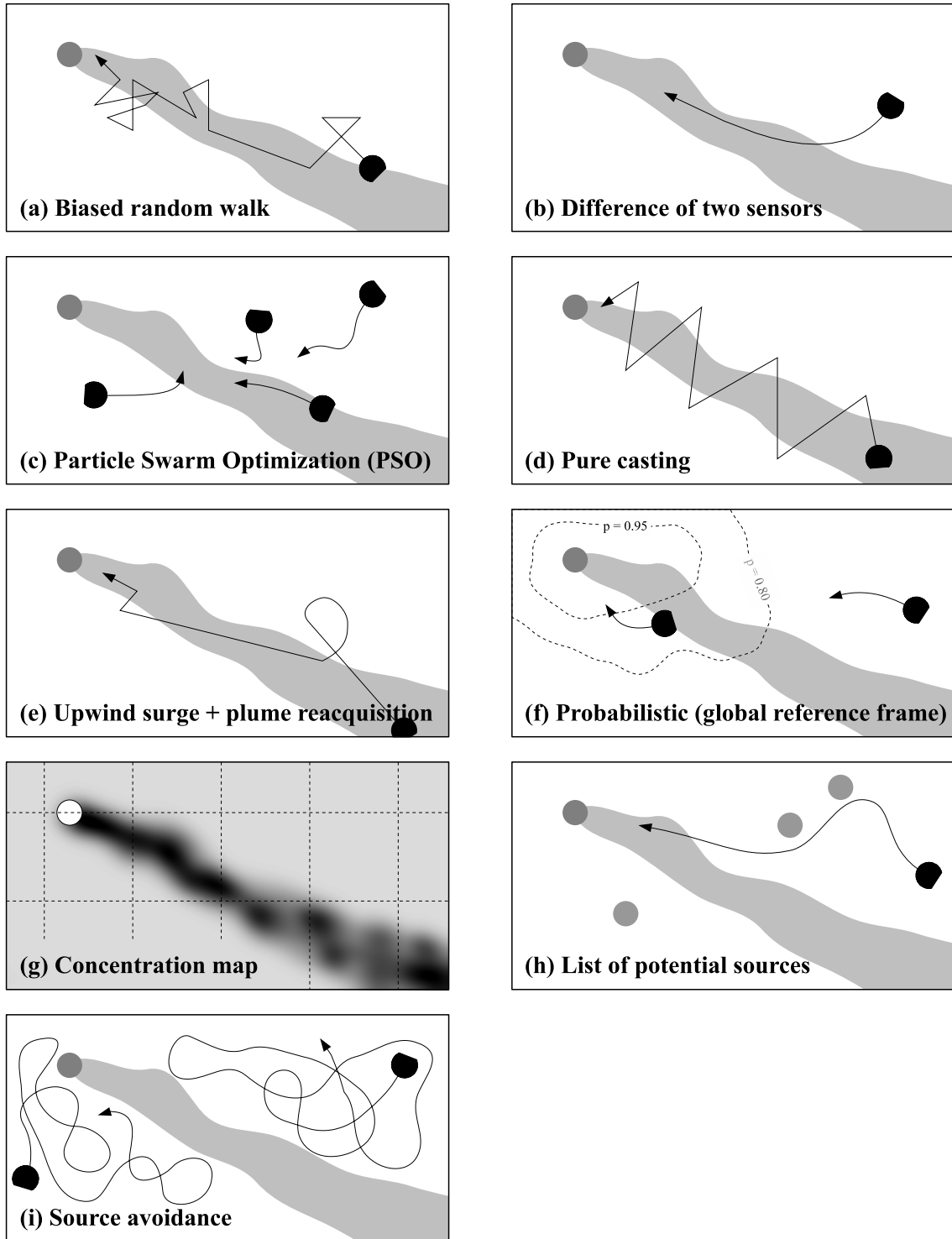


Figure 1.3: Odor source localization approaches. Almost all algorithms discussed and tested so far can be attributed to one of these 9 approaches.

with real robots and in simulation. PSO, in its origin, is a function optimization heuristic based on particles moving through the parameter space. Particles have a velocity and a position and move smoothly through the parameter space. This makes this heuristic suitable for robotic experiments, where each particle is represented by a physical robot. For many other function optimization heuristics (e. g., genetic algorithms), robots would have to jump between measurements.

1.6.2 Bio-inspired Algorithms

Another class are algorithms that try to imitate behaviors observed in nature. The ability to localize odor sources is crucial to many living species and played an important role throughout the evolution process. Hence, odor source localization strategies used by animals are supposedly highly optimized.

Most bio-inspired algorithms are based on the well-studied behavior of moths [46] [47] [48] [49] [50]. Trajectories of (silkworm) moths mainly contain three behaviors [51]:

- ▷ Upwind surge: When exposed to pheromones, the insect goes upwind.
- ▷ Casting: After having lost the plume, the insect swings from side-to-side (with increasing amplitude) for a few seconds in the hope to reacquire the plume.
- ▷ Spiraling: If the insect still hasn't found the plume, it switches to a circular motion pattern.

Based on one or more of these three rules (or variants thereof), researchers have come up with a variety of discrete-state algorithms. These algorithms are in general based on a wind direction sensor and a binary odor concentration sensor, i. e., the robot is either in the plume or out of the plume, but ignores different concentration levels. Since most odor sensors provide a smooth concentration value rather than a binary signal, their output is typically thresholded and kept high (i. e., in the plume) as long as there has been at least one above-threshold value in the past few measurements. The latter accounts for the peaky plume structure.

Some of the proposed algorithms are entirely based on casting [52] [50] [24]. Robots thereby move upwind under a certain angle until they lose the plume, and then turn back towards the plume to sweep through it in opposite direction. A number of variants of this behavior have been tested, including the use of adaptive upwind angles [53], casting around the plume boundary only [36], and version adapted for underwater odor source localization [18].

Some authors combine casting with upwind surge [54] [55] to speed up the algorithm. With these algorithms, the robots move upwind as long as they are in the plume, and only apply casting when they lose the plume.

Other algorithms are primarily based on the spiraling behavior. Spiraling itself does not require knowing the wind direction and can therefore be used in environments without a main wind flow [56] [57]. With a wind direction sensor, spiraling can be combined with upwind surge, an algorithm that has been tested by Hayes et al. [9] [10] on a multi-robot system with and without collaboration.

We discuss three bio-inspired algorithms based on the same core principles in greater details in Chapter 3.

1.6.3 Algorithms Based on Probabilistic Inference

Recently, an odor source localization algorithm called *infotaxis* and based on probabilistic inference has been proposed [30] [58]. The location of the source is thereby modeled as a probability

distribution which is derived from concentration measurements (observations) made in the environment, and the robot tries to reduce the entropy of this distribution by moving to locations for which a high information gain is expected. Such locations are at the plume boundary, or — more precisely — where the robot suspects the plume boundary to be.

With such an approach, the peaky structure of the plume is transformed into a much smoother function (in [30]: the expected entropy decrease). The navigation rule is then simply that of gradient following on this function. The function however has its optimum at locations that yield high information gains, and not where the source is located. Hence, the robot does not necessarily move towards the source, even though this holds true for most practical cases. In any case, the robot calculates a belief (probability distribution) of the location of the source.

To date, the infotaxis algorithm was tested in mathematical simulations only, and there are no reports on experiments with real robots.

Probabilistic inference for localizing odor sources have also been proposed for static sensor networks [59] [60] [61] [62].

We discuss a general probabilistic model and two derived algorithms in Chapter 4.

1.6.4 Map-Based Techniques

Several authors analyzed odor source localization algorithms based on odor concentration maps [63] [6] [64] [65]. Their results show that the source is not always at the place where the highest average odor concentrations are measured, even if such places are likely to host the source in many scenarios. A good indicator for the source location, however, is the variance of the measured samples at a given location. Close to the source, this variance was found to be significantly higher than further away — a rule that is very simple to apply to statistical concentration maps.

1.6.5 Other Approaches

Two other approaches for localizing odor sources are worth mentioning here. Both algorithms allow to localize odor sources, but are not plume tracking algorithms per se.

Kowadlo et al. suggested a combination of vision, olfaction and airflow maps for odor source localization [13]. Using the airflow map, their algorithm compiles a list of locations which are particularly useful for taking odor concentration measurements. The robot then visits these locations to record the odor concentration, and uses the resulting information to reason about the potential location of the source. In a second step, the robot uses vision to prune the list of potential sources, and finally olfaction again to determine whether an object is a source or not. The algorithm is similar to the probabilistic algorithms, although the utility function is not based on probability theory. In addition, an interesting concept found here is the list of potential odor sources: the (continuous) solution space is reduced to a (discrete) list of potential odor source locations, which can subsequently be verified with different techniques.

Another quite original concept was suggested by Lilienthal et al. [39] [40]: in their experiments, robots were programmed to avoid odor and to randomly wander around in areas of low concentration. From the robot trajectories recorded during the experiment, it was straightforward to determine regions with high concentrations, as the robot simply left them out. Such information helps inferring the location of the source in a similar fashion as the map-based techniques do, but are much easier to implement.

1.7 Multi-Robot Collaboration Schemes

Most of the algorithms discussed above are generally intended for single-robot systems, and the literature on multi-robot odor source localization is indeed sparse. To our knowledge, there exist only three approaches to date.

Hayes et al. applied a bio-inspired algorithm based on spiraling and upwind surge to multiple robots [10] [9], and studied the effect of a primitive broadcasting communication scheme. In particular, they studied a communication scheme called KILL in which all robots stop as soon as one perceives an above-threshold odor concentration, and a scheme called ATTRACT whereby robots that do not perceive any odor join others that have found plume information. Experiments were carried out with real robots and in an embodied simulation, and the performance metric was linear combination of time and group energy, the latter being proportional to the sum of the distances the robots traveled. The KILL strategy was found to save significant power, whereas the ATTRACT strategy did not reveal any advantage in their setup.

Second, all algorithms based on PSO [37] [43] [44] [45] are intended for use in multi-robot systems. PSO requires robots to communicate at least locally, and robots must be aware of each other's position. In the standard PSO algorithm, collaboration is however limited. The only variable they share is the (locally or globally) highest concentration, and the position where it was measured. As a result, robots will have a tendency to move towards the same local optimum if communication is global, and may bump into each other. Jatmiko et al. therefore introduced an extension called CPSO [44] in which robots share their positions and use a repulsive force to avoid collisions and make sure robots remain spread over some area. Robots however do not directly take concentration measurements of other robots into account.

Finally, the infotaxis algorithm has been extended to multiple robots [58]. Robots thereby share all their observations (concentrations and positions) to collaboratively infer the location of the source. All information from all robots is integrated into a single model, which is the maximum amount of information robots can share. The authors reported that in some scenarios, super-linear performance increase can be achieved by using multiple robots.

1.8 Adjacent Research Fields

Related problems to odor source localization with airborne plumes include plume modeling and simulation [7] [37], gas distribution mapping [65] or pollution monitoring [66], olfactory-guided coverage [67], odor classification [68] [69], fluid dynamics of sniffing [1], and search in general [70] [71].

1.9 Research Questions

The primary goal of our research is two-fold: First and foremost, we compare and contrast different algorithms for odor source localization, in order to assess what algorithms are suitable for a given scenario or application. Particular attention is thereby given to multi-robot algorithms. Second, we demonstrate their applicability on real robots, and formulate potential real-world limitations or caveats.

This thesis by no means intends to test and compare all existing odor source localization techniques, but it does help a designer of a future odor source localization system choose the right algorithm (or the right class of algorithms), and provide him with information about the performance he might expect, the hardware he will need, or the limitations he will have to cope with.

1.10 Methodology

To address these research questions, we are working at four different levels of abstraction between theory and application:

- ▷ **Pure theory, mathematical models:** For some algorithms, performance bounds can be derived within a mathematical model. (Example: expected performance of the casting algorithm, Section 3.3.3.)
- ▷ **Abstract simulation:** To measure the pure and idealistic performance of an algorithm (without any real-world effects), simulations are carried out at a high abstraction level. The robots are thereby dimensionless points, and interaction with the environment is implemented through simple models. Usually, such simulations are easy to set up and run very fast. Throughout this thesis, we carried out more than 10000 such runs. (Example: simulations with the probabilistic odor source localization algorithms, Section 4.3.)
- ▷ **Realistic robotic simulation:** A good estimate of the real-robot performance of an algorithm can be obtained in simulation with a realistic robotic simulator, such as Webots [72]. Robots are thereby simulated with all their sensors and actuators that interact with the simulated environment. As the simulation setup can easily be reconfigured, this abstraction level is suitable for analyzing the influence of parameters of the algorithm, the sensors and actuators, or the environment. Even physically impossible configurations (e. g., noise-free sensors) can be tested, which allows to study the impact of real-world effects on the performance. Throughout the thesis, we carried out more than 5000 such runs. (Example: simulation experiments with the bio-inspired algorithms, Section 3.2.2.)
- ▷ **Systematic real-robot experiments:** A realistic performance value is obtained by running experiments with real robots in a controlled environment, namely the available wind tunnel. The wind tunnel still allows us to control (or at least accurately characterize) a number of parameters, but experiments are done with real sensors and activators in a real plume. Such experiments provide the experimenter with very valuable hands-on experience and intuition for a specific algorithm since they point out real-world issues and limitations. Real-robot experiments are more tedious to set up and time-consuming to run. Nevertheless, we carried out more than 500 experimental runs total, and their outcome constitutes an important part for the analysis and comparison of the algorithms. (Example: real-robot experiments with the crosswind formation algorithm, Chapter 5.)

We did not carry out any real-robot experiments in realistic environments. This would have required setting up a completely new experimental site, which is a substantial investment in terms of time and money. Furthermore, we wanted to maximally exploit the wind tunnel facility³, as this is equipment that other research laboratories do not have available, and therefore best complements the work done in other laboratories. Nevertheless, we are aware that field experiments are important as a final field validation of odor source localization algorithms.

The majority of the experiments (at all abstraction levels) we carried out were in quasi-laminar wind flow at very low Reynolds numbers. We obviously cannot account for all potential environments and situations that occur in nature and carry out systematic experiments for all of them. Laminar flow is not only a simple, but a basic scenario with few free parameters. Effects such as meandering or turbulence, which could influence some algorithms more than others, are ignored.

³Note that we have by far not exploited all possibilities a wind tunnel can offer. Hence, it would have been a lost opportunity to not fully exploit it.

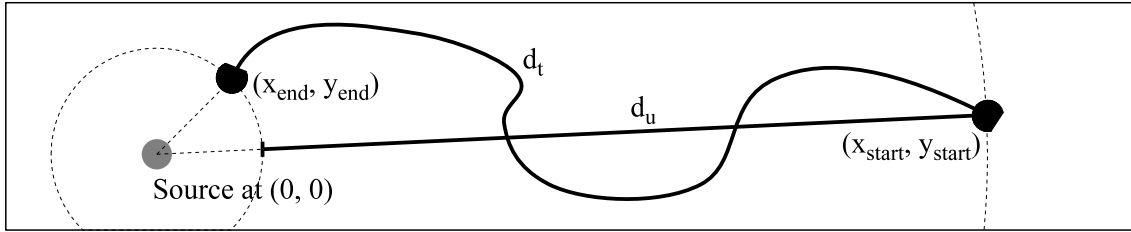


Figure 1.4: Schematic depiction of the distance overhead, $\frac{d_t}{d_u}$. Without loss of generality, the source is assumed to be at $(0,0)$. The upwind distance, d_u , denotes the distance by which the robot is closer to the source after the run, as compared to before the run. The traveled distance, d_t , is simply the length of the trajectory covered by the robot.

This is not to say that such effects should always be ignored when comparing odor source localization algorithms, but plain laminar flow is certainly the simplest environment algorithms can be tested in, before adding more complexity.

1.11 Metrics

Our main performance metrics when comparing plume tracking algorithms are the distance overhead, d_o , and the success rate, s_r . The former is calculated as

$$d_o = \frac{d_t}{d_u} \quad (1.1)$$

where d_t denotes the effectively traveled distance by the robot and d_u the upwind distance. The upwind distance is the Euclidian distance between the robot and the source at the beginning of the experimental run (minus the distance at the end of the run, in case the robot did not reach the source) and therefore stands for the shortest path a robot could take if it knew the location of the source. For a source at $(0,0)$, this can be expressed as

$$d_u = \sqrt{x_{\text{start}}^2 + y_{\text{start}}^2} - \sqrt{x_{\text{end}}^2 + y_{\text{end}}^2} \quad (1.2)$$

and is depicted in Figure 1.4. How to accurately measure d_t and d_u depends on the experimental setup, but is usually not a problem.

The advantage of the distance overhead metric is two-fold:

- ▷ d_o is independent from the downwind distance at which the robot was released as long as the plume structure (width, intermittency, concentrations) remains the same over the whole length⁴. Hence, the results of different starting positions can be compared.
- ▷ d_o is the same for a robotic vehicle with differential-drive kinematic constraints, as well as a completely holonomic vehicle. (In contrast, the time to reach the source is not: it penalizes algorithms with which robots have to quickly change the movement direction, as a differential-drive robot cannot do that in infinitesimally short time.)

The success rate, s_r , is the fraction of runs in which the robot successfully found the source. Note that — in contrast to the distance overhead — this value can only be compared if the upwind

⁴In the wind tunnel and the corresponding simulation setup, the plume profile does not change significantly over most of the length.

distance is the same. All experiments presented in this thesis were carried out with an upwind distance of $d_u \approx 14$ m.

Combining the distance overhead and the success rate to an *overall performance* would not make sense in general terms, as the weighting between the two heavily depends on the application. Imagine a system to find leaks in a gas pipes of industrial plants, for instance. If there are leaks in the pipe, we absolutely want to find them. Obviously, the faster this can be done, the better, but the primary goal is to succeed. On the other hand, a robot looking for explosive substances after a bomb alarm has been raised should find the bombs as fast as possible. Failing to find them is equivalent to being too slow, as the bombs will blow off in both cases.

1.12 Contributions

The main contributions of this thesis are:

1. A comparison of 6 odor source localization algorithms under laminar flow conditions in a systematic environment. The algorithms are compared with respect to their performance, required information, and suitability for different types of environments.
2. A probabilistic model for odor source localization, and several extensions to take constraints of the robot and the environment into account. A map-centric implementation demonstrates the potential of that model.
3. A multi-robot odor source localization algorithm based on a crosswind line formation, and its evaluation with up to 5 robots.
4. An in-depth evaluation of three single-robot plume tracking algorithms inspired by the behavior of moths.
5. A lightweight robot-centric implementation of the probabilistic model, and the observation that the resulting trajectories are similar to insect trajectories.
6. The setup of an experimental environment in a wind tunnel to carry out systematic real-robot odor source localization experiments, as well as the implementation of a Webots [72] physics plugin to simulate wind and odor.

Alongside this thesis, we also contributed to the following two open source projects:

- ▷ We implemented the core of SwisTrack 4 [73] [74], an open source visual tracking software for multi-agent systems.
- ▷ We implemented the Khepera III Toolbox [75], an open source software toolbox for the Khepera III robot.

1.13 Structure of This Thesis

The remainder of this PhD thesis is structured as follows:

Chapter 2 provides an in-depth description of the hardware and software tools developed and set up to test the algorithms with real robots and in simulation. This includes the tools used in the wind tunnel, the robots with their extension boards, as well as the odor simulation plugin developed for Webots [72], the simulation software we use. This chapter does not discuss any

odor source localization algorithms, but provides all the details of the experimental setup used for the experiments discussed in later chapters.

We then present three bio-inspired algorithms in Chapter 3. After introducing the algorithms, the thesis reports on real-robot and simulation experiments, and compares them to results obtained within a theoretical framework. We then discuss real-robot experiments carried out with obstacles (i. e., in turbulent flow). The second part of this chapter is devoted to multi-robot experiments with bio-inspired algorithms, and study the performance of a multi-robot system without collaboration. These results also allow us to make assertions about multi-robot systems employing collaboration.

Chapter 4 starts by introducing a probabilistic model for odor source localization, and subsequently presents two vastly different implementations of this model. The first implementation is a map-centric implementation for multi-robot systems (and even static sensors) which is tested in simulation only. This implementation is also applied to a simplified train station scenario which serves us to demonstrate the possibilities of the model. The second implementation is a lightweight robot-centric implementation, for which we present results obtained through abstract simulations, embodied simulations, and real-robot experiments. This part is concluded with a note on the similarity to insect trajectories.

Chapter 5 then introduces a lightweight odor source localization algorithm inherently designed for multi-robot systems. The algorithm is based on a crosswind formation and achieves excellent performance in our setup.

Note that chapters 3, 4 and 5 can be read independently from each other. The algorithms are contrasted in Chapter 6, which also states high-level advantages and limitations of the algorithms. Chapter 7 concludes the thesis.

2 Development and Setup

An important aspect of this PhD thesis is the evaluation of algorithms in simulation and with real robots. Inspired by previous experiments done in other research laboratories, we developed the necessary hardware and software tools and set up environments for both real-robot and simulation experiments.

2.1 Real-Robot Setup

Our main experimental site is an old wind tunnel at EPFL. The wind tunnel was built in the 1970ties and since then used for various wind engineering experiments and measurements. Due to the retirement of the professor in charge in 2006, we inherited this wind tunnel and can use it for robotic experiments. Many design choices in our experimental setup are directly or indirectly related to this wind tunnel facility, and sometimes trade-offs between optimality and feasibility.

Figure 2.1 shows a sketch of the experimental setup used in the wind tunnel. The wind tunnel is about 18 m long and 4 m wide. Since some space on both ends can only be accessed with difficulty, the usable length is about 15 m. An ethanol source is deployed close to the wind inlet, and the robots are starting about 14 m downwind from that point. The robots are equipped with an odor sensor (artificial nose) and a wind direction sensor.

In the remainder of this chapter, we describe the individual parts of this experimental setup, and motivate our design choices.

2.1.1 Odor

A crucial choice at the root of the whole development was the type of chemical to use for the real-robot experiments. Ideally, the substance would be visible in the air, not leave any traces on the floor, and — most importantly — be harmless to both humans and robots (electronics). The substance should furthermore be cheap, readily available, easily releasable into the air, and have neutral buoyancy therein. The latter excludes all substances that do not have the same temperature as the ambient air, as hot substances tend to rise and cold substances tend to fall. Furthermore, good chemical sensors for that substance must be available.

We considered a number of substances. Interestingly, the most difficult property to fulfill is the visibility in the air. For a particle to be visible in the air, it needs to have dimensions in the order of one wavelength of visible light, i. e., ≈ 600 nm. Particles (or droplets) of that size tend to leave traces on floor or electronics. Hence, smoke and water vapor are not suitable for our experiments. Molecules are typically much smaller and therefore not visible, but there is a vast choice of readily available and easily releasable substances on the market.

We finally decided to use ethanol (C_2H_5OH), which is harmless to humans and robots in the small quantities used. Due to the low boiling point, it hardly leaves traces on the floor or on the robot. In addition, small ethanol sensors are readily available from several companies. The only disadvantage of ethanol is that it is not visible in the air.

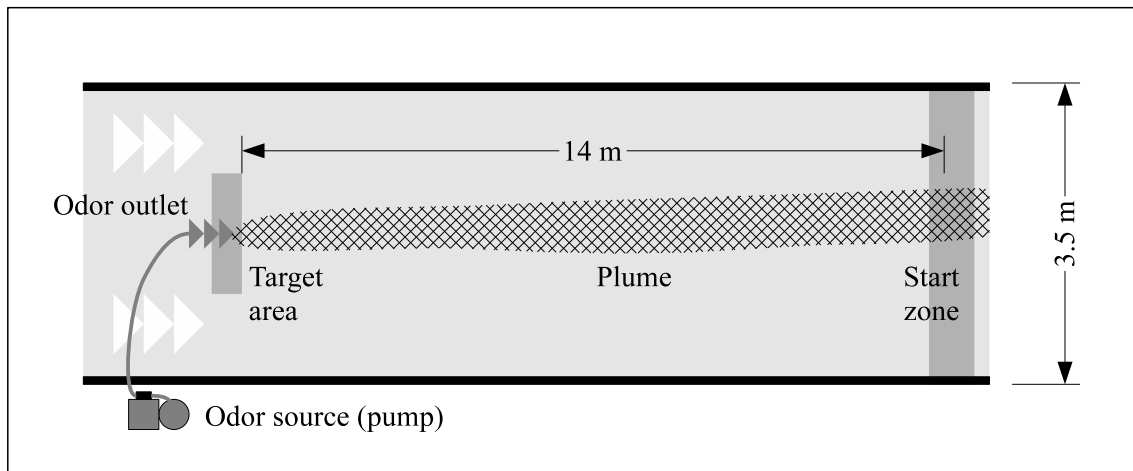


Figure 2.1: Sketch of the experimental setup in the wind tunnel.

2.1.2 Khepera III Robot

The Khepera III mobile robot (see Figure 2.2) is used for all real-robot experiments. The Khepera III is a differential-drive robot of about 12 cm diameter and 7 cm height (without any additional modules), engineered and produced by K-Team SA, Switzerland. It hosts a KoreBot board running an embedded Linux and a WLAN 802.11 CompactFlash card. The robot is equipped with two DC motors with optical encoders enabling the use of odometry on flat surfaces, 9 infrared sensors for obstacle avoidance, two infrared floor sensors, and 5 ultrasound sensors.

Other extension boards can be stacked on top. We developed two such extension boards¹ for detecting ethanol vapor and the wind direction. Both boards are connected to the main processor by means of the I²C bus, and sampling rates as seen from Linux are in the order of 150 Hz maximum.

While the bare robot running Linux has an autonomy of a bit more than 2 h with new batteries, the odor and wind modules reduce this autonomy to approximately 1 h 30 min of continuous use.

The Khepera III robot was a brand new product by the time we bought it. The first robots we received were still prototypes in early stage. We were actively involved in debugging and testing the platform, and fixed a number of issues in collaboration with K-Team. Hardware flaws were addressed by K-Team, while the software was mainly improved by us. As a result of this effort, the Khepera III Toolbox [75] was published under an open source license. This software toolbox replaces K-Teams *libkorebot* and — while solving synchronization and reliability issues — offers a much simpler and more concise API to access the sensors and actuators of the robot. In addition, a number of small programs allow to access these sensors and actuators without writing a single line of source code.

2.1.2.1 Computational Performance

The KoreBot, which is the main processing unit on the Khepera III robot, hosts an Intel XSCALE PXA-255 running at 400 MHz and 64 MB RAM. XSCALE processors are based on the ARMv5 architecture (32-bit), which offers about 400 MIPS at this speed.

In order to evaluate what we can afford in terms of algorithms, we carried out performance and power measurements with this processor. The KoreBot was thereby used on a Khepera III robot,

¹Voltage regulation for both boards was outsourced to a third board which is not further discussed in this thesis.

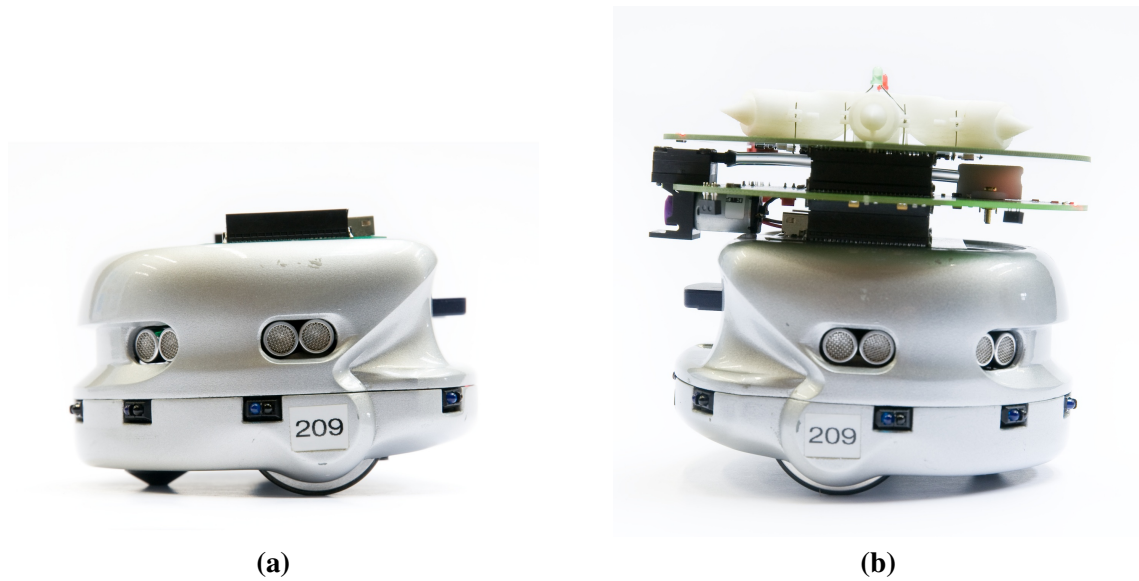


Figure 2.2: Pictures of (a) a bare Khepera III robot, and (b) a Khepera III robot equipped with an odor and a wind sensor board.

but all extension modules were removed. In addition, the motors were left uninitialized. Using the power measurement chip on the battery circuitry, we then measured the voltage and current while the processor was idle (running Linux), and while it was at maximum load. The difference is a good estimate for the power that is needed to carry out instructions.

This difference in fact includes not only the raw instruction being executed, but whatever else within the processor is required to execute it. Through this, we observed that the processor draws about 185 mW more when under load (as compared to being idle), no matter what the instructions being executed were. Note that this figure includes the efficiency of the power regulation circuitry, but not the efficiency of the battery.

We then measured the time it takes for specific instructions to complete. For that, we run these instructions in a tight loop and counted the number of instructions executed within a time frame of 15 – 30 s. The results were corrected for the time it takes to execute an empty loop.

Table 2.1 lists the execution times of additions, multiplications and divisions for different data types. (Note that these values differ significantly from processors found in current PCs, which are mostly based on the i386 or x64 architectures with floating point support.) Due to missing floating point instructions in the ARMv5 architecture, all floating point operations have to be emulated with integer operations, which takes significantly more time. 64-bit integers have to be emulated with 32-bit instructions as well, and divisions with this data type are particularly costly.

2.1.3 Odor Sensor Board

The odor sensor board is equipped with a MiCS-5521 volatile organic compound (VOC) sensor from E2V Technologies PLC, UK. The sensitive surface inside this sensor has a very fast response time (≈ 0.1 s upon exposure to ethanol, ≈ 1 s to recover in fresh air) as compared to other VOC sensors on the market, which is a clear advantage for real-time robotic applications. The sensor is not very selective, though: it reacts not only to ethanol, but also to many other volatile organic compounds and hydrocarbons (at different sensitivities). The sensitivity to alcohol — although not provided in the technical datasheet — seems to be comparable to other state-of-the-art VOC sensors, and therefore in the 1 – 10 ppm range.

Table 2.1: Time and energy required to execute an operation of a given data type on the KoreBot. The numbers are relative factors with respect to a 32-bit integer addition (minimum and maximum of 5 runs on 5 different Korebots each). A 32-bit integer addition takes about 29.31 ns to execute, and draws 5.42 nJ of energy. Both these figures can be multiplied with the corresponding factor from the table to obtain the time and energy of an operation.

Data type	Addition	Multiplication	Division
Integer (32-bit, int)	1	1.16 – 1.17	3.99 – 4.01
Integer (64-bit, long long int)	2.5 – 2.51	4.33 – 4.97	52 – 52.1
Floating point (32-bit, float)	4.66 – 4.67	6.49 – 6.51	4.99 – 5.01
Floating point (64-bit, double)	8.99 – 9.01	10.9 – 10.9	9.83 – 9.86

To take advantage of the fast response time of the sensitive surface, air is continuously drawn into the sensor using a small gas pump (NMP 05 S, KNF Neuberger). Without the pump, it would take significantly longer for the plume to enter the sensor package and reach the surface. The setup is conceptually similar to the olfaction system of humans or some animals, except that air is not exhaled through the same pipes as it enters. Note that we did not study the influence of the pump speed (or modulation thereof) in our experiments, but we believe that the impact is minor over a large speed range.

A microcontroller on the same board controls the pump speed and reads the odor sensor value. The main processor on the KoreBot can communicate with that microcontroller via the I²C bus. The sensor values can thereby be read in two modes:

- ▷ In *last value mode*, the microcontroller returns the most recent value that has been completely acquired. This is useful for programs that need to measure the current odor concentration.
- ▷ In *streaming mode*, the microcontroller returns all measured values exactly once. This is implemented with a circular streaming buffer and a simple streaming protocol, and useful for recording the evolution of the odor concentration.

The I²C interface also allows to set the pump speed and disable the LEDs on the board.

2.1.3.1 Sensor Comparison

There are several sensors for ethanol detection on the market. Before we chose the MiCS-5521 sensor for our odor board, we compared three such sensors with respect to sensitivity and speed. The three sensors are from two different companies (E2V and Figaro Engineering), and based on resistive sensor technology. All three sensors are in the same price range (approx. CHF 50.- / piece).

It should be noted that we did not have any specialized high-quality tools available for determining the sensor parameters in an accurate fashion. Our requirements were moderate as well, however: we only wanted to find out whether some sensors are significantly faster, or significantly more sensitive than others.

To achieve that, we mounted the three sensors next to each other on a breadboard, and connected appropriate heating and sensor resistors (see Figure 2.3). The sensors were powered by an industrial power supply, and their sensor pins connected to different channels of an oscilloscope. Ethanol evaporating in a small glass was blown onto the sensors with a little hand fan. To measure the impulse response, the glass cover was removed for about 0.5 s, and then put back again.

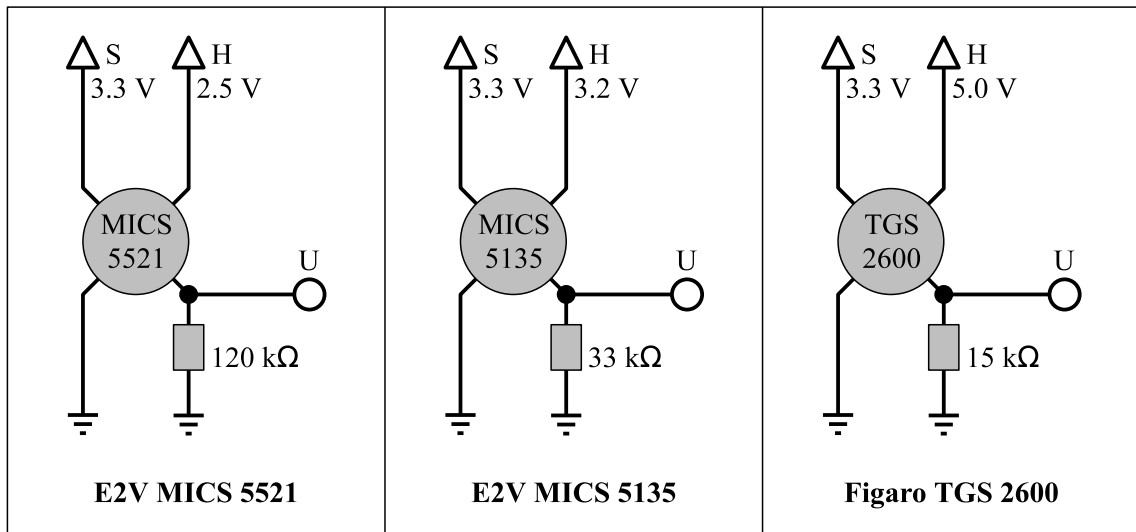


Figure 2.3: Circuits used to compare the different sensors. The measurement voltage (S) was the same for all sensors. The sensors were used in their original package (as recommended, without removing the cap), and measurement resistor as well as heater voltage were chosen according to indications from the corresponding data sheet. The voltage U was recorded with an oscilloscope.

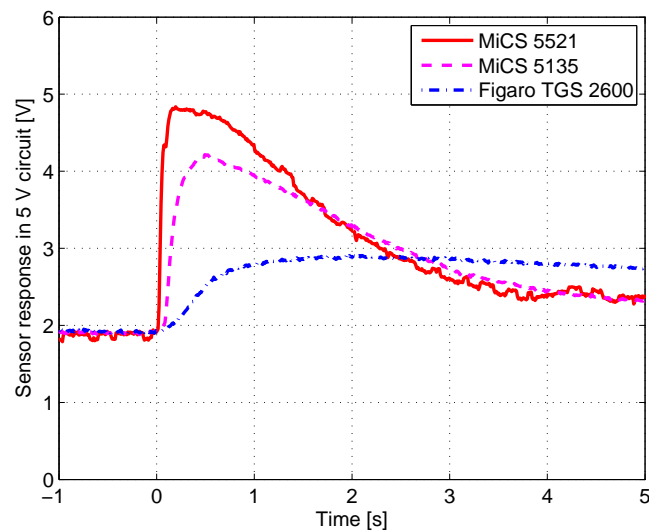


Figure 2.4: Impulse response of three different resistive VOC sensors when exposed to ethanol for about 0.5 s. While all three sensors have about the same sensitivity to ethanol, the MICS 5521 from E2V is significantly faster.

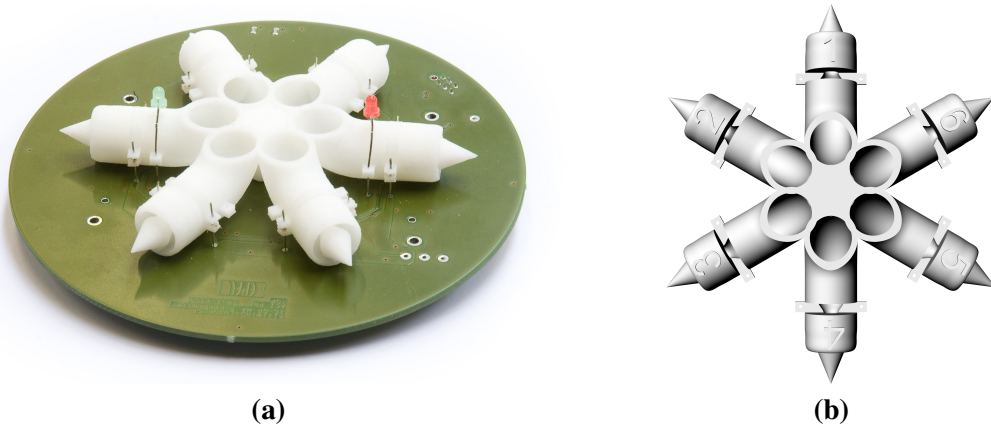


Figure 2.5: (a) Picture of the wind sensor board. (b) 3D model of the six tubes. Each tube hosts one NTC thermistor.

The results of this test are shown in Figure 2.4. The MiCS-5521 sensor proved to be significantly faster than the other two sensors, especially upon exposure. Regarding their sensitivity, we did not find significant differences.

2.1.4 Wind Sensor Board

The wind sensor board consists of 6 NTC thermistors (Honeywell 111–202CAK–H01) connected to a microcontroller. These thermistors are placed inside tubes arranged in a star-like fashion, as shown in Figure 2.5, and heated to about 85°C. The tubes are designed to laminarize the wind flow, and the thermistors are placed at the narrowest point. When put in the wind, wind flows through the tubes and cools down the thermistors. This in turn changes their resistance, which is measured with the ADC of the microcontroller. This design is heavily inspired by the wind sensor built by Ishida et al. [26].

On the microcontroller, the 6 values s_1, \dots, s_6 are measured simultaneously, and passed to a probabilistic model to infer the wind speed and direction. While the wind speed is entirely modeled (no calibration data), the wind direction is determined by matching the measured values against calibration data. Figure 2.6 depicts this graphically.

Formally, the algorithm proceeds as follows. In a first step, the measured values are normalized with respect to their sum, i. e.,

$$q_i = \frac{s_i}{s_1 + \dots + s_6} \quad \text{for } i = 1, \dots, 6 \quad (2.1)$$

The sum $s_1 + \dots + s_6$ holds information about the wind speed (that we did not process further), while q_1, \dots, q_6 carry information about the wind direction. Since this normalization makes one of these values superfluous (by knowing any 5 values, the 6th value can be calculated), we only use the subset q_1, \dots, q_5 .

In a second step, the following log-likelihood is calculated for each angle α :

$$-L(\alpha) = \frac{\alpha - \alpha_{\text{prev}}}{\sigma_A^2} + \sum_{i=1, \dots, 5} \frac{q_i - \mu_{i, \alpha}}{\sigma_{i, \alpha}^2} \quad (2.2)$$

and the best angle as well as its log-likelihood value are selected:

$$\alpha_{\text{best}} = \arg \min L(\alpha) \quad (2.3)$$

$$L_{\text{best}} = L(\alpha_{\text{best}}) \quad (2.4)$$

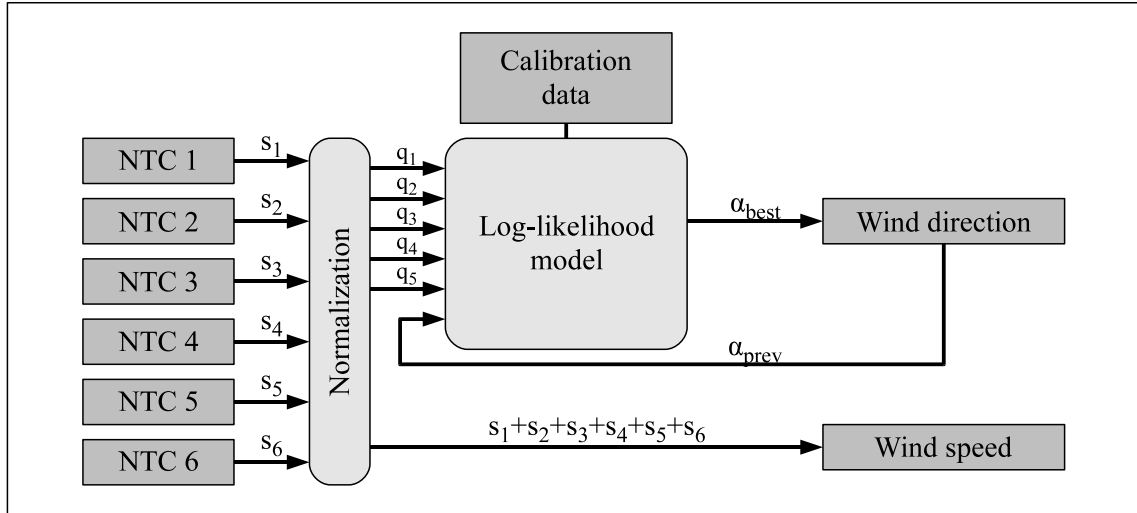


Figure 2.6: Data processing on the wind sensor board.

$\mu_{i,\alpha}$ and $\sigma_{i,\alpha}$ thereby denote the calibration data for the NTC values (mean and standard deviation). For a given angle α , a measurement c_i is assumed to be a sample of the normal distribution $\mathcal{N}(\mu_{i,\alpha}, \sigma_{i,\alpha})$.

The standard deviation σ_A stands for the wind direction variation we consider probable between two samples, and a good value for it was experimentally found to be 16° . (Note that σ_A was kept constant and did not vary with L_{best} of the previous step.)

A fixed-point integer version of this model was implemented on the microcontroller. To optimize the calculation time further, only angles in the range $\alpha_{\text{prev}} \pm 45^\circ$ are evaluated, and fine-grained evaluation is done in a successive approximation fashion to take advantage of the local smoothness of $L(\alpha)$. Evaluation is carried out down to a resolution of 0.7° and takes about 1 ms of execution time on the 40 MHz microcontroller.

From the main CPU, the wind direction as well as the raw sensor values can be accessed through the I²C bus in a fashion similar to the odor sensor board.

2.1.4.1 Calibration and Error

Our calibration data² consists of 64 tuples $(\mu_{1,\alpha}, \sigma_{1,\alpha}, \dots, \mu_{6,\alpha}, \sigma_{6,\alpha})$ for 64 angles equally spaced between 0° and 360° . For the log-likelihood model, mean and variance are linearly interpolated in between these calibration angles.

Calibration is carried out by letting the robot slowly turn around itself for about 10 times in laminar flow, and recording the normalized values (q_1, \dots, q_6) . Since the robot is turning at constant speed, each recorded sample $q_{i,n}$ can easily be attributed to the angle $\beta_{i,n}$ at which it was taken.

For each sensor i , the calibration values are then computed as follows:

$$\mu_{i,\alpha} = \sum_n q_{i,n} \cdot \mathbb{P}(W(\alpha) = \beta_{i,n}) \quad (2.5)$$

$$\sigma_{i,\alpha} = \sqrt{\sum_n (q_{i,n} - \mu_{i,\alpha})^2 \cdot \mathbb{P}(W(\alpha) = \beta_{i,n})} \quad (2.6)$$

²Note that we carry out the calibration for all 6 normalized NTC values, even if only 5 are used by the log-likelihood model.

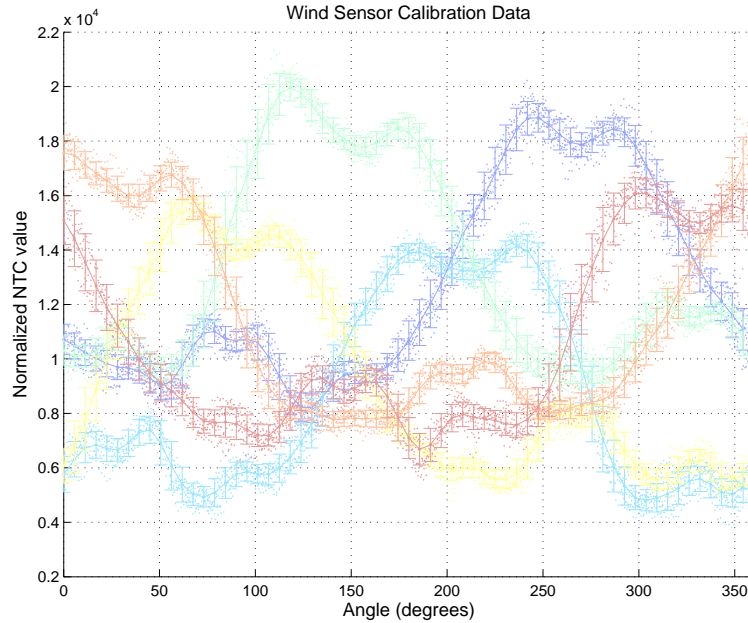


Figure 2.7: Calibration data of one of the wind sensor boards. Each band stands for one NTC sensor (values after normalization). The dots in the background depict the measurements obtained when letting the robot turn around itself. The solid lines show the mean, $\mu_{i,\alpha}$, and the bars indicate the standard deviation, $\sigma_{i,\alpha}$, computed from these measurements. $\mu_{i,\alpha}$ and $\sigma_{i,\alpha}$ were subsequently uploaded as calibration data onto the corresponding wind sensor board.

$W(\alpha)$ denotes the following Gaussian kernel:

$$W(\alpha) \sim \mathcal{N}\left(\alpha, \left(\frac{360^\circ}{64}\right)^2\right) \quad \text{defined over } [\alpha - 180^\circ, \alpha + 180^\circ] \quad (2.7)$$

and acts as anti-aliasing filter. High frequencies in the measurements are simply averaged out, as the 64 calibration tuples anyway could not capture them. Visual inspection (see Figure 2.7) shows that the chosen resolution is sufficient — the measurement data used for calibration do not show any higher frequencies.

Using this calibration procedure, a good wind sensor board achieves a standard error of less than 4° (that is, 95.4% of the values are within $\pm 8^\circ$, and 68.2% within $\pm 4^\circ$ of the true wind direction). Over the whole 360° , this error is normally distributed, whereas for individual angles, errors distributions are more complicated and tend to be skewed towards specific angles.

2.1.4.2 Version 1 of the Wind Sensor Board

Prior to the version with 6 thermistors, we built a version with 4 thermistors placed around the star-shaped obstacle depicted in Figure 2.8. Such obstacles create turbulence especially in downwind direction and therefore yield signals fluctuating much more than the turbulence in the wind field would suggest. This design is therefore less suitable to determine the wind direction. Nevertheless, we obtained accuracies in the order of $\pm 10^\circ$ in general, with the exception of a few critical angles where the error occasionally went up to 25° .

With this board, a simple floating point version of the same probabilistic algorithm (without the $P(\alpha = A(\alpha_{\text{prev}}))$ term) was running on the main CPU of the Khepera III robot. As this CPU does not have any floating point unit (FPU), this limited the update rate to about 2–4 Hz, even though the evaluation resolution was 10° .

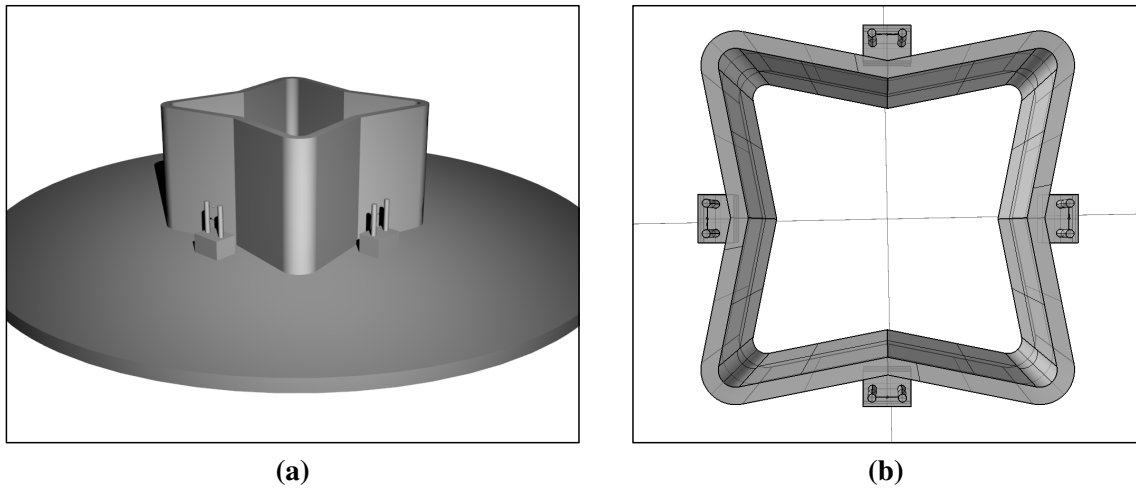


Figure 2.8: (a) Perspective view and (b) top view of a 3D rendering of the star-shaped obstacle on the first version of the wind sensor board. When wind is flowing around this obstacle, the 4 thermistors are cooled down differently and as a function of the wind direction.

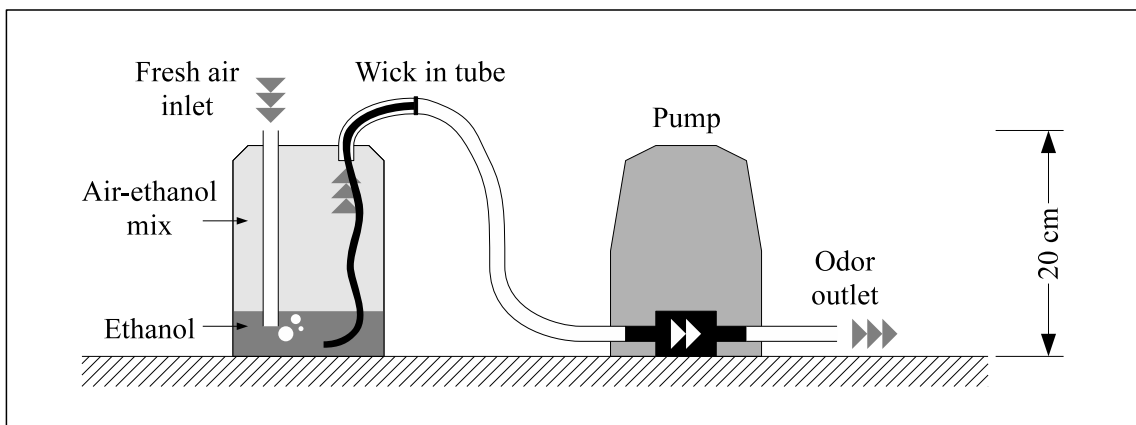


Figure 2.9: Sketch of the odor source.

Version 1 of the wind sensor board was used for all experiments with the bio-inspired algorithms (see Chapter 3), while version 2 was used for the experiments with the algorithms based on the probability theory (Chapter 4) and on formations (Chapter 5).

2.1.5 Odor Source

With a boiling point of 78°C , ethanol evaporates quite quickly at room temperature. We therefore based our odor source on spontaneous (unheated) evaporation of ethanol.

A sketch of the odor source is drawn in Figure 2.9. A bottle with a few deciliters of ethanol is connected with a tube to a pump. Part of the ethanol evaporates and mixes with the air on top, which is pumped towards the odor outlet. To increase the air-ethanol interface surface, a wick was inserted into the tube. In addition, incoming fresh air is led with another piece of tube to the bottom of the bottle.

Once the pump is started (at $1.2\text{l}/\text{min}$ in our experiments), air continuously flows through the whole system and keeps ethanol evaporation at an equilibrium point. This equilibrium point

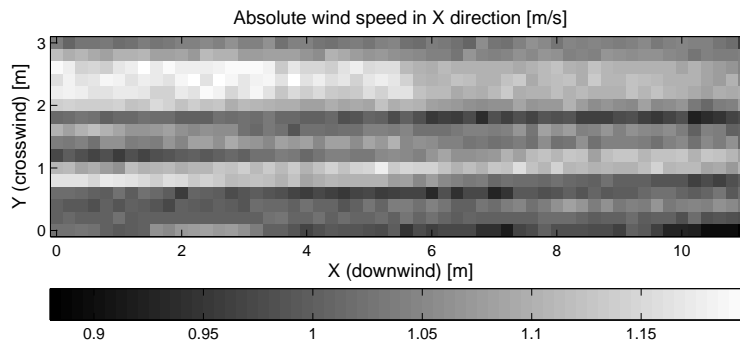


Figure 2.10: *Magnitude of the wind velocity (over 11 m). Each point is an average of 25 measurements with an Extech 407119 hot-wire anemometer mounted on the traversing system of the wind tunnel. The grid has a resolution of 20 cm in both X and Y direction. The wind speed was measured at the height of the robot’s wind sensor board.*

depends on the ambient temperature, which stays almost constant over several hours of time in the wind tunnel. (Temperatures in the wind tunnel are between 17°C in a winter night and 27°C on a hot summer day. Differences between day and night are in the order of 4°C .) Hence, the ethanol concentration at the odor outlet can be considered constant during the experiments.

The nominal quantities released were never measured or recorded, as none of the experiments showed any peculiarities that would point towards changes in the odor release rate.

An electronic board was developed allowing to control the pump speed with NMEA commands sent over a USB serial port connection.

2.1.6 Wind Tunnel

The wind tunnel has a usable area of 18×4 m, with a height of 1.9 m. Wind speeds of up to 5 m/s can be generated in that part of the facility, which is by far enough for odor source localization experiments. For convenience reasons, we used about 15 m of the total length and covered this part with a flat floor on which the robots can move without getting stuck.

The wind tunnel is furthermore equipped with a 3-axis traversing system, which allows to scan the whole tunnel with any sensor probe.

2.1.7 Wind and Odor Profile

We are working with wind speeds of roughly 1 m/s in the wind tunnel. As shown in Figure 2.10, the wind speed is almost homogeneous with differences of 0.2 m/s between the maximum and minimum speeds.

Only the magnitude of the wind velocity was measured, but not its direction. The odor profile, measured with the odor board itself and shown in Figure 2.11, reveals however that the wind is slightly drifting towards one side of the wind tunnel, at a rate of about 5 cm/m. This is presumably due to the fan, but we believe that this drift does not significantly affect the results.

2.1.8 Camera System

In order to allow for absolute localization of the robots in the wind tunnel, we use 6 overhead cameras as sketched in Figure 2.12. When setting up the system, one of the major challenges in the wind tunnel was the low ceiling: the total height is only about 1.9 m, yielding a distance

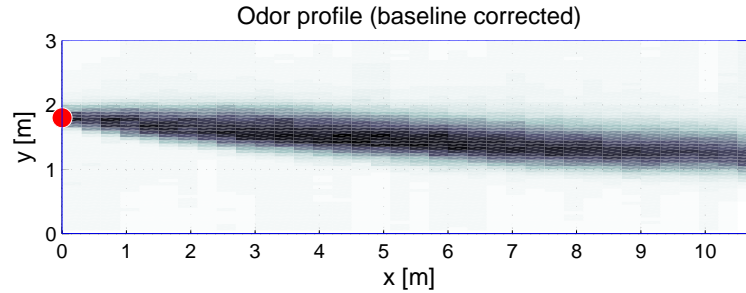


Figure 2.11: *Odor profile (over 11 m). Each measurement point is an average over about 20 seconds. The grid has a resolution of 30 cm in X direction, and 5 cm in Y direction. The odor was measured at the height of the robot’s odor sensor board using the traversing system of the wind tunnel.*

between the camera lens and the marker on the robot of about 1.6 m. To cover an area of 18 x 4 m, wide angle lenses (with high distortion) are necessary. We are using lenses with a focal distance of 1.8 mm.

Another challenge was posed by the metallic (and therefore reflective) wall and ceiling surfaces, which make it very hard to achieve the homogeneous and diffuse light conditions ideal for tracking. Our setup therefore works with two LEDs mounted on the robot, and a dark environment. The two LEDs are red and green, and mounted on the wind sensor board (which anyway has to be on top of the robot). Despite the Bayer pattern of the camera, red and green can be distinguished with high accuracy using simple thresholding. As the background is all black, this allows for robust tracking of multiple robots almost anywhere in the wind tunnel.

The current system allows us to achieve frame rates of 10–20 Hz, and delays of 100 ms (from image acquisition until the robots receive the position information). The spatial resolution of the system is about 1–2 cm, while the accuracy varies between 2 and 8 cm depending on whether a robot appears in the image center or close to the border. Given that robots are moving at 10–30 cm/s (i. e., 1–3 cm between camera frames), this is a good trade-off between spatial and temporal resolution.

2.1.8.1 Camera Trigger

To synchronize the 6 cameras, we built the trigger box displayed in Figure 2.13. This box provides power and a trigger signal to up to 8 cameras. Cameras are configured to start image acquisition on this external trigger. The trigger is generated by a microcontroller, and its interval can be configured via USB.

The following procedure allows us to get the cameras into synchronized state:

1. Disable the trigger signal. This is done by sending an NMEA sentence over USB to the trigger box.
2. Reset and start all cameras, and wait until they are ready. (SwisTrack takes care of that when a production run is launched.)
3. Set the trigger interval and enable the trigger signal, again by sending an NMEA sentence to the trigger box.

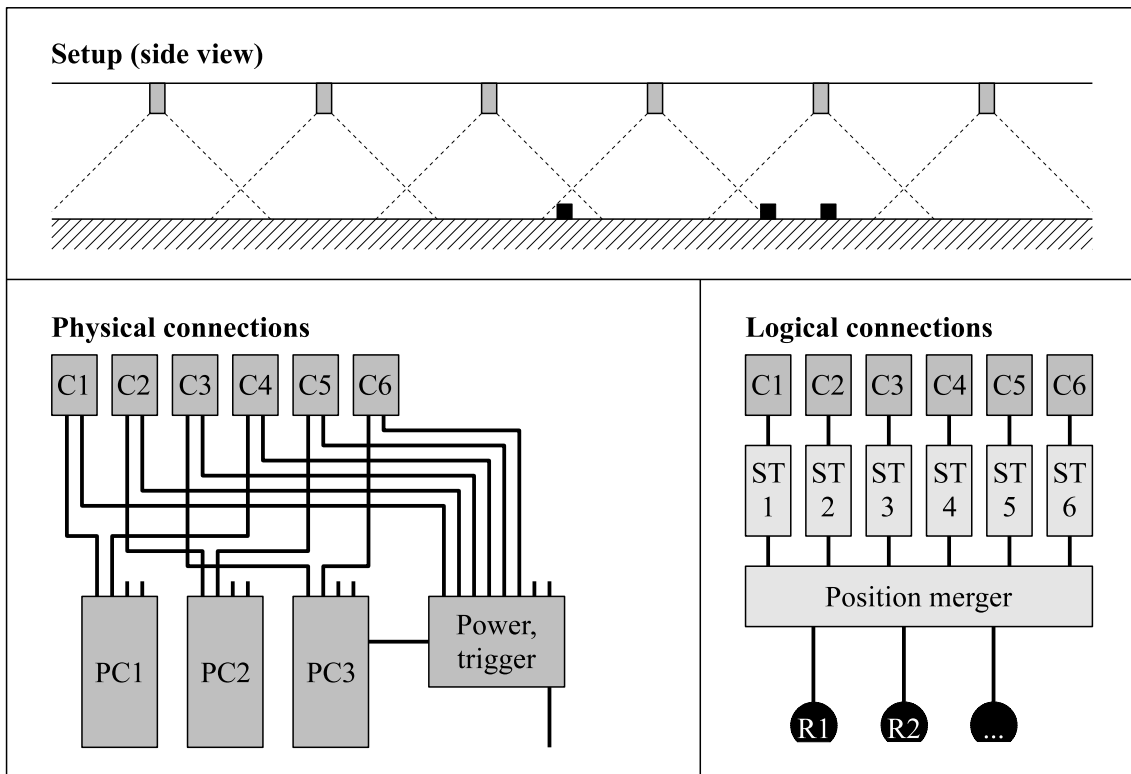


Figure 2.12: Sketch of the camera system setup, and its physical and logical connections. $C1, \dots, C6$ are the cameras, while $ST1, \dots, ST6$ are instances of SwisTrack [73], the software processing the camera images and determining the robot positions. The position merger is a separate program running on PC3 which collects the positions reported by all cameras, merges them, and sends them to the robot (for positioning, closed-loop), or logs them (for trajectory analysis, open-loop).

Experience with the system has shown that this is extremely reliable, and due to the very low probability that trigger signals are being lost in the cable, the camera system remains synchronized for hours.

2.1.8.2 SwisTrack

Camera images are read and processed by SwisTrack 4 [73]. SwisTrack is an open source tracking software for multi-agent systems, and was at version 3 [76] at the beginning of this PhD thesis. We greatly enhanced the user interface, introduced a component-based architecture and added support for GigE cameras. With the help of other developers who added new features as well, version 4 of SwisTrack was released in 2007, and gradually improved and enhanced later on. Notably thanks to the new component-based architecture, the project attracted a series of researchers from other



Figure 2.13: Picture of the camera trigger box (front and back).

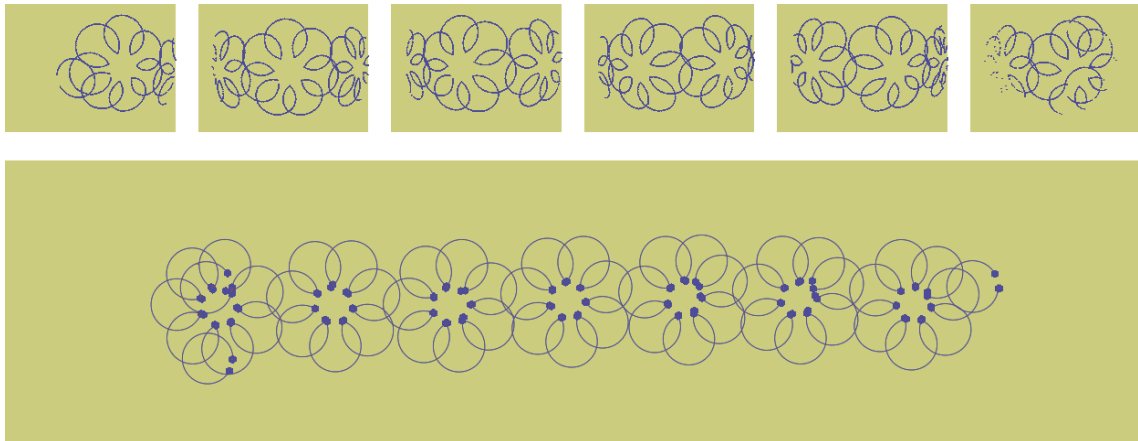


Figure 2.14: Screen shot of the optimization program. The big area in the lower part shows the wind tunnel arena with the robot trajectories recorded using wheel odometry and optimized on their intersection points. The small areas in the upper part of the screen show the same trajectories as recorded by each of the 6 cameras. Circles close to image borders are highly distorted, which is what we want to correct for.

universities, who partly contributed their own components, and partly just used the software.

Each of the 6 cameras runs a separate instance of SwisTrack. For performance reasons, two cameras each are served by one computer. As images are acquired, they are thresholded to find the red and the green LEDs using a standard blob detection algorithm. Pairs of red and green blobs sitting close enough to each other are then combined to particles (robots), and the obtained particle positions are projected into a world coordinate system using a 2D version of the Tsai transform [77].

A separate program collects the particles reported by each SwisTrack instance and merges them. Merging is based on a distance criterion, i. e., particles that are very close to each other ($< 13 \text{ cm} \approx$ diameter of the Khepera III robot) are merged into one robot. The program then stores the robot positions into a file (open loop, e. g., for offline analysis later on) and sends it via UDP to the robots if desired (closed loop, e. g., for absolute localization).

2.1.8.3 Calibration

While image acquisition and blob detection parameters can easily be tuned by hand, the Tsai transform requires a more sophisticated calibration procedure.

The Tsai transform is a non-linear model of the projection of a scene onto a camera chip surface. It takes into account pixel size, spherical lens distortion, misalignments between the camera chip and the lens, as well as 3D rotation and translation. For square pixels, this amounts to 11 parameters. Note that most of these parameters have a direct physical meanings (e. g., focal length, rotation, ...) and can therefore be guessed quite accurately. Nevertheless, tiny changes in the parameters can yield substantial errors, and systematic parameter optimization is therefore indispensable.

To find these 11 parameters, we first start by recording image-world point pairs. For that, we calibrate the odometry of a Khepera III robot and let it move to create approximately the rose pattern shown in Figure 2.14. This pattern consists of 49 circles, for which we record the real-world movement with wheel odometry on the robot, and the image position with the cameras.

Since odometry allows us to obtain the relative movement only but not the absolute real-world

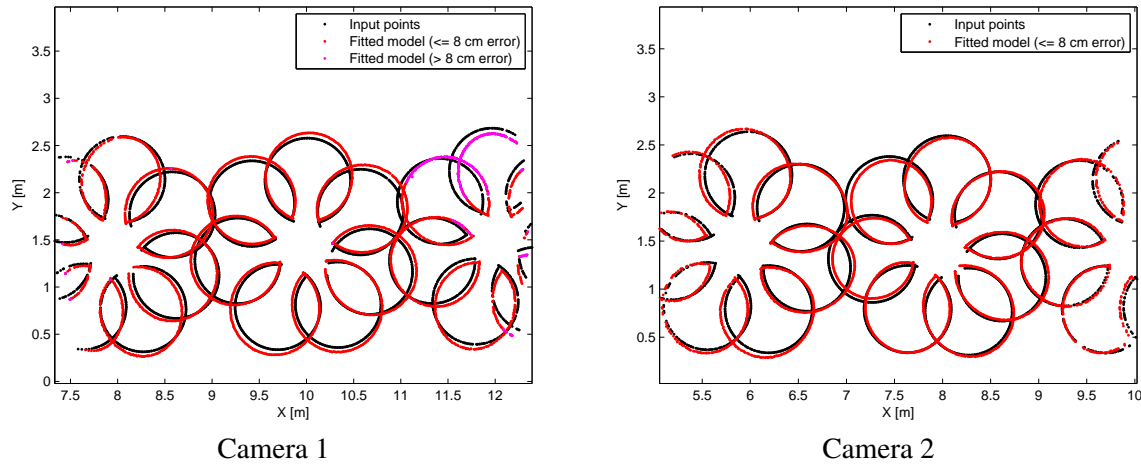


Figure 2.15: Sample plots of the Tsai transform with fitted parameters. The parameters of the transform were first optimized using linear search, and in a second step improved using a PSO algorithm.

position of the robot, we do not a priori know where the circles are. Using the information from the camera system, we can however find intersection points between circles. Since intersections must occur at the same physical location, we can apply an optimization algorithm to infer the relative position among circles by minimizing the distance between intersection points. In order to obtain absolute coordinates, we acquire three reference points (with reference circles) at known locations and add them to the optimizer.

On clean data sets with well calibrated wheel odometry on the robot, optimization takes a few seconds and achieves global accuracies in the order of 5 cm, and local accuracies (distances between nearby points) of less than 2 cm. This is absolutely sufficient for our purposes.

Each image position of the robot recorded by any of the 6 cameras can now be assigned to a real-world position in a global reference frame. This global reference frame is furthermore the same for all cameras. These image-world pairs are then used to fit the parameters of the Tsai transform. We implemented two version for this:

- ▷ A *linear search optimization algorithm* starts from an initial guess of the parameter set, and optimizes the parameters one-by-one in an order that was found to yield good results. Within seconds, solutions with a maximum error of below 10 cm (at the border of the camera image) are obtained.
- ▷ These results can be further improved using a *PSO algorithm*, which usually comes up with solutions around 8 cm after a few minutes of optimization. As an example, plots for two cameras are provided in Figure 2.15.

This calibration procedure has two important advantages. First, all cameras are calibrated at the same time based on the same raw data and the same global coordinate frame. This not only saves time, but calibrates the systems as a whole, instead of calibrating each camera individually. Therefore, positions in overlapping areas (i. e., areas seen by more than one camera) will automatically be consistent. Second, the raw data for calibration are recorded using exactly the same markers (red/green LEDs) at exactly the same height above the floor as during the experiments later on. This excludes errors that could occur with misaligned markers or different marker detection techniques.

2.2 Simulation Setup

In Webots [72], a commercial realistic robotic simulator, we implemented the same experimental setup (see Figure 2.1) in simulation. Webots ships with a calibrated model of the Khepera III robot which allows to use almost the same software implementation of the algorithms as on the real robots.

As Webots does not include any support for odor or smoke, the simulation environment (Figure 2.17) was augmented with a wind and odor propagation model, and the robot model was extended with the corresponding sensors to measure the odor concentration and a wind direction (Figure 2.16).

2.2.1 Experimental Arena

The experimental arena is a rectangular area of 16 m length and 4 m width, which corresponds roughly to the dimensions of the wind tunnel. At 1 m from one end of the arena, a circular odor source of radius 12 cm is placed. The robot is placed at roughly 14.5 m downwind from that spot.

2.2.2 Advection Model

A constant wind field of 1 m/s is used. This corresponds to a constant laminar flow which is comparable to the one in the wind tunnel. In the coordinate system indicated in Figure 2.17, the wind vector at position \mathbf{u} , $\mathbf{a}(\mathbf{u})$, can be written as

$$\mathbf{a}(\mathbf{u}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad (2.8)$$

2.2.3 Odor Propagation Model

The odor propagation model closely resembles the filament-based model proposed by Farrell et al. [7]. This model is easy to implement and requires only a very limited amount of CPU power. Yet, it generates an intermittent plume which is similar to the real plume in the wind tunnel.

Odor is thereby simulated as a set of filaments ($i = 0, \dots, N$), each containing a constant amount $s = 8.3 \cdot 10^9$ of molecules or particles. Each filament is defined by its position, $\mathbf{p}_{i,t}$, and its width, $w_{i,t}$.

In each time step, the position of a filament is updated according to the wind flow and a stochastic process:

$$\mathbf{p}_{i,t+\Delta t} = \mathbf{p}_{i,t} + \mathbf{a}(\mathbf{p}_{i,t})\Delta t + \mathbf{v}_p \quad (2.9)$$

The stochastic component \mathbf{v}_p is a vector of three independent Gaussian random variables, $\mathcal{N}(0, \sigma_p^2)$, with standard deviation $\sigma_p = 0.1$ m. The simulation time step, Δt , was set to 32 ms.

To model molecular dispersion, filaments become wider with time while their peak concentration decreases. The width of a filament evolves as

$$w_{i,t+\Delta t} = w_{i,t} + \frac{\gamma}{2w_{i,t}} \quad \text{with } \gamma = 4 \cdot 10^{-7} m^2/s \quad (2.10)$$

The odor source releases 100 such filaments per second with an initial width of $w_{i,0} = 10$ cm and an initial position which is uniformly distributed over the circular area of the source. This yields a plume comparable to the real plume in the wind tunnel. Note that no rigorous calibration of the filament-based model vs. reality has been carried out.

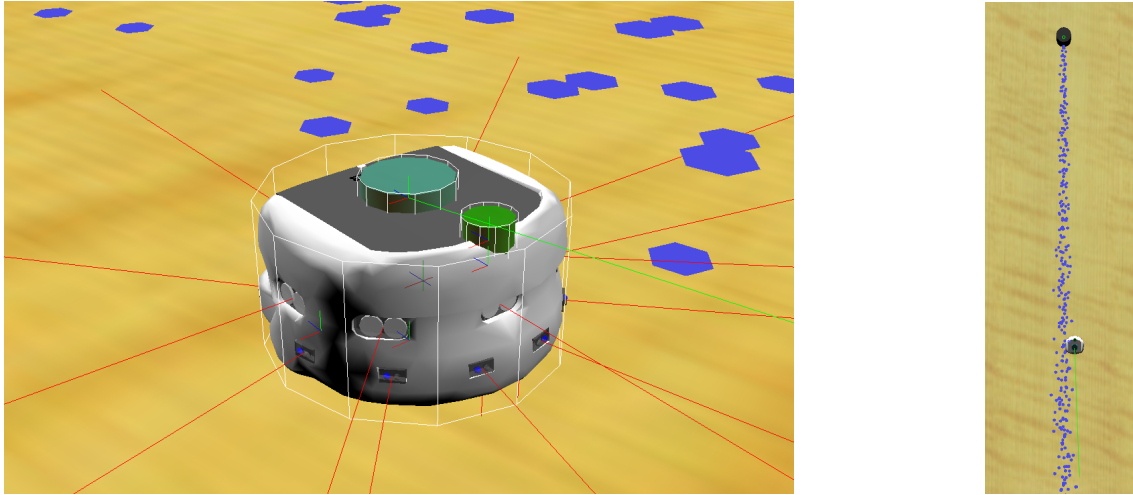


Figure 2.16: Simulated Khepera III robot equipped with an odor sensor (small cylinder on top of the robot) and a wind sensor (big cylinder). The hexagons in the air represent odor filaments.

2.2.4 Odor Sensor Model

The odor concentration at time t and position \mathbf{u} is calculated as the sum over the concentration contribution of all filaments,

$$C_t(\mathbf{u}) = \sum_{i=0}^N c_{i,t}(\mathbf{u}) \quad (2.11)$$

and each filament i contributes

$$c_{i,t}(\mathbf{u}) = \frac{s}{w_{i,t}^3} \exp\left(-\frac{|\mathbf{u} - \mathbf{p}_{i,t}|}{w_{i,t}^2}\right) \quad (2.12)$$

to the concentration. Hence, the concentration decays exponentially with increasing distance from the center of a filament.

The virtual odor sensor reports this concentration $C_t(\mathbf{u})$ without adding any additional noise, as the noise is negligible even on the real platform.

2.2.5 Wind Direction Sensor Model

The wind sensor reports a noisy wind measurement,

$$\mathbf{a}_s(\mathbf{u}) = \mathbf{a}(\mathbf{u}) + \mathbf{v}_a \quad (2.13)$$

where \mathbf{v}_a is a vector with samples of a zero-mean normal distribution ($\mathcal{N}(0, \sigma_a^2)$). Since the wind field is constant in all our simulations, the reported value in world coordinates is simply

$$\mathbf{a}_s(\mathbf{u}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} \mathcal{N}(0, \sigma_a^2) \\ \mathcal{N}(0, \sigma_a^2) \\ \mathcal{N}(0, \sigma_a^2) \end{pmatrix} \quad (2.14)$$

This vector is rotated into the local reference system of the robot to account for the robot's pose.

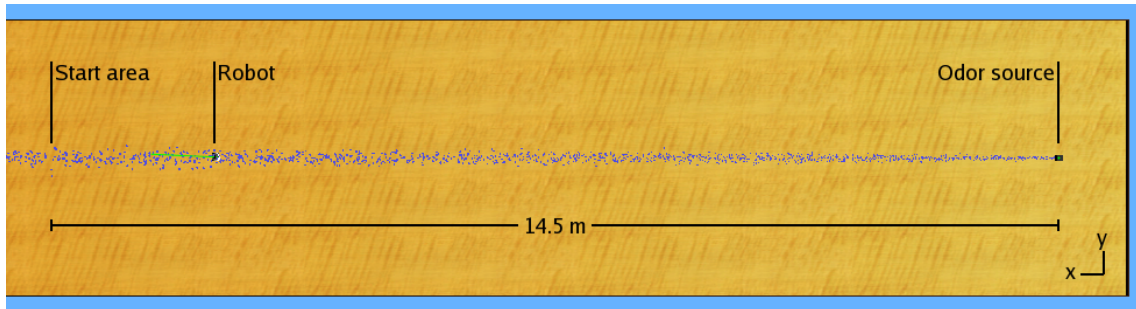


Figure 2.17: Simulated environment in Webots.

2.3 Summary and Conclusion

With the tools introduced in this chapter, we have a robust and flexible setup available for running odor source localization experiments with real robots and in simulation. It should however be noted that the development effort was a substantial part of the present PhD work. It took us almost 2 years until the first experiments (still single-robot) could be carried out in the wind tunnel, and the setup was gradually improved and enhanced afterwards. The current state of the setup allows to quickly and painlessly carry out dozens of experimental runs in the wind tunnel, and hundreds of runs in simulation.

One may argue that we went a step too far in terms of robustness of the setup, and invested too much time in fixing issues that would only come up every now and then. This is certainly true for SwisTrack [73], for which we invested several man-months to have a nice software that allows us to dynamically change parameters in a graphical front-end. Even if tuning these parameters in the code is a tedious and time-consuming operation, we would probably have spent several days or weeks doing that, but not months.

Yet, this robustness of the setup made it possible for students to help with this project and be efficient, even if their projects often lasted for a few weeks only. Thanks to the Khepera III Toolbox [75], for instance, students didn't need much guidance when they started working with the Khepera III robot. They not only had a good documentation on how to use the robot available, but also a substantial amount of source code to start with. This is very motivating for students, and saves a lot of time in direct supervision.

In addition, thanks to publishing them as open source projects, both SwisTrack and the Khepera III Toolbox were picked up by many other researchers all around the world, who used them and contributed to these projects in a variety of ways. From a global perspective, this open source approach is therefore very beneficial to the research community as a whole.

3 Bio-Inspired Algorithms

Biology offers plenty of examples for odor source localization. It is therefore not surprising that a number of bio-inspired algorithms have been proposed in the scientific literature. In 2005 (at the beginning of this thesis), such algorithms were ubiquitous in the odor source localization literature, with the casting behavior of moths being the most prominent one being imitated. Gradient-based approaches were known to not perform well in the intermittent plume structure, and biology offered attractive alternatives to be studied. There are indeed good reasons for considering such algorithms:

- ▷ A number of animals — from small insects to dogs and taller species — are very good at localizing odor sources [1]. Many species rely on this ability for mating and food scavenging, two crucial tasks directly affecting survival in evolutionary terms.
- ▷ Simple behavioral models for moths are known [51]. Implementing such behaviors on a real robot is straightforward, and therefore attractive for research and applications.

Bio-inspired algorithms were therefore the first algorithms we tested in our setup. More specifically, we implemented three such algorithms (*casting*, *surge-spiral*, and *surge-cast*), and evaluated them through simulation and real-robot experiments with different parameters, in different wind flows, and with multi-robot systems.

In the first part of this chapter, we first introduce the algorithms (Section 3.1), and report on experiments with the real robots (Section 3.2.1) and in simulation (Section 3.2.2) in laminar flow. We then compare the results of all these experiments with the theoretically calculated performance in Section 3.3. In Section 3.4, we add obstacles to the setup to study the impact of turbulence on the algorithms.

The second part of the chapter is devoted to multi-robot version of the same algorithms. Using multiple robots instead of just one is an elemental extension which is intuitively thought of improving the performance, especially if the robots collaborate. In Section 3.5, we report on simulation experiments with up to 5 robots that are not collaborating, and discuss collaboration in Section 3.6.

3.1 Algorithms

The three algorithms discussed in this chapter are a combination of upwind surge, casting, and spiraling — three behaviors that have been observed with insects [51]. The algorithms use only binary odor information, that is, they either *perceive the odor* or *do not perceive any odor*, but ignore different concentration levels. Commonly, the measured concentration is thresholded to obtain this binary value, and more elaborate processing was used only for the experiments with obstacles.

Furthermore, all three algorithms need a wind sensor to measure the wind direction. As molecules are mainly transported by advection, this piece of information is very valuable, and as important as the odor sensor in environments with a main wind flow. The wind speed is ignored.

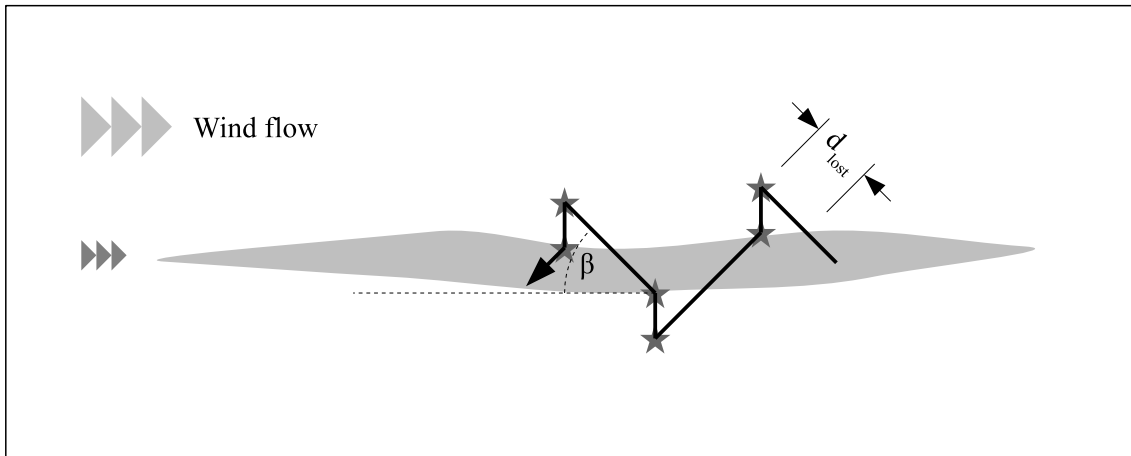


Figure 3.1: Sketch of the casting algorithm. The stars indicate where the wind direction is measured.

Since we are only interested in the plume tracking behavior, the robot starts in the plume, and declares failure if it gets too far away from it¹. This allows us to rule out arena geometry effects, which could greatly influence the results (e. g., high variance introduced by randomized search techniques).

Similarly, source declaration is done by a supervisor (ideal source declaration) and therefore does not affect the results.² Experiments are considered successful if the robot has come in physical vicinity of the source.

3.1.1 The Casting Algorithm

The casting algorithm is very similar the one described by Li et al. [53]. As shown in Figure 3.1, a robot in the plume moves upwind with an angle β until it is out of the plume for a certain distance, denoted d_{lost} . Once the plume is lost, the robot turns and moves crosswind until it hits an odor packet, and then moves upwind with angle β again.

The wind direction is measured each time the robot switches to plume reacquisition, and when it encounters the plume again.

3.1.2 Surge-Spiral

The surge-spiral algorithm is similar to Hayes' algorithm presented in [9], except that here we focus exclusively on its use for plume traversing. Hence, we have a single spiral gap parameter.

A robot in the plume moves straight upwind until it loses the plume for a distance d_{lost} . It then tries to reacquire the plume by moving along an Archimedean spiral with gap size d_{gap} . Unlike [9], we start our spiral in upwind direction (as drawn in Figure 3.2) to avoid the 90° turn required in [9].

The wind direction is measured when the robot switches from upwind surge to spiraling, and when it switches back to upwind surge.

¹On the real robots, this is done using IR sensors detecting the arena boundaries. In Webots, a supervisor controller takes care of this.

²On the real robots, this is done using IR sensors detecting a specific colored patch on the floor. In Webots, a supervisor controller takes care of this.

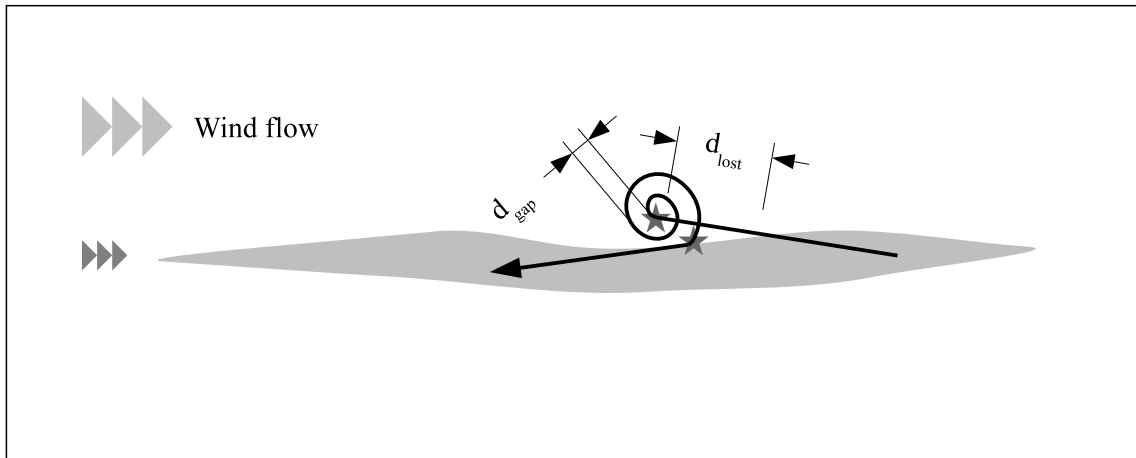


Figure 3.2: Sketch of the surge-spiral algorithm. The star indicates where the wind direction is measured.

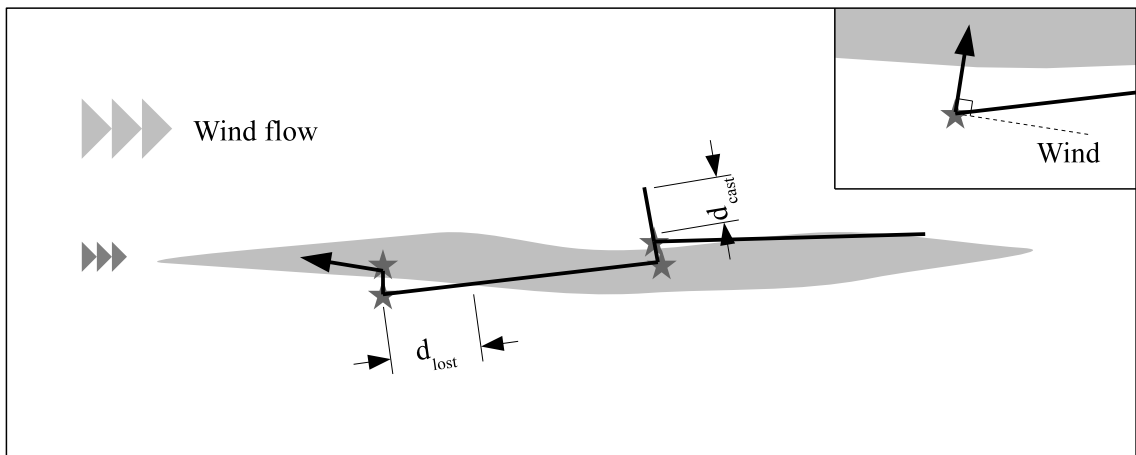


Figure 3.3: Sketch of the surge-cast algorithm. The stars indicate where the wind direction is measured.

3.1.3 The Surge-Cast Algorithm

The *surge-cast* algorithm [78] is a combination of upwind surge and cross-wind casting. It is similar to the surge-spiral algorithm, with the spiral being replaced by cross-wind movement.

A robot in the plume moves straight upwind until it loses the plume for a distance d_{lost} . It then tries to reacquire the plume by moving cross-wind for a set distance (d_{cast}), first on one side and then on the other. To maximize the chances of hitting the plume in the first cross-wind movement, the robot measures the wind direction to estimate from which side it left the plume.

If the robot did not reacquire the plume by casting, the run is considered unsuccessful. In a real application, the robot would probably switch back to plume finding behavior, or try to reacquire the plume with a larger cast distance or with spiraling.

The wind direction is measured when the robot switches from upwind surge to casting and when it switches back to upwind surge, as indicated in Figure 3.3.

Table 3.1: Mean values (except for the success ratio) of all configurations tested with the real robots in the wind tunnel. 20 runs per configuration were carried out. The distance overhead is the traveled distance divided by the upwind distance ($\frac{d_t}{d_u}$).

Configuration	A	B	C	D	E	F	G
Success ratio	90 %	100 %	85 %	100 %	86.4 %	90 %	40 %
Distance overhead [m/m]	1.1638	1.4323	1.6256	1.1429	1.1211	1.102	1.0585
Traveled distance [m]	17.08	21.08	23.90	16.65	16.36	16.08	15.33
Time to target [s]	179.9	231.2	263.3	161.2	165.4	162.1	152.0
Ratio in plume	78.8 %	63.3 %	58.1 %	82.0 %	83.4 %	84.7 %	86.9 %
Upwind speed [m/s]	0.083	0.064	0.056	0.091	0.089	0.090	0.096
Mean robot speed [m/s]	0.096	0.091	0.091	0.103	0.099	0.099	0.101

3.2 Results in Laminar Flow

3.2.1 Real Robots

We first carried out experiments with the real robots in the wind tunnel. More precisely, we carried out 20 runs for each of the following configurations:

Algorithm	Specific parameter
A Casting	$\beta = 10^\circ$
B Casting	$\beta = 20^\circ$
C Casting	$\beta = 30^\circ$
D Surge-spiral	$d_{\text{gap}} = 0.58 \text{ m}$
E Surge-cast	$d_{\text{cast}} = 0.72 \text{ m}$
F Surge-cast	$d_{\text{cast}} = 0.43 \text{ m}$
G Surge-cast	$d_{\text{cast}} = 0.14 \text{ m}$

The forward speed of the robot (on straight lines) was approximately 10.6 cm/s and the plume lost distance was set to $d_{\text{lost}} = 40 \text{ cm}$ for all experiments. The plume threshold was determined before each run by measuring the response of the sensor in fresh air in the wind tunnel.

In each run, the robot was released in the odor at a position about 14.5 m downwind from the target area, and the corresponding algorithm was launched. If the robot reached the target area around the odor outlet (determined with the floor sensors), the run was considered successful. Whenever a robot gave up or touched an arena wall (determined with the infrared proximity sensors), the run was stopped and considered unsuccessful. During the run, the trajectory (using odometry) and the odor concentration were recorded. Distance and upwind distance were derived from the trajectory, and the duration of each run was measured on a host computer.

Table 3.1 shows the mean values of the data recorded during the experiments. Besides the success ratio, the most interesting of these values is the ratio between the traveled distance (d_t) and the upwind distance (d_u), which is plotted in Figure 3.4. This value indicates what distance the robot had to drive in order to come 1 m closer to the source, and is therefore bigger or equal to 1. Furthermore, a selection of runs of all three algorithms is plotted in Figure 3.5.

The surprisingly good result of configuration A should be taken with a grain of salt, since the wheel diameter difference produced some bending of the trajectory (see Figure 3.5) which

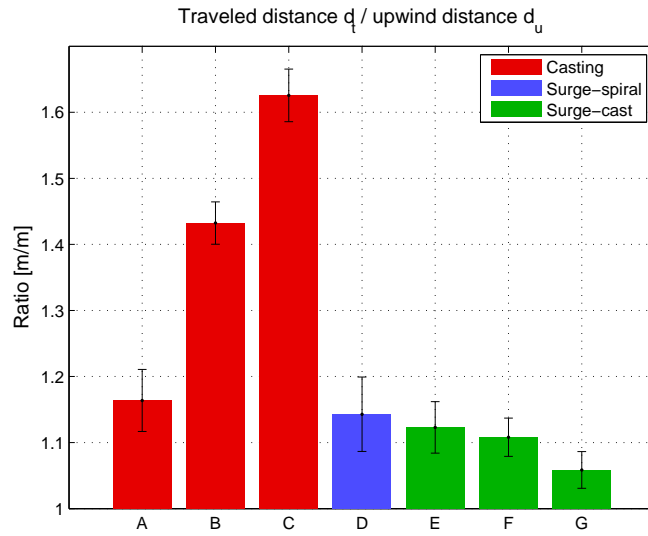


Figure 3.4: Distance overhead (mean with 95 % confidence interval for normal data) obtained through real-robot experiments. Only successful runs were included in the analysis, and the success rates are indicated in Table 3.1. Lower values are better.

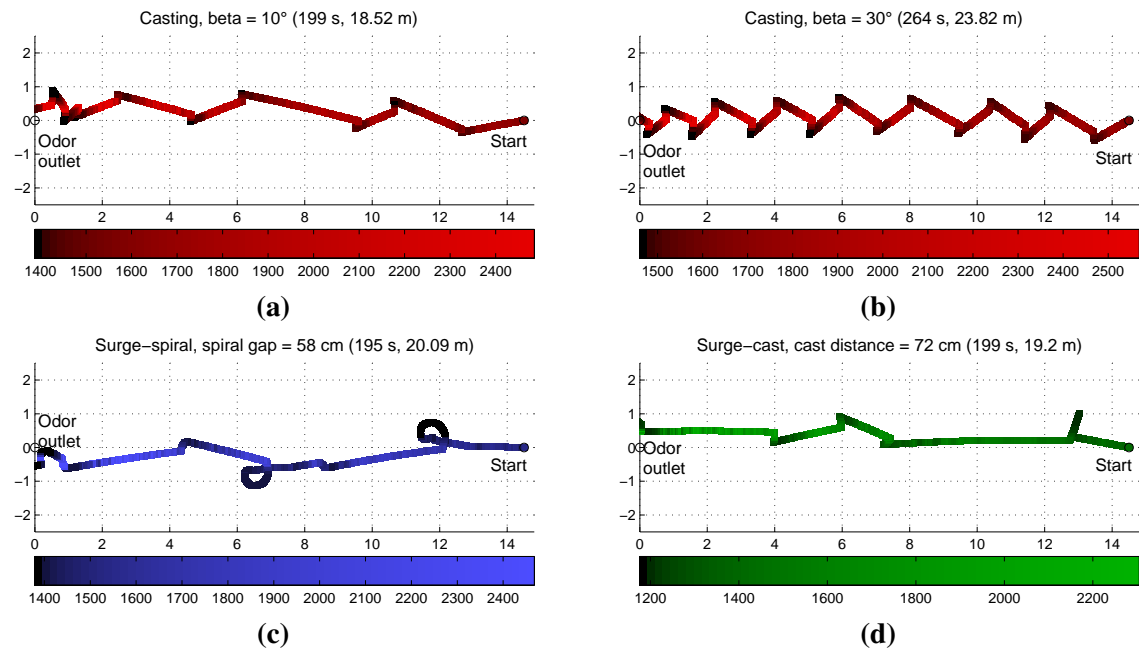


Figure 3.5: Sample real-robot trajectories with odor concentration shading produced by the real robot. The bars below the plots indicate the translation from shading to odor sensor response (in arbitrary units). Note that straight trajectories are bent because of a tiny difference (0.08 mm) in wheel diameter between the left and the right wheel. The plume threshold was set to 100 units above the baseline concentration value indicated on the left side of the colored bar. (a, b) Successful runs of the casting algorithm. (c) Successful, but unlucky run of the surge-spiral algorithm. (d) Successful, but unlucky run of the surge-cast algorithm.

worked in favor of the algorithm. Without this effect, one would expect the success rate of this configuration to be very low [79].

At first glance, the surge-cast algorithm outperforms casting and is at least as good as the surge-spiral algorithm. The overlapping confidence intervals do not allow us to make a statistical judgment about the configurations D, E and F. However, since the surge-cast algorithm moves backward and forward only instead of making complete circles, its performance is expected to be slightly better.

The current implementation of surge-cast is less robust than surge-spiral. This is mainly the case for configuration G, in which the cross-wind distance is clearly too small. However, one should bear in mind that the algorithm gives up after unsuccessful cross-wind movement, instead of switching to spiraling (as moths do) or increasing the cross-wind distance.

3.2.2 Robotic Simulation

After the real-robot experiments, we passed on to experiments with the Webots simulation environment presented in Section 2.2, and analyzed three parameters for each of the three algorithms, totaling to 9 sets of experiments:

Algorithm	Specific parameter	σ_a	d_{lost}
A Casting	variable	10 cm	61.4 cm
B Casting	$\beta = 25^\circ$	variable	61.4 cm
C Casting	$\beta = 25^\circ$	10 cm	variable
D Surge-spiral	variable	10 cm	61.4 cm
E Surge-spiral	$d_{\text{gap}} = 22.2$ cm	variable	61.4 cm
F Surge-spiral	$d_{\text{gap}} = 22.2$ cm	10 cm	variable
G Surge-cast	variable	10 cm	61.4 cm
H Surge-cast	$d_{\text{cast}} = 27$ cm	variable	61.4 cm
I Surge-cast	$d_{\text{cast}} = 27$ cm	10 cm	variable

d_{lost} stands for the plume lost distance, while σ_a denotes the noise of the wind direction sensor, as defined in Section 2.2.5. Note that this noise is expressed in meters, as it is the standard deviation of the error term added to the wind vector. Each set of experiments consists of 9 choices for the variable parameter with 50 independent runs each. In each run, the robot was released in the odor at a position about 14.5 m downwind from the target area, and the corresponding algorithm was launched. If the robot reached the odor source, the run was considered successful. If the robot touched an arena wall, the run was aborted and declared unsuccessful. Distance and upwind distance were derived from the trajectory, recorded during the run.

The forward speed of the robot (on straight lines) was 10.6 cm/s and therefore the same as with the real-robot experiments in the wind tunnel. The plume threshold was set to $c = 100$.

Sample runs for three chosen parameter configurations are shown in Figure 3.6. In the following paragraphs, we discuss the results for each of the three algorithms. A comparison with the real-robot results will be provided later in Section 3.3.

3.2.2.1 Casting

The results for the *casting* algorithm are displayed in Figure 3.7. The upwind angle has a major influence on the performance. Small angles yield a low distance overhead, but also a low success rate. In our setup, only configurations with $\beta > 20^\circ$ resulted in acceptable success rates.

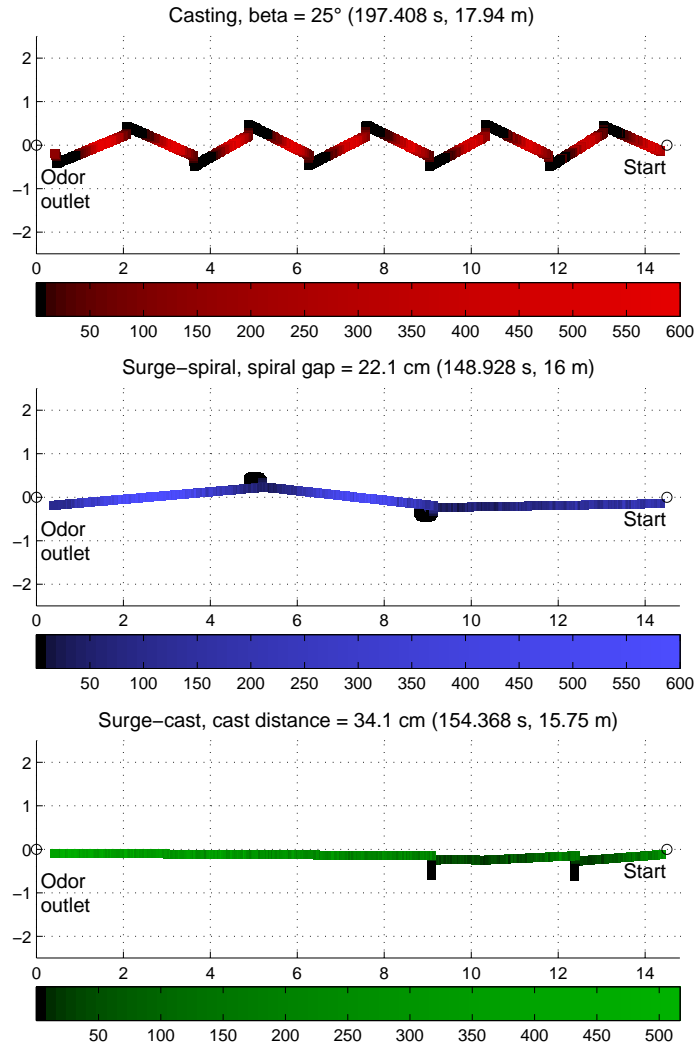


Figure 3.6: Successful simulation runs of all three algorithms (configurations A, D and G). The bars below the plots indicate the concentration shading.

A similar behavior can be observed for the plume lost distance: on one end of the scale, the success rate drops significantly, whereas on the other end, the performance gets worse. Hence, choosing parameters for this algorithm is ultimately a trade-off between performance and robustness.

The accuracy of the wind sensor only has a marginal impact on the performance, and no visible influence on the robustness as long as the accuracy is below a certain threshold. If the noise is too high, however, the algorithm does not work at all.

3.2.2.2 Surge-Spiral

The *surge-spiral* algorithm is extremely robust and virtually all 1350 runs succeeded. With the simple plume used in this setup, a spiral of increasing radius will always reacquire the plume before hitting a wall. In addition, the performance is fairly good over a wide parameter range. As expected, a small spiral gap is advantageous, at least as long as the robot reacquires the plume within one turn. Figure 3.8 (E) also suggests that higher d_{lost} yield slightly better performance.

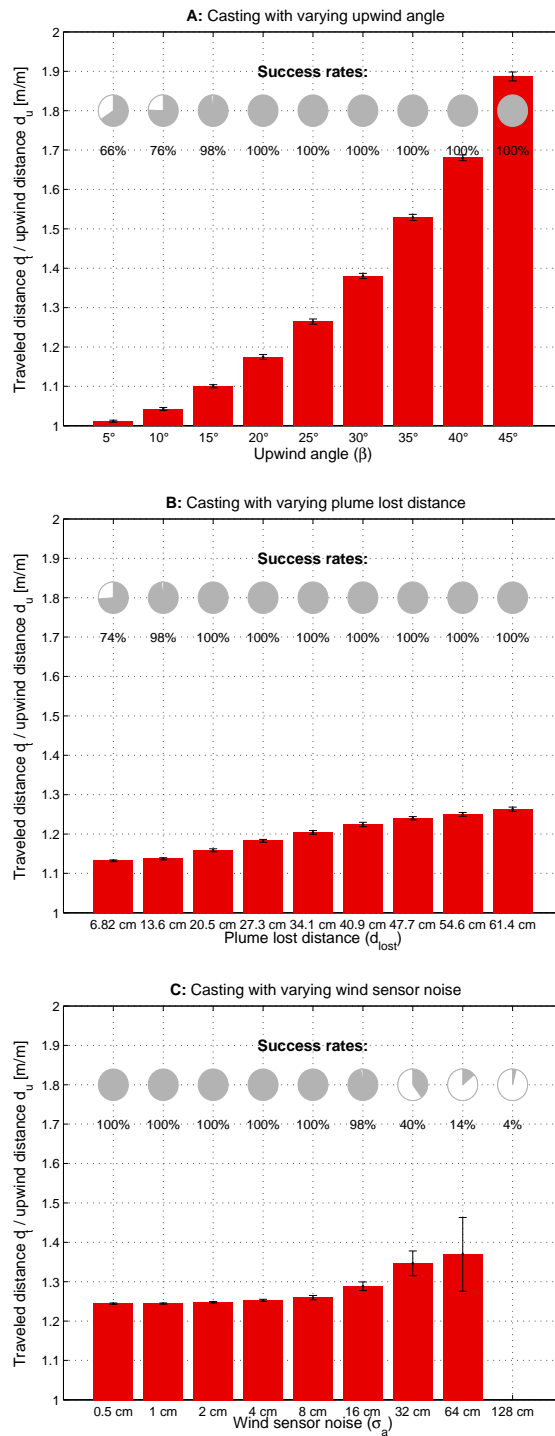


Figure 3.7: Simulation results obtained with the casting algorithm. The error bars indicate the 95 % confidence interval for the mean (assuming normally distributed data). **A:** With varying upwind angle (β). **B:** With varying plume lost distance (d_{lost}). **C:** With varying noise on the wind sensor reading (σ_a). The last bar is omitted because of the small number of successful runs.

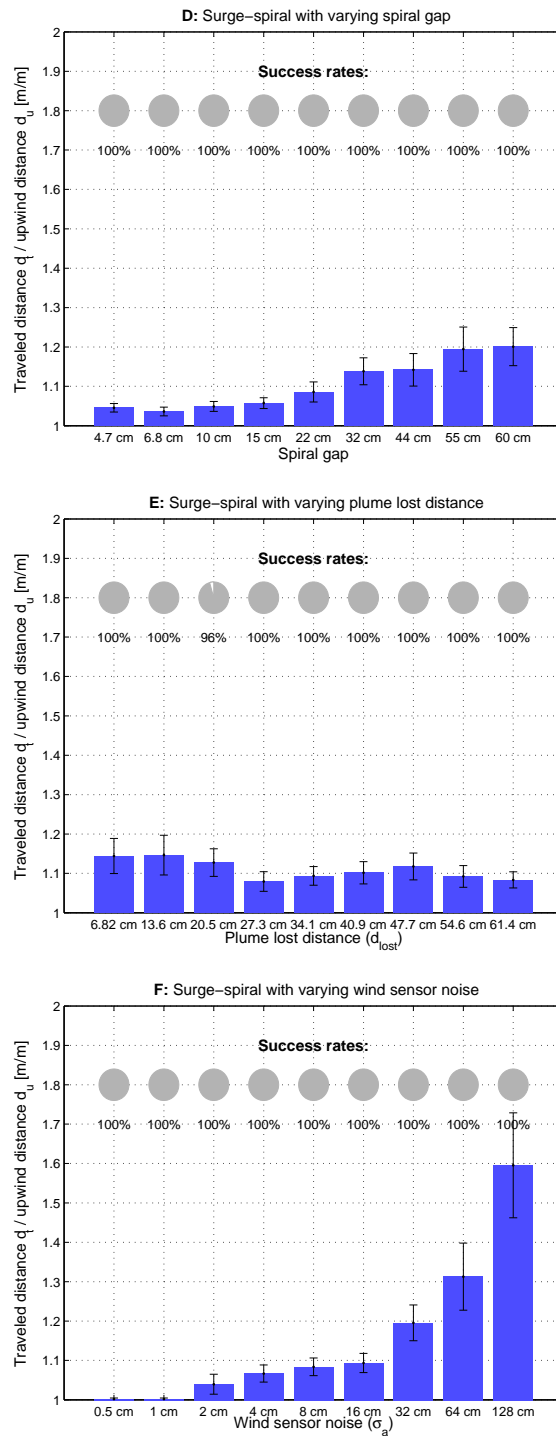


Figure 3.8: Simulation results obtained with the surge-spiral algorithm. The error bars indicate the 95 % confidence interval for the mean (assuming normally distributed data). **D:** With varying spiral gap (d_{gap}). **E:** With varying plume lost distance (d_{lost}). **F:** With varying noise on the wind sensor reading (σ_a).

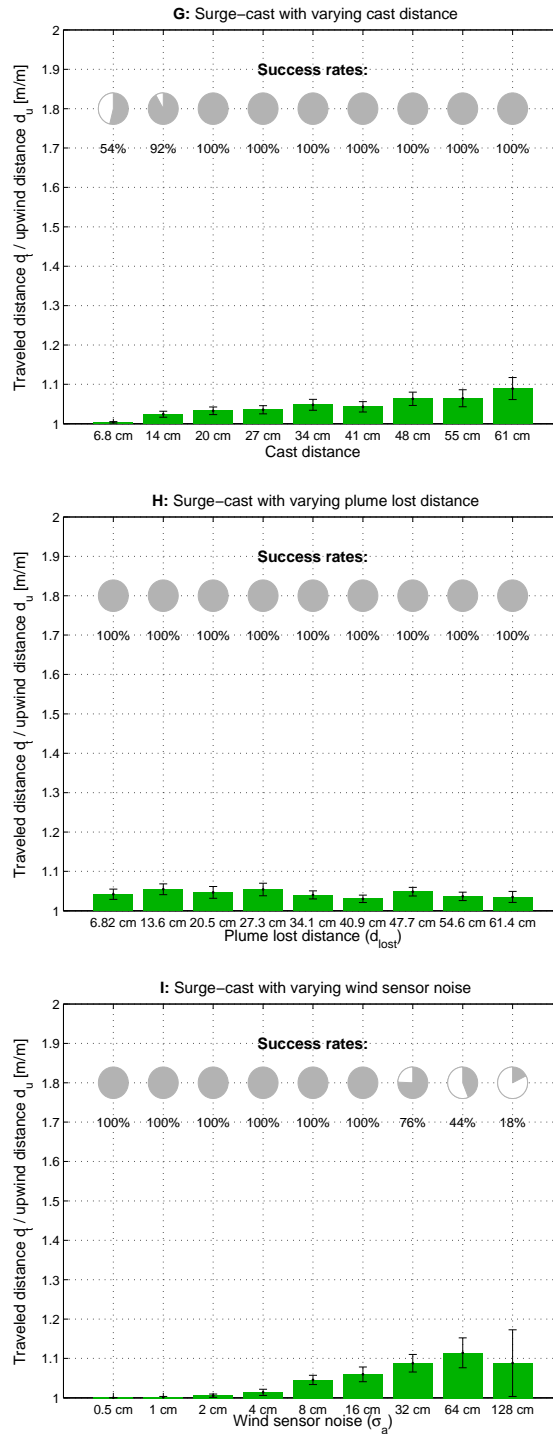


Figure 3.9: Simulation results obtained with the surge-cast algorithm. The error bars indicate the 95 % confidence interval for the mean (assuming normally distributed data). **G:** With varying cast distance (d_{cast}). **H:** With varying plume lost distance (d_{lost}). **I:** With varying noise on the wind sensor reading (σ_a).

This, however, is simply due to the fact that the upwind steps get larger, and could have a negative influence in non-laminar flow conditions. In contrast to the casting algorithm, wind sensor accuracy only affects the distance overhead of *surge-spiral*, and not its success rate. For high noise values, the distance overhead becomes significantly larger, as the algorithm more often fails guessing on which side the plume is.

3.2.2.3 Surge-Cast

The results obtained with the *surge-cast* algorithm are comparable to those of the *surge-spiral* algorithm. As expected, the distance overhead grows almost linearly with the cast distance, but at a fairly low rate. Furthermore, for very low cast distances, the algorithm fails to work reliably - the robot simply does not get back to the plume.

Furthermore, the wind sensor noise seems to mainly affect the success rate, which we have observed with the *casting* algorithm as well.

3.3 Theoretical Analysis of the Laminar Flow Performance

In addition to the real-robot experiments and the simulation experiments, we studied the same three algorithms from a theoretical perspective. Within a simple framework, we derive two equations for each algorithm: an analytical expression for the performance under ideal conditions with a perfect wind sensor, and a probabilistic model taking into account the error of the wind direction sensor. The latter allows us to numerically calculate the performance distribution, which we compare to the results obtained in simulation and with the real robots.

The theoretical model is thereby closely related to the experimental setup used for the real robots and in simulation experiments. However, plume and sensors are abstracted to simple mathematical objects that allow for a mathematical analysis. A comparison of the three models is given in Table 3.2.

Table 3.2: Comparison of the setup for the real-robot and simulation experiments, and the theoretical model.

	Real robots	Simulation	Theory
Environment	wind tunnel	Webots [72]	MATLAB
Wind	\approx laminar	laminar	laminar
Plume	real, ethanol	filaments [7]	straight line
Plume width (w)	\approx 35 cm	35.4 cm	35.4 cm
d_{lost}	\approx 60 cm	61.4 cm	61.4 cm
Robot	Khepera III	Khepera III	point
Locomotion	diff.-drive	diff.-drive	holonomic
Odometry	good	perfect	perfect
Wind sensor error	non-gaussian	$\mathcal{N}(0, (5.7^\circ)^2)$	$\mathcal{N}(0, (5.7^\circ)^2)$
Odor sensor error	negligible	Gaussian, small	0
Odor sensor delay	$t_{90\%} \approx 0.1$ s	none	none

3.3.1 Wind and Plume Model

We consider a 2D space with a perfectly laminar wind flow and a single odor source emitting a chemical substance at constant rate. This substance is only transported by (large-scale) advection.

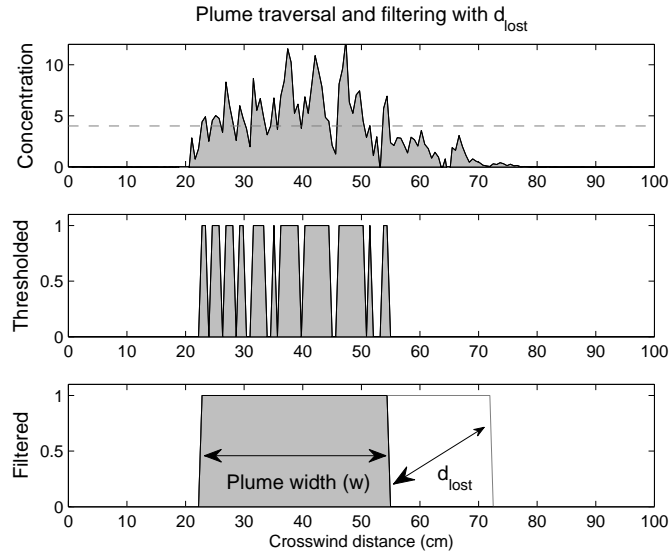


Figure 3.10: Preprocessing of the odor concentration signal by the robots. In the real-robot and simulation experiments, the concentration threshold was tuned manually. Note that the plume lost distance, d_{lost} , is measured along the trajectory of the robot, while the plume width, w , is measured in cross-wind direction.

Small-scale advection (responsible for the intermittent structure of the plume) and diffusion (an effect a few orders of magnitude smaller) are not modeled. For simplicity and without loss of generality, the wind is blowing in positive x direction at a speed of 1 m/s, i. e.,

$$a(u) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ [m/s]} \quad (3.1)$$

at any position u .

Since the algorithms only take a binary input value from the odor sensor, we model our plume as a straight line of constant width w starting at the odor source and extending to infinity in the direction of the wind. The robot — modeled as a point in 2D space — is considered *in the plume* if it lies on this line, and *out of the plume* otherwise.

While this model is far from physical reality, the behavior of all three algorithms in such a simplified model is approximately the same as in the real plume. As the algorithms pass the (binary) odor sensor input through a filter to smooth out all “gaps” shorter than the distance d_{lost} (see Figure 3.10), there is no need to model these gaps. This is actually the purpose of that filter, which has been shown to work well in the experiments with the real robot.

3.3.2 Wind Direction Sensor Model

The output of the wind direction sensor at position u , $\mathbf{a}_s(u)$, is modeled as an unbiased sensor with added Gaussian noise. That is,

$$\mathbf{a}_s(u) = a(u) + \mathbf{v}_a \quad (3.2)$$

where

$$\mathbf{v}_a \sim \begin{pmatrix} \mathcal{N}(0, \sigma_a^2) \\ \mathcal{N}(0, \sigma_a^2) \end{pmatrix} \text{ [m/s]} \quad (3.3)$$

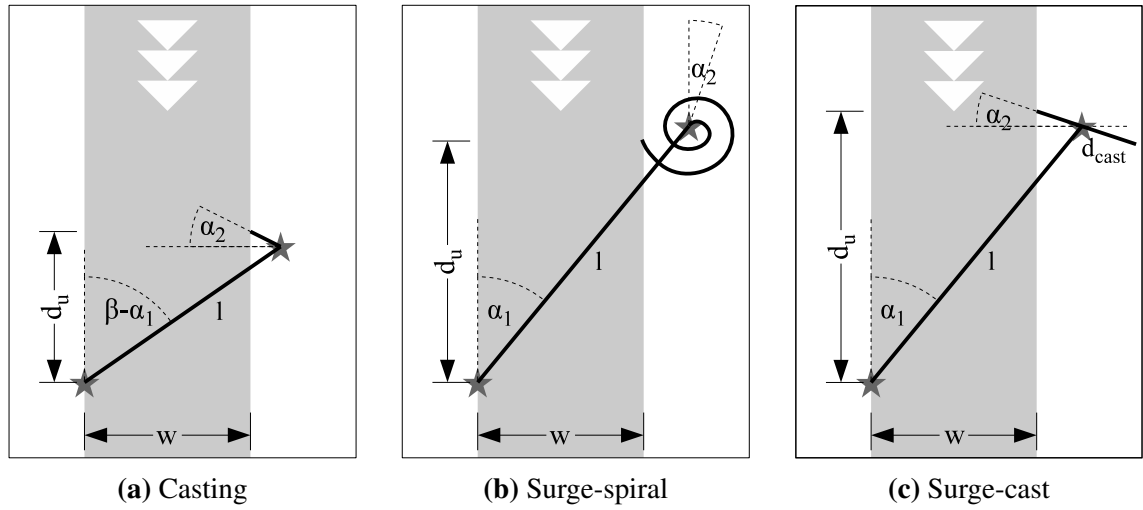


Figure 3.11: Basic patterns (building blocks) of the three algorithms. In one iteration, the robot moves along the thick line and takes wind sensor measurements (α_1 and α_2) at the places indicated by the stars.

This is the same model that we used in simulation. Even though the noise is added to the X and Y components of the wind vector, the distribution of the angular noise is approximately Gaussian as well for sufficiently small σ_a . Each wind direction measurement is therefore susceptible to an angular error modeled by the random variable

$$\alpha \sim \mathcal{N}\left(0, \frac{\sigma_a^2}{|a(u)|}\right) = \mathcal{N}(0, \sigma_a^2) \quad \text{with } \sigma_a = 0.1 \text{ [rad]} \quad (3.4)$$

3.3.3 Performance of the Casting Algorithm

We now calculate the distance overhead for all three algorithms with and without taking into account the wind direction sensor noise. For the case without noise, we derive an expression for the mean distance overhead, d_o , while for the case with the wind direction sensor noise, we numerically calculate the distribution of distance overhead and the mean of the success rate, s_r .

The procedure is the same with all three algorithms, as they all proceed by repeating a basic pattern until the source is found. These basic patterns are depicted in Figure 3.11. Each repetition of this pattern (called *iteration* in the remainder of this chapter) brings the robot closer to the source, but also entails a certain probability to lose the plume completely. In this section, we present our approach in details for the *casting* algorithm, while the following sections only provide the equations for the other two algorithms.

3.3.3.1 Ideal Wind Direction Sensor

With an ideal wind sensor ($\alpha_1 = 0$, $\alpha_2 = 0$), the trajectory produced by the *casting* algorithm in our theoretical model is deterministic. Its distance overhead can be written as

$$d_o(\beta) = \frac{\frac{1}{\sin\beta} + f(1 + \sin\beta)}{\left(\frac{1}{\sin\beta} + f\right)\cos\beta} \geq \frac{1}{\cos\beta} \quad (3.5)$$

with

$$f = \frac{d_{\text{lost}}}{w} \quad (3.6)$$

Even though this expression may look complicated, it can easily be derived by looking at the geometry of the trajectory.

3.3.3.2 Noisy Wind Direction Sensor

Distribution of one iteration With the *casting* algorithm, the basic pattern is produced by the following two steps:

1. Move upwind with an upwind angle β until the plume is lost for a distance greater than d_{lost} .
2. Move cross-wind until the plume is found again. Whether to turn left or right for this cross-wind motion is decided using the measured wind direction (with respect to the robot's heading).

Both steps consist of measuring the wind direction, turning towards the respective angle and moving forward while continuously sampling the odor concentration. While sampling speeds and accelerations are large enough to be ignored, the wind direction measurement introduces a non-negligible error. Each of the two readings is susceptible to noise modeled by the random variables α_1 resp. α_2 . Note that as each reading is assumed to be independent, α_1 and α_2 are independent as well. Hence, the robot actually goes upwind with an angle $\beta - \alpha_1$ and cross-wind with $\frac{\pi}{2} + \alpha_2$.

Under these assumptions, we can — for a single iteration — calculate the distribution of the distance that the robot covers (\mathbf{d}_t), the distribution of the distance by which it approaches the source (\mathbf{d}_u), as well as the probability that the robot loses the plume completely and fails the run ($1 - s$). Using trigonometry, the following equations³ are obtained:

$$\mathbf{d}_t = \mathbf{l} + d_{\text{lost}} \frac{\sin|\beta - \alpha_1|}{\cos \alpha_2} \quad (3.7)$$

$$\mathbf{d}_u = \mathbf{l} \cos(\beta - \alpha_1) + d_{\text{lost}} \sin|\beta - \alpha_1| \tan \alpha_2 \quad (3.8)$$

$$s = \begin{cases} 1 & \text{if } \alpha_2 < \beta - \alpha_1 \\ 0 & \text{otherwise} \end{cases} \quad (3.9)$$

where

$$\mathbf{l} = \frac{\mathbf{w}'}{\sin|\beta - \alpha_1|} + d_{\text{lost}} \quad (3.10)$$

$$\mathbf{w}' = \begin{cases} w & \text{if } \alpha_1 < \beta \\ 0 & \text{otherwise} \end{cases} \quad (3.11)$$

Note that \mathbf{d}_t , \mathbf{d}_u and s are dependent random variables, as they are generated using the same samples of α_1 and α_2 .

The mean success probability of a single iteration can be calculated by marginalizing over α_1 and α_2 :

$$E(s) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P(s|\alpha_1, \alpha_2) P(\alpha_1) P(\alpha_2) d\alpha_1 d\alpha_2 \quad (3.12)$$

The mean distance overhead, $E(\frac{\mathbf{d}_t}{\mathbf{d}_u})$, of one iteration could be calculated in a similar fashion, but is not of particular interest.

³To simplify the notation, we use random variables (i. e., distributions) as if they were regular variables (e. g., a sample of that distribution), but write them in bold font.

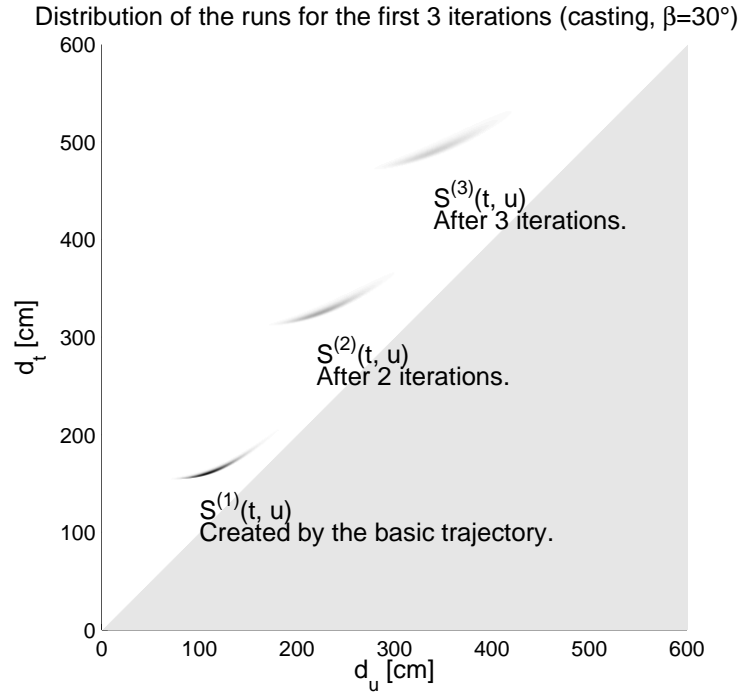


Figure 3.12: $(\mathbf{d}_t, \mathbf{d}_u)$ space with the first 3 iterations (overlaid) of the casting algorithm with $\beta = 30^\circ$. $S^{(1)}(t, u)$ is the distribution for a single iteration and all other distributions are convolutions of it. The gray shaded lower right triangle is impossible to reach because $d_t > d_u$ (by construction).

Distribution of a whole run What we would like to calculate instead is the distance overhead of a complete run. To do that, we need to combine the iteration distributions until the upwind distance exceeds 14 m. Let us define

$$S^{(1)}(t, u) = P(\mathbf{d}_t = t, \mathbf{d}_u = u, \mathbf{s} = 1) \quad (3.13)$$

as the distribution of the successful runs after *one* iteration in the space spanned by d_t and d_u . Note that $S(t, u)$ is not a probability density function in the strict sense, because

$$\int_{-\infty}^{\infty} S^{(1)}(t, u) dt du = E(\mathbf{s}) \leq 1 \quad (3.14)$$

While the distribution of the successful runs after two iterations, $S^{(2)}(t, u)$ is simply the convolution of $S^{(1)}(t, u)$ with itself, the distribution after three iterations, $S^{(3)}(t, u)$, is the convolution of $S^{(2)}(t, u)$ with $S^{(1)}(t, u)$, and so on. Hence, we can combine any number of iterations by applying the convolution equation,

$$S^{(i)}(t, u) = \int_{-\infty}^{\infty} S^{(i-1)}(t-t_1, u-u_1) S^{(1)}(t_1, u_1) dt_1 du_1 \quad (3.15)$$

once for each added iteration. Note that this is only valid because iterations are mutually independent, i. e., $\alpha_1^{(j)}$ is indep. of $\alpha_1^{(k)}$, and $\alpha_2^{(j)}$ is indep. of $\alpha_2^{(k)}$, $\forall j \neq k$. As an example, the first three iterations for $\beta = 30^\circ$ are depicted in Figure 3.12.

Since $S^{(1)}(t, u)$ does not include the failing runs after one iteration, $S^{(i)}(t, u)$ do not include them neither. Hence, the fraction of successful runs after i iterations is simply

$$E(\mathbf{s}^{(i)}) = \int_{-\infty}^{\infty} S^{(i)}(t, u) dt du = (E(\mathbf{s}))^i \leq 1 \quad (3.16)$$

To calculate the distribution of complete runs, it suffices to combine iterations until

$$S^{(i)}(t, u) = 0 \quad \forall u < 14 \text{ m} \quad (3.17)$$

and to collect statistics.

Implementation Issues Carrying out the above procedure analytically is clearly not viable. However, the equations can easily be solved numerically, either with a Monte-Carlo approach (randomly selecting input samples and calculating a histogram of the output samples) or by discretizing the distributions and calculating the joint distributions numerically. We have chosen the second approach, as it provides smoother output distributions. This corresponds to particle filtering with fixed particles placed on a regular grid.

The implementation is fairly straightforward. We have chosen a resolution of 0.02° for α_1 and α_2 and cut them off at $\pm 25^\circ$. The distribution $S^{(i)}(t, u)$ was approximated with a 30 m by 30 m square lattice with a resolution of 1 cm in each direction. Care must be taken with big values for d_t and d_u that do not fit within this lattice, as one would lose probability mass when ignoring them. We simply added these values to the cell (30 m, 30 m) which approximates their $\frac{d_t}{d_u}$ ratio with 1.

In addition, equation (3.15) is in $O(n^2)$ where n denotes the number of cells of $S^{(i)}(t, u)$, which can be large. Since most values in this matrix are zero or very small, however, the algorithm can be boosted by using sparse matrices. After each iteration, we furthermore removed all values with $d_u > 14$ m after having calculated the necessary statistics.

3.3.3.3 Results

Figure 3.13 shows the distance overhead of the *casting* algorithm, and Figure 3.14 the corresponding success rates. For comparison purposes, we overlaid the real-robot and simulation results on both plots.

For $\beta > 15^\circ$, the theoretically derived distance overhead distribution is almost normal and in accordance with our previous findings [80].

The simulation results (boxes) match very well with the theoretical distribution. Small differences with larger upwind angles could be due to the placement of the odor sensor. In the theoretical model, this sensor was assumed to be centered on the robot, while the real sensor was put in front of the robot, at about 7 cm from its kinematic center.

In addition, equation (3.5) (dashed line) for an ideal wind sensor is a very good approximation of the mean obtained with our non-ideal wind sensor. While an ideal wind sensor would allow us to reach the optimal performance for very steep upwind angles, its performance is slightly worse for $\beta > 8^\circ$. Randomness can indeed boost the performance here, as the relationship between performance and effective upwind angle is not linear.

The real-robot results are significantly worse in terms of distance overhead, but better when comparing the success rate (except for $\beta = 30^\circ$). Reasons for this are believed to be two-fold. First, closer inspection of the real robot trajectories revealed that the cross-wind angle was almost systematically biased towards the downwind direction. Similarly, the actual upwind angle was $2^\circ - 5^\circ$ higher than what was configured. This is an artifact of the measurement resolution of the wind direction sensor, which was only 10° [78]. Second, the flow right in front of the odor source was slightly turbulent and sometimes caused additional errors in the wind direction measurement. Even though these were manually removed in the most detrimental cases, the trajectories close to the source are still slightly less ideal.

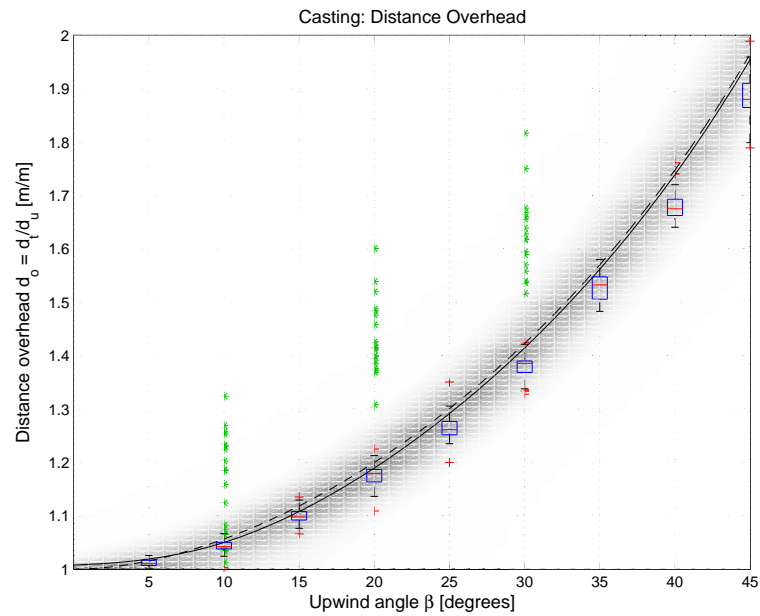


Figure 3.13: Theoretical and experimental distance overhead of the casting algorithm. Green stars: Real-robot experiments (20 runs each). Blue boxes and red crosses: Box-plot of the simulation results (50 runs each). The box shows the lower/upper quartile and the red line denotes the median. Red crosses stand for outliers. Gray shading: Theoretically derived distribution of the distance overhead for a noisy wind direction sensor. Solid black line: Mean of the latter. Dashed black line: Expected performance with an ideal wind direction sensor, calculated using equation (3.5).

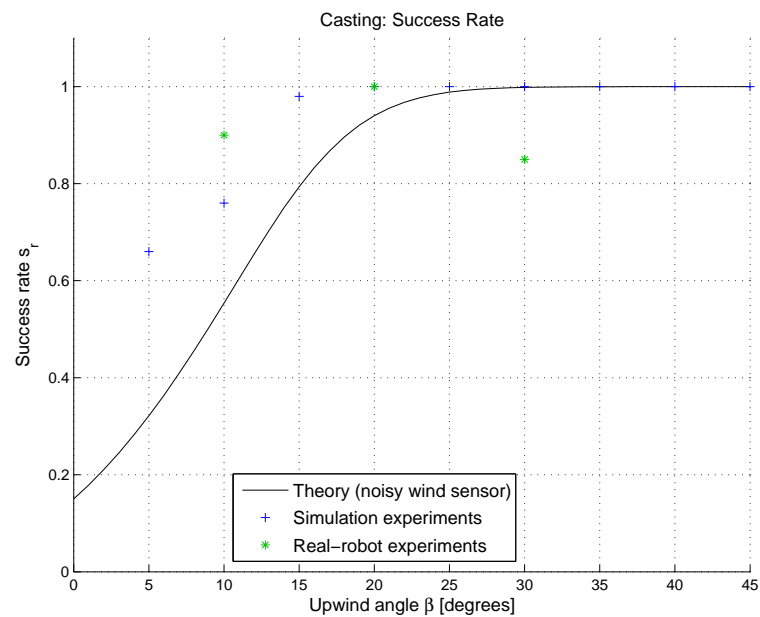


Figure 3.14: Theoretical and experimental success rate of the casting algorithm.

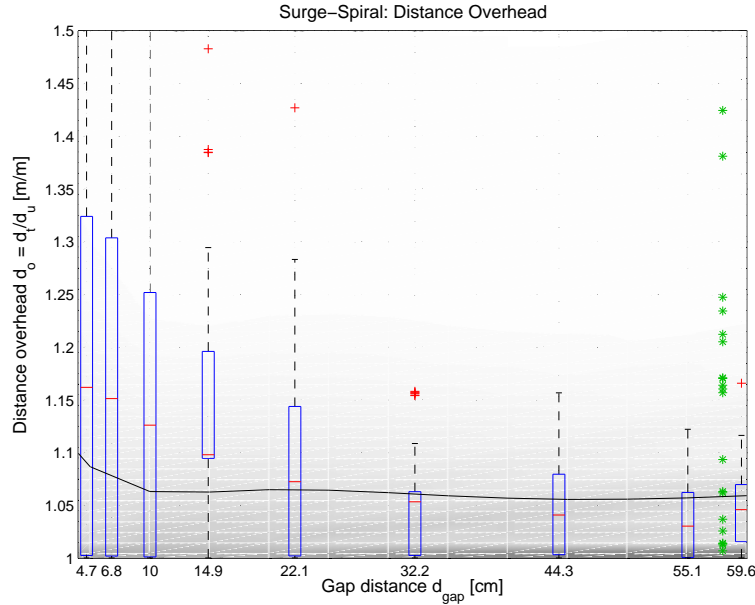


Figure 3.15: Theoretical and experimental performance of the surge-spiral algorithm. Green stars: Real-robot experiments (20 runs each). Blue boxes and red crosses: Box-plot of the simulation results (50 runs each). The box shows the lower/upper quartile and the red line denotes the median. Red crosses stand for outliers. Gray shading: Theoretically derived distribution of the distance overhead. Black line: Expected mean distance overhead, calculated from the distribution.

3.3.4 Performance of the Surge-Spiral Algorithm

3.3.4.1 Ideal Wind Direction Sensor

With an ideal wind direction sensor, the distance overhead of the *surge-spiral* is simply

$$d_o(d_{gap}) = 1 \quad (3.18)$$

Since the robot starts in the plume and moves straight upwind in this ideal plume, it never leaves it. Hence, in contrast to the *casting* algorithm, the *surge-spiral* algorithm achieves optimal performance under ideal conditions.

3.3.4.2 Noisy Wind Direction Sensor

With a noisy wind direction sensor, we again break the basic pattern of the algorithm into two steps:

1. Move straight upwind until the plume is lost for a distance greater than d_{lost} . Due to the wind direction measurement error, the actual upwind angle is α_1 .
2. Move along an Archimedean spiral until the plume is found again. The wind direction measurement here (α_2) only serves to decide whether to start the spiral towards left or right.

The *surge-spiral* algorithm does not have any failure condition⁴. Under the ideal assumptions taken here and the fact that the spiral increases, the robot will eventually reacquire the plume. The

⁴In the simulation and real robot experiments, the only condition for failure was when the robot touched the arena wall. This, however, was very unlikely even with the real robots and never happened.

three distributions therefore are

$$\mathbf{d}_t = \mathbf{l} + r_l(d_{\text{gap}}, d_{\text{lost}} \mathbf{b} \sin |\alpha_1|) \quad (3.19)$$

$$\mathbf{d}_u = \mathbf{l} \cos \alpha_1 + r_y(d_{\text{gap}}, d_{\text{lost}} \mathbf{b} \sin |\alpha_1|) \quad (3.20)$$

$$\mathbf{s} = 1 \quad (3.21)$$

where

$$\mathbf{l} = \begin{cases} d_{\text{lost}} & \text{if } \alpha_1 < 0 \\ \frac{w}{\sin \alpha_1} + d_{\text{lost}} & \text{otherwise} \end{cases} \quad (3.22)$$

$$\mathbf{b} = \begin{cases} -1 & \text{if } \alpha_2 < \alpha_1 < 0 \\ -1 & \text{if } 0 < \alpha_1 < \alpha_2 \\ 1 & \text{otherwise} \end{cases} \quad (3.23)$$

$r_l(d_{\text{gap}}, x)$ and $r_y(d_{\text{gap}}, x)$ are the trajectory length resp. the upwind component of the spiraling maneuver with spiral gap d_{gap} and distance x from the plume. As these values are difficult to calculate analytically, we numerically integrated over a spiral trajectory to find them. This is much more precise than approximating the spiral with a circle and straightforward to implement.

Since $\sin \alpha_1 \rightarrow 0$ for small α_1 , it is clear that a good wind direction sensor will significantly increase the upwind step length, and therewith significantly improve the performance of the algorithm.

The rest of the calculation is exactly the same as introduced in Section 3.3.3.

3.3.4.3 Results

Figure 3.15 shows the distance overhead for the *surge-spiral* algorithm. Despite the high variance of the simulation results, the overall match between simulation and theory is pretty good. Both capture the drop in performance for small spiral gaps, and both predict a fairly constant performance over a wide range of larger gap distances.

As opposed to the *casting* algorithm, the distribution generated by *surge-spiral* is almost exponential. While most runs yield a good distance overhead value, some runs are very bad. Indeed, a number of outliers can be observed in the simulation and real-robot results (which are available for $d_{\text{gap}} = 58$ cm only).

For large d_{gap} values, the theoretically derived distance overhead distribution becomes slightly bumpy. This is a result of the discrete number of iterations that the algorithm performs. The number of real-robot and simulation runs is too small to observe the same effect there.

3.3.5 Performance of the Surge-Cast Algorithm

3.3.5.1 Ideal Wind Direction Sensor

For the same reasons as for the *surge-spiral* algorithm, the distance overhead for the *surge-cast* algorithm under ideal conditions is

$$d_o(d_{\text{cast}}) = 1 \quad (3.24)$$

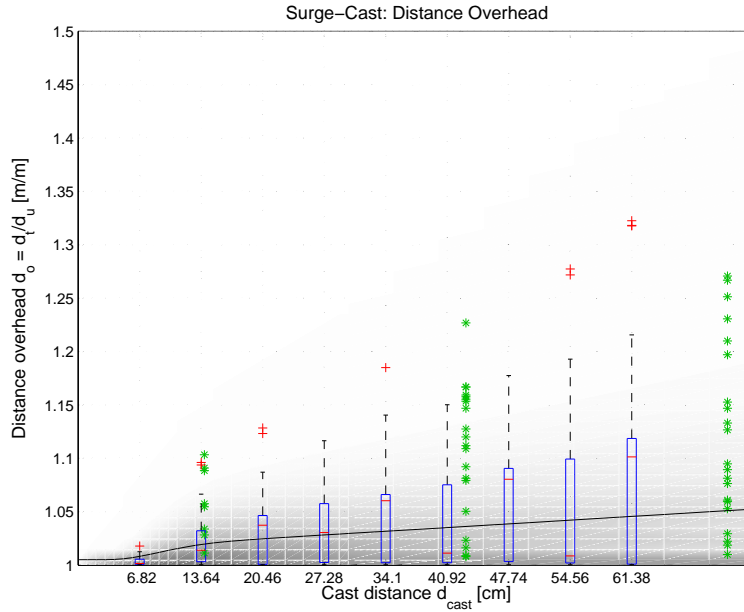


Figure 3.16: Theoretical and experimental performance of the surge-cast algorithm. Green stars: Real-robot experiments (20 runs each). Blue boxes and red crosses: Box-plot of the simulation results (50 runs each). The box shows the lower/upper quartile and the red line denotes the median. Red crosses stand for outliers. Gray shading: Theoretically derived distribution of the distance overhead. Black line: Expected mean distance overhead, calculated from the distribution.

3.3.5.2 Noisy Wind Direction Sensor

Since *surge-cast* and *surge-spiral* only differ in their plume reacquisition strategy their equations look very similar:

$$\mathbf{d}_t = \mathbf{l} + \mathbf{c}_t(\alpha_1, \alpha_2, d_{\text{lost}}) \quad (3.25)$$

$$\mathbf{d}_u = \mathbf{l} \cos \alpha_1 + \mathbf{c}_u(\alpha_1, \alpha_2, d_{\text{lost}}) \quad (3.26)$$

$$\mathbf{s} = \begin{cases} 1 & \text{if } d_{\text{lost}} \frac{\sin \alpha_1}{\cos \alpha_2} < d_{\text{cast}} \\ 0 & \text{otherwise} \end{cases} \quad (3.27)$$

Instead of the spiraling maneuver, however, the *surge-cast* algorithm casts to reacquire the plume. The corresponding equations for c_t and c_u are:

$$\mathbf{c}_t = d_{\text{lost}} \frac{\sin |\alpha_1|}{\cos \alpha_2} + \mathbf{b} \quad (3.28)$$

$$\mathbf{c}_u = d_{\text{lost}} (\cos \alpha_1 + \sin |\alpha_1| \tan \alpha_2) \quad (3.29)$$

$$\mathbf{b} = \begin{cases} d_{\text{cast}} & \text{if } \alpha_2 < \alpha_1 < 0 \\ d_{\text{cast}} & \text{if } 0 < \alpha_1 < \alpha_2 \\ 0 & \text{otherwise} \end{cases} \quad (3.30)$$

Note also that the *surge-cast* algorithm fails if the robot does not find the plume by casting backward and forward.

The rest of the calculation is again the same as introduced in Section 3.3.3.

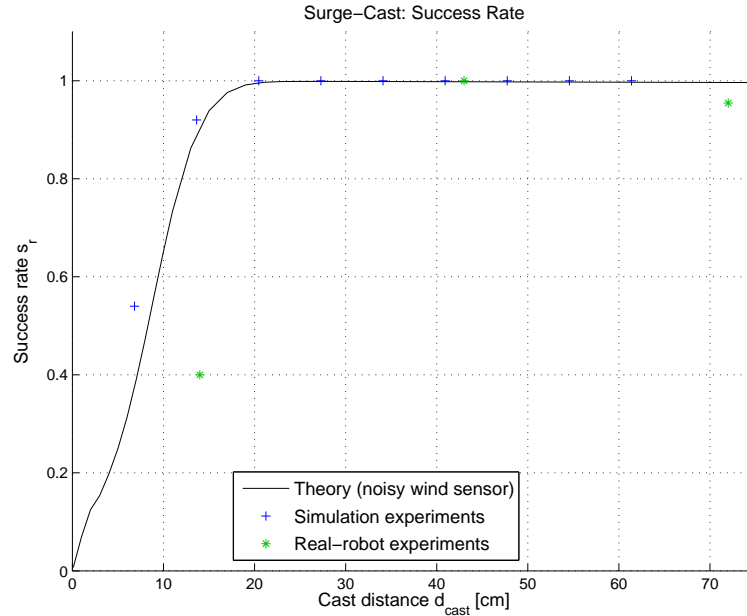


Figure 3.17: Theoretical and experimental success rate of the surge-cast algorithm.

3.3.5.3 Results

The distance overhead and the success rate of the *surge-cast* algorithm are plotted in Figure 3.16 resp. Figure 3.17.

The match between simulation and theory is excellent for both the distance overhead and the success rate. The exponential distribution predicted by the theory is visible on the outliers of the simulation results.

The real-robot results are only slightly worse, but follow the same trends. While a few very good runs can be observed with the real robots, their performance does not exactly follow an exponential law. Closer inspection of the individual runs suggests that this is mainly due to the odometry bias which makes the robot turn slowly when it intends to go straight. Such errors do not have a big impact on a bad run, but makes a perfect run very unlikely.

For both the *surge-cast* and the *surge-spiral* algorithms, the theoretical prediction with an ideal wind sensor does not provide an accurate model of the performance of upwind surge algorithms. At least in laminar flow, these algorithms highly depend on the wind direction sensor and its accuracy.

3.4 Results in Non-Laminar Flow and with Obstacles

All experiments up to this point were carried out in laminar flow. Many real-world applications, however, will have to deal with obstacles and turbulence, and it is important to know if the algorithms are able to deal with such difficulties. In this section, we therefore report on a number of real-robot experiments carried out with different obstacles, and discuss our observations.

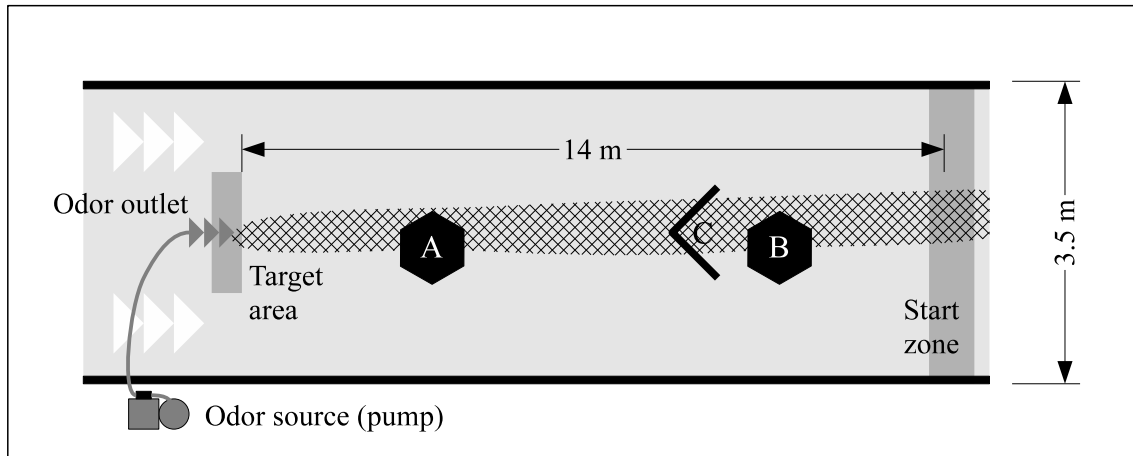


Figure 3.18: Sketch of the experimental setup with obstacles A, B, and C. Note that only one of the three obstacles was placed at a time.

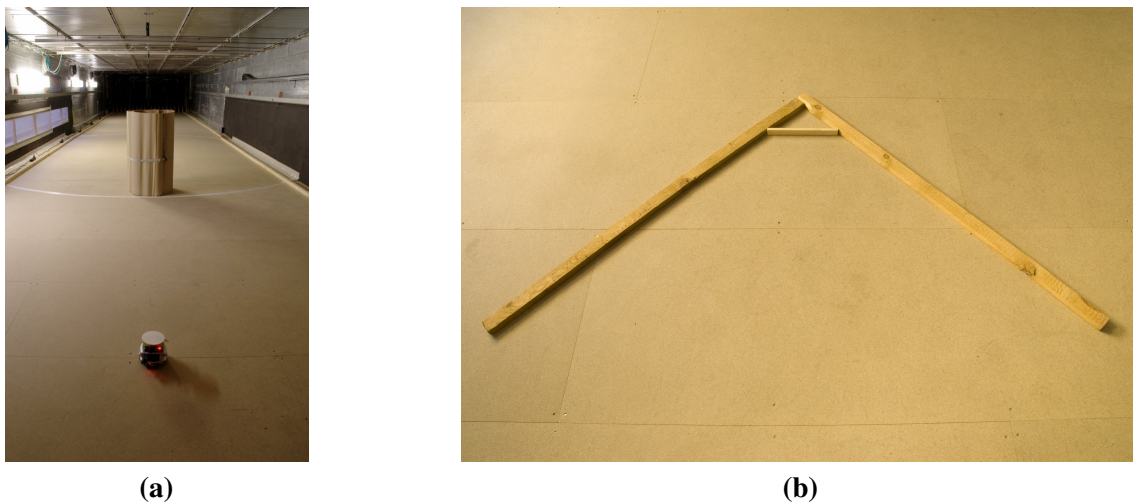


Figure 3.19: (a) Wind tunnel with hexagonal obstacle at position B. (b) V-shape obstacle (C).

3.4.1 Experimental Setup

The experimental setup was similar to the one used in our previous experiments without obstacles (see Section 3.2.1). The experiments were thereby carried out in the wind tunnel with a wind flow of roughly 1 m/s speed. This wind flow is laminar unless obstacles are placed.

We considered the three experimental configurations shown in Figure 3.18. Configuration (A) has a tall obstacle (changing wind flow) placed at 4 m downwind from the source. The shape of this obstacle can be described as a hexagon with an irregular border. Configuration (B) has the same obstacle, but placed 10 m downwind from the source. As the wind flow behind these obstacles was turbulent, the plume got very diluted in these areas. The robot both had difficulties measuring the wind direction and discriminating between plume and fresh air behind the obstacles. Configuration (C) consists of a V-shaped surface obstacle (leaving the wind flow unchanged) placed at 9 m downwind. Pictures of these obstacles are shown in Figure 3.19, and the resulting plume in Figure 3.20.

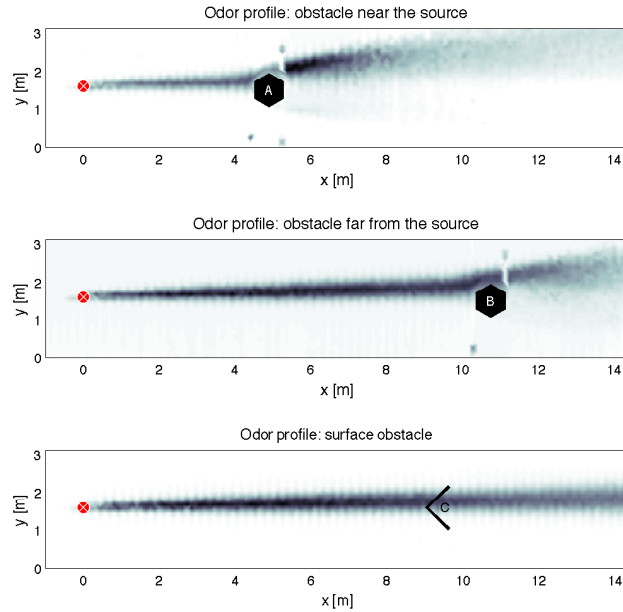


Figure 3.20: From top to bottom: Plume with obstacles at positions A, B and C. Note that concentration levels are relative – only the shape of the plumes can be compared. The plume maps were recorded by systematically scanning the wind tunnel (traversing system) with a MiCS-5521 sensor.

3.4.2 Algorithms

The plume tracking strategy used here is based on the surge-spiral algorithm. To deal with obstacles, we enhanced it with either a Braitenberg [81] obstacle avoidance or wall following algorithm using the 9 infrared sensors of the Khepera III robot. The left and right wheel speeds, s_l and s_r , are thereby calculated as a linear combination of the raw infrared proximity sensor values, v_i , i. e.,

$$s_l = o_l + \sum_{i=1}^9 w_i v_i \quad \text{and} \quad s_r = o_r - \sum_{i=1}^9 w_i v_i \quad (3.31)$$

and the weights, w_i , and bias speeds, o_l and o_r , are chosen such that the resulting behavior is either obstacle avoidance or wall following.

With the obstacle avoidance algorithm, both the surge-spiral algorithm and the Braitenberg obstacle avoidance algorithm are running in parallel, and the output of the surge-spiral algorithm is simply the bias speed for the Braitenberg algorithm. As long as the robot is far away from any obstacle, the Braitenberg weights sum up to zero and leave the surge-spiral algorithm unmodified. When the robot is close to an obstacle, obstacle avoidance dominates surge-spiral.

With wall following, the surge-spiral algorithm is the only active algorithm in open space. When the robot approaches an obstacle, it switches to wall following and sticks to this mode until it has reached the other side of the obstacle. To find out when this has happened, the robot measures the wind direction in regular intervals and switches back to surge-spiral as soon as the wind is blowing towards the obstacle.

To deal with turbulence and dilute plume, the sensory input was processed as follows:

1. If the wind direction sensor returned a low confidence value for a measurement, that measurement was repeated and the confidence threshold decreased.

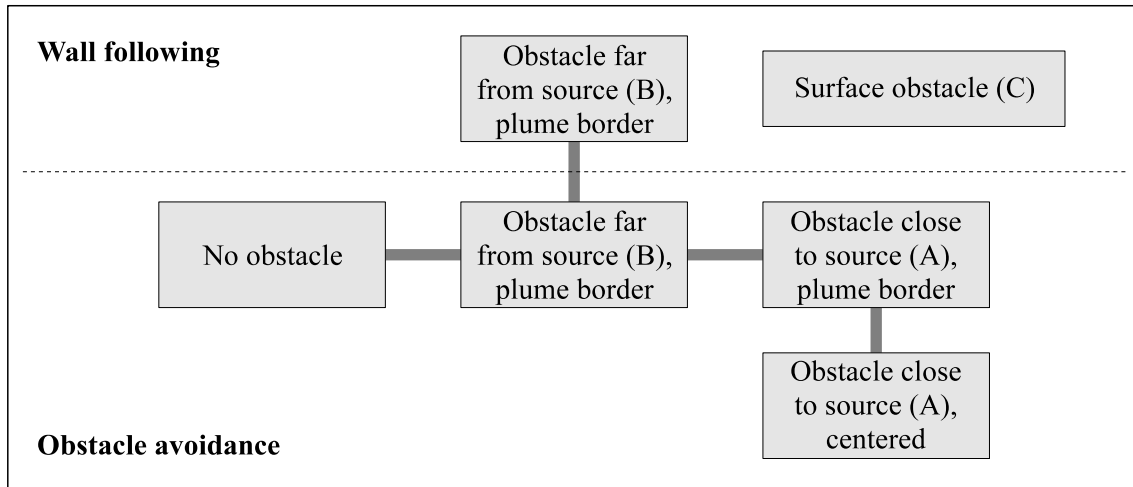


Figure 3.21: Obstacle configurations used in the experiments. Note that only one obstacle was placed at a time. Gray lines connect configurations for which the results can be directly compared.

2. The variable odor threshold, t_i , was dynamically adjusted using the following additive-increase-multiplicative-decrease scheme:

$$t_{i+1} = b_{i+1} + \delta \quad (3.32)$$

$$b_{i+1} = \begin{cases} b_i + \alpha & \text{if } c_i > b_i \\ b_i(1 - \beta) + c_i\beta & \text{otherwise} \end{cases} \quad (3.33)$$

with $\alpha = 0.01$ (experimentally found to be near-optimal) and varying β and δ . While β defines how fast the algorithm adapts to baseline changes, δ affects the width of the plume, as perceived by the robot. c_i denotes the raw measurement, while b_i stands for the variable baseline.

As with our previous experiments, we only consider plume traversal and intentionally omit plume finding (i. e., randomized or systematic search until the plume is found) and source declaration (i. e., declaring that the source is in close vicinity), to prevent those two parts from interfering in the results. Hence, the robot starts in the plume, and source declaration is done by a supervisor (ideal source declaration). Experiments are considered successful when the robot has come in physical vicinity of the source, and unsuccessful if it bumps into an arena boundary.

3.4.3 Results

We run experiments for the 6 different configurations listed in Figure 3.21. That same figure also indicates which configurations can be compared against each other. In the following paragraphs, we compare these configurations and discuss their results.

3.4.3.1 No Obstacle vs. Obstacles A and B

In the first series of runs, we compared the impact of obstacle A and B on the performance. 10 runs each were carried out for different values of β and δ , and the results plotted in Figure 3.22. The Braitenberg obstacle avoidance algorithm was used in all runs. In spite of the small number of runs carried out for each case, the results clearly show that the distance overhead increases and the success rate decreases when obstacles are present. Furthermore, there is evidence that the

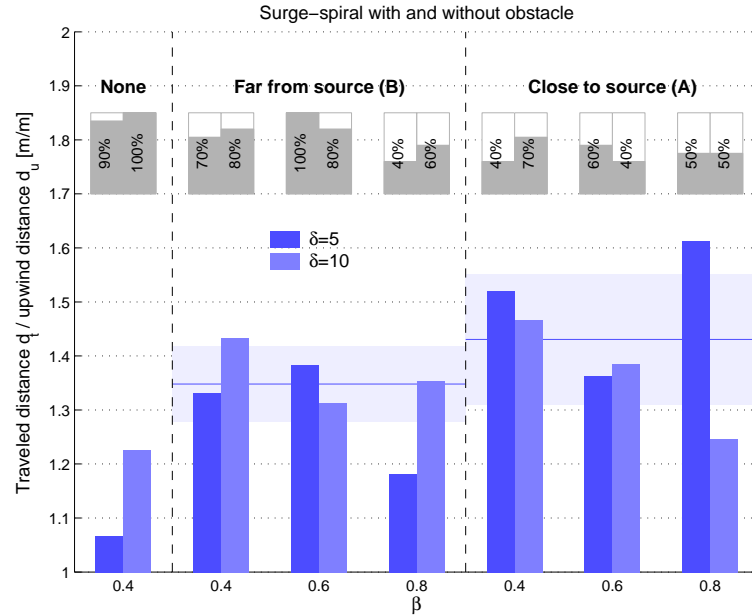


Figure 3.22: Comparison of runs with and without obstacles. The blue bars show the distance overhead (lower is better) and the gray bars indicate the success rate. The blue horizontal line is the mean for the group with the 95 % confidence interval.

results are slightly worse in the case of obstacle A, which confirms our intuition that turbulent flow induces a performance penalty.

Even though the bars seem to suggest that higher β values (i. e., faster threshold adaptation) yield lower distance overheads, there is statistically not enough evidence to support this statement.

3.4.3.2 Obstacle A on Plume Boundary vs. in Center

Figure 3.23 shows the impact of moving obstacle A from the plume boundary to the plume center, with the effect that the plume splits into two almost equal lobes behind the obstacle. It turns out that this has a negative effect on the distance overhead, while keeping the success rates at similar levels.

3.4.3.3 Obstacle Avoidance vs. Wall Following

Finally, we carried out experiments with the wall following algorithm. As shown in Figure 3.24, the success rate jumps to one, at the expense of a slightly higher distance overhead.

3.4.3.4 Surface Obstacles

Braitenberg obstacle avoidance can perform poorly in case of non-convex obstacles. As sketched in Figure 3.25, the obstacle avoidance version of our algorithm is likely to get trapped in obstacle C, while wall following is able to deal with it.

We carried out one set of 10 runs for $\beta = 0.4$ and $\delta = 10$ with both variants of the algorithm. While none of the obstacle avoidance runs succeeded, all wall following runs were successful with a mean distance overhead of 1.59.

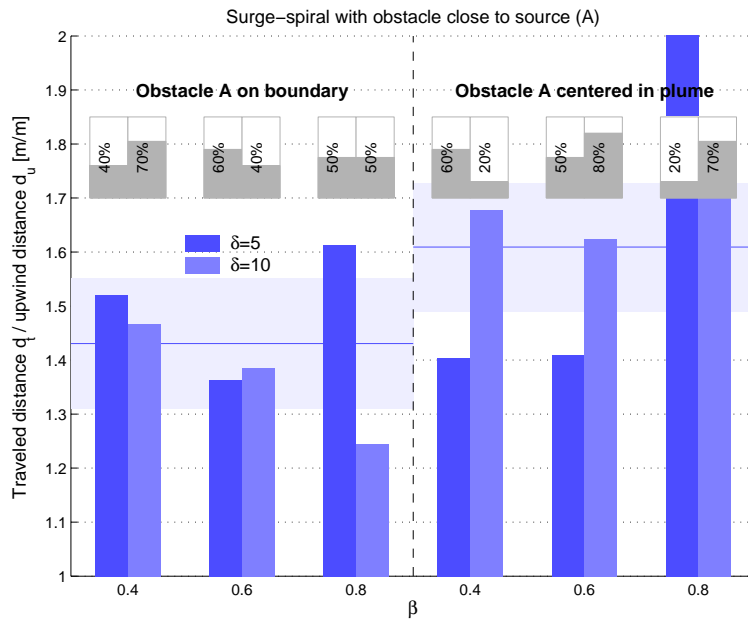


Figure 3.23: Effect of the position of the obstacle within the plume.

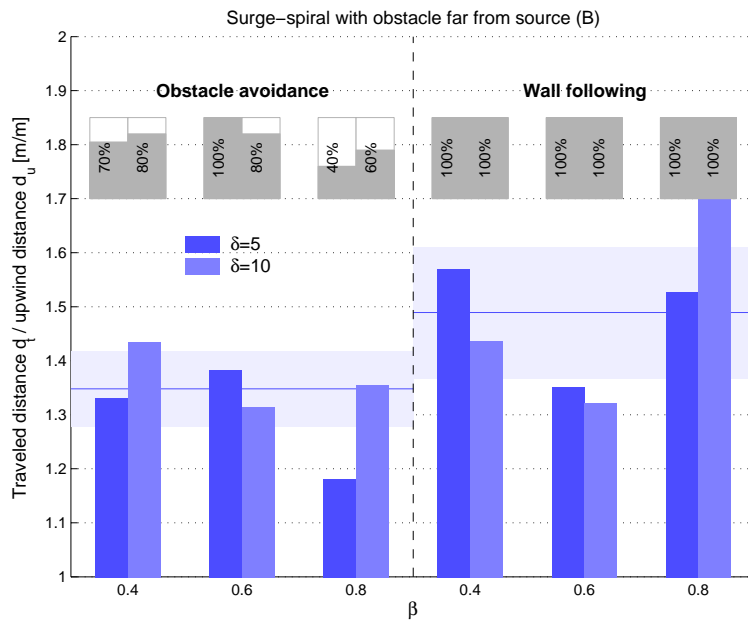


Figure 3.24: Comparison between obstacle avoidance and wall following.

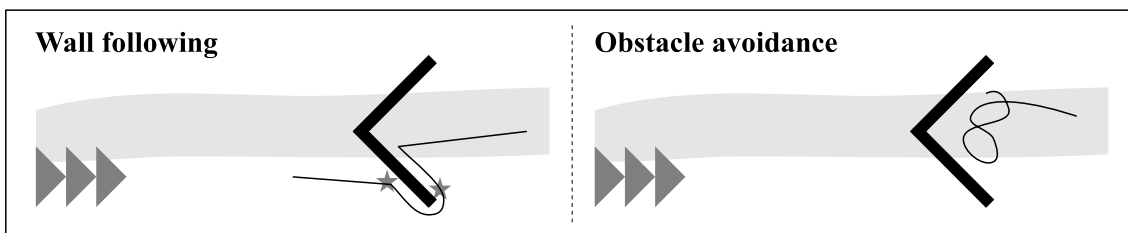


Figure 3.25: Wall following vs. Braitenberg obstacle avoidance with non-convex obstacles.

3.4.4 Discussion

Three main conclusions can be drawn from the experiments.

First, obstacles induce a penalty both because of the path constraints and the turbulence downwind the obstacle. The latter causes the plume to get diluted and become more peaky, but also affects the wind direction sensor accuracy which we have previously shown to have a big impact on the performance. As turbulent flow is difficult to simulate with computer programs [7], real-robot experiments seem to be more suitable to study the effect of obstacles.

Second, both Braitenberg obstacle avoidance and wall following are able to deal with convex obstacles. Simple obstacle avoidance yielded a slightly better distance overhead in our experiments, but wall following scored with high success rates. For non-convex obstacles, however, wall-following is the preferred technique.

Finally, the surge-spiral algorithm seems to be a good candidate algorithm for plume traversal in complex environments. A few initial runs (not systematically recorded) with the casting algorithm did not provide satisfactory results in our scenarios.

To our knowledge, this is the first systematic real-robot study on the performance of plume traversing algorithms in environments with obstacles. It is clear that the scenarios chosen here are not representative for the vast amount of potential real-world scenarios, and it may be too early to generalize the results presented here.

The effect of obstacles on odor source localization definitely needs to be studied further. Potential future research directions include studies with more than one obstacle, moving obstacles, or obstacles in immediate proximity of the source.

3.5 Multi-Robot Experiments without Collaboration

We have so far shown a number of results obtained for a single robot tracking a plume up to the source with three different algorithms. In this section, we are studying the same algorithms with multiple non-cooperating robots. In particular, we compare the performance difference when moving from a single-robot to a homogeneous multi-robot system with 2 or 5 robots. Our performance metric is again the distance overhead (traveled distance d_t divided by upwind distance d_u). As we require only one robot to reach the odor source, we use the distance overhead of the first robot to reach the source as the performance of the robotic team.

As sketched in Figure 3.26, the difference in distance overhead between single-robot (**A**) and non-cooperating multi-robot systems (**C**) consists of two components. First, randomness due to the noise in the system boosts the performance of the system (Figure 3.26 (**1**)). This performance gain can be calculated by using the distribution of the distance overhead of single-robot experiments, and would be achieved if the robots were not interacting with each other (**B**). Second, physical interference among the robots result in a loss in performance (Figure 3.26 (**2**)), which we quantify by running simulations in a realistic robotic simulator [72]. Cooperation among robots (**D**, not discussed in this section) would again result in a performance gain.

3.5.1 Expected Performance of Multi-Robot Experiments

Assume a performance value q that can be associated with each experimental run. In our case, this metric is the distance overhead (traveled distance d_t divided by upwind distance d_u) of the first robot that reaches the source. Hence, a small q value stands for a good performance, with $q = 1$ being the optimum.

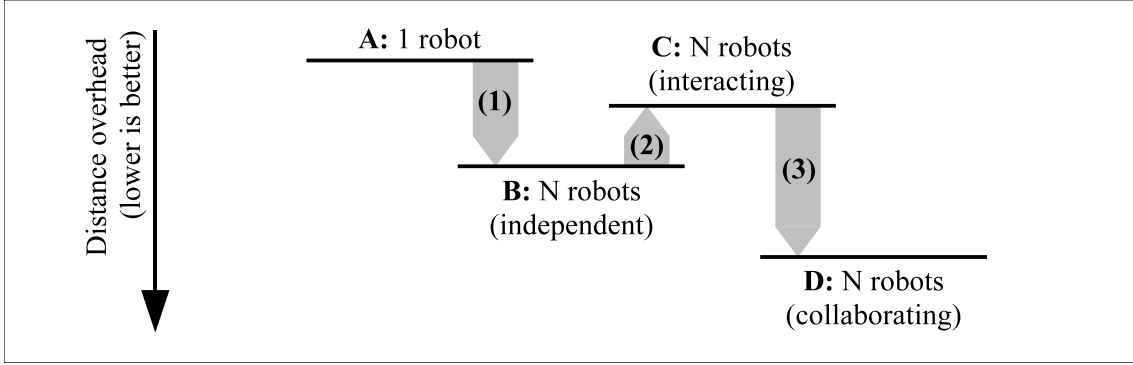


Figure 3.26: Distance overhead of single-robot vs. multi-robot systems. (1) Performance gain due to randomness (mathematically derived in Section 3.5.1). (2) Performance loss due to physical interference among robots (simulated with a robotic simulator). (3) Performance gained with collaboration among robots (not discussed in this section).

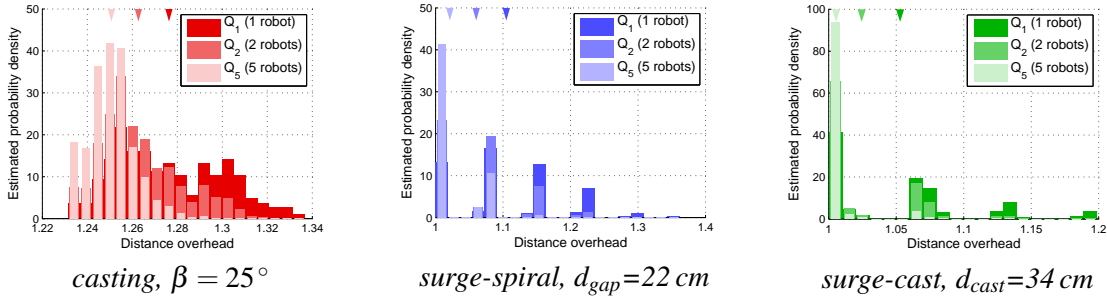


Figure 3.27: Q_1 : Experimentally measured distribution of the distance overhead with a single robot. Q_2 and Q_5 : The expected distributions for 2 resp. 5 robots, based on the assumption that the robots do not physically interfere with each other. The triangles on top of the diagram indicate the mean values of the respective distributions.

The distance overhead of an experiment with a single-robot algorithm (with a fixed set of parameters) can be expressed as a distribution Q_1 , which can be approximated by performing a large number of runs, for instance. Examples of such distributions estimated with 200 runs are shown in Figure 3.27.

If two independent robots are going for the same source, their performances q_a and q_b are random samples drawn from Q_1 . Clearly, the smaller of these two numbers (corresponding to the faster robot) will set the overall performance of the team,

$$q_{ab} = \min(q_a, q_b) \quad (3.34)$$

Hence, the performance distribution of a system with two independent robots is the distribution of q_{ab} , and can be expressed as

$$Q_2(q) = \frac{1}{c_Q} \iint [\min(q_a, q_b) = q] Q_1(q_a) Q_1(q_b) dq_a dq_b \quad (3.35)$$

$$\text{with } c_Q = \iint Q_1(q_a) Q_1(q_b) dq_a dq_b \quad (3.36)$$

where $[\cdot]$ stands for the Iverson bracket. Generalizing this for N robots is straightforward.

3.5.1.1 Calculating Q_N

Closed-form expressions for Q_N only exist for a few well-known distributions. If Q_1 is exponentially distributed with mean $\frac{1}{\lambda}$, for instance, then Q_N is exponentially distributed with mean $\frac{1}{N\lambda}$.

The algorithms used in this section yield complicated distributions, however, and an approximation by an exponential distribution would be very rough for the *surge-spiral* and *surge-cast* algorithms, and not justifiable for the *casting* algorithm. We therefore calculated the distributions for multiple robots numerically, by randomly sampling from the distribution Q_1 (Monte-Carlo simulation). Formally, we estimated the distribution Q_N with 100000 samples of the form

$$\min(q_1, q_2, \dots, q_N) \quad (3.37)$$

where q_1, q_2, \dots, q_N are randomly selected performance samples of the single-robot runs. Distributions obtained in this way for $N = 2$ and $N = 5$ robots executing the *casting* algorithm are shown in Figure 3.27. On that figure, it can be observed how the distribution and its mean value shift towards the left (lower distance overhead) as the number of robots increases.

3.5.2 Experiments

For all three algorithms, we run experiments with 9 different parameters (upwind angle β for *casting*, spiral gap d_{gap} for *surge-spiral*, and cast distance d_{cast} for *surge-cast*), each with 1, 2, or 5 robots. The experiments were run in the Webots simulation environment presented in Section 2.2, and the robots were therefore physically interacting with each other, corresponding to case C in Figure 3.26. For $\beta = 25^\circ$, $d_{\text{gap}} = 22$ cm and $d_{\text{cast}} = 34$ cm, we performed 200 independent runs and calculated the ideal performance as described in Section 3.5.1. For all other configurations, 50 runs were carried out.

In each run, the robots were released in the odor at fixed positions (evenly spaced) between 14.5 m and 16 m downwind from the source. If one robot reached the odor source, the run was stopped and considered successful. During the run, the trajectory, the measured odor concentration and the measured wind direction were recorded for each simulation step. Distance and upwind distance were derived from the trajectory.

The forward speed of the robot (on straight lines) was 10.6 cm/s and therefore same as with the real-robot experiments in the wind tunnel. The plume threshold was set to $c = 100$.

3.5.3 Casting

The results for the *casting* algorithm are presented in Figure 3.28. The differences between the single-robot and the multi-robot experiments are very small and statistically not significant for most configurations. However, as a general trend, multiple robots seem to yield slightly better performance for upwind angles $\beta > 20^\circ$, and worse performance otherwise.

As Figure 3.28 (b) reveals, even the ideal performance (for robots that are not physically interfering) is not much better than the single-robot performance. Indeed, single-robot experiments with the *casting* algorithm yield a compact — almost normal — performance distribution with a small variance, and the resulting "left shift" of the distribution for multiple robots is small.

A noticeable gain can be observed for the success rate, however. For small angles, where the success rate with a single robot is small, a team of robots can achieve very high success rates. This robustness is an advantage often cited in the context of multi-robot systems. Surprisingly, physical interference seems not to have a big influence here. As shown in Table 3.3, the actual success rates

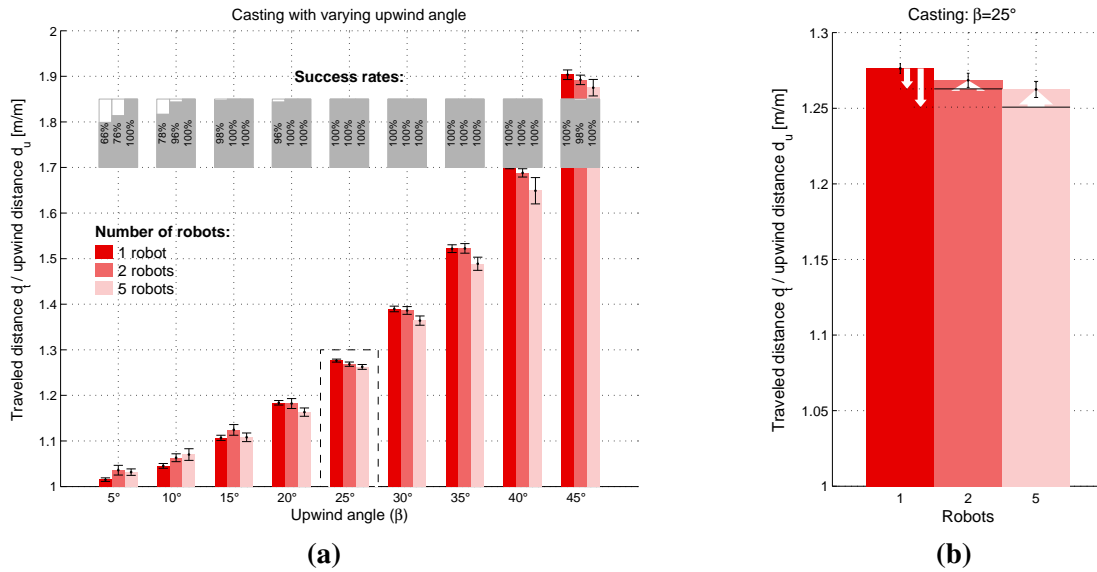


Figure 3.28: (a) Simulation results obtained with the casting algorithm. The error bars indicate the 95% confidence interval for the mean. (b) Close-up for $\beta = 25^\circ$. The thin arrows indicate the intrinsic performance gain by passing from a single-robot to a multi-robot system (Figure 3.26 (1)), while the thick arrows indicate the performance loss due to physical interaction between the robots (Figure 3.26 (2)).

Table 3.3: Comparison between the actual and the expected success rate obtained with the casting algorithm.

Algorithm	1 robot	2 robots		5 robots	
	actual	actual	expected	actual	expected
Casting, $\beta = 5^\circ$	0.66	0.76	0.884	1.0	0.995
Casting, $\beta = 10^\circ$	0.78	0.96	0.952	1.0	0.999

obtained in the multi-robot experiments are close to the expected success rates calculated based on the success rate of the single-robot runs.

3.5.4 Surge-Spiral

The picture for the *surge-spiral* algorithm (see Figure 3.29) looks pretty different. As the performance distribution of single-robot runs resembles an exponential distribution, its mean value decreases as $\frac{1}{N}$ with increasing numbers N of robots. Hence, large performance gains are expected.

For small spiral gaps, no performance gain is noticed in the simulation results. For large gaps, however, the difference between the single-robot and the multi-robot runs are significant. Even though the theoretical model would expect an even larger difference, the multi-robot runs were clearly faster than the single-robot runs and achieved similarly good results as the single-robot runs with small spiral gaps. The reason for this is that a spiraling robot spends enough time aside the plume, such that other robots can safely overtake. This could be interpreted as an indirect and implicit coordination scheme (without communication), whereby robots losing the plume try to make room for other robots in the plume.

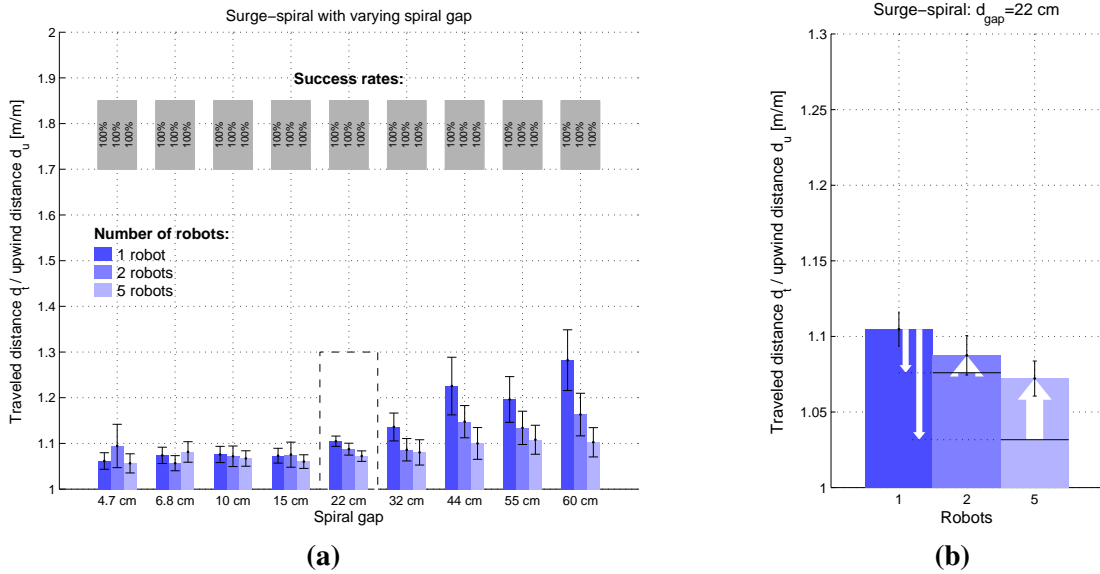


Figure 3.29: (a) Simulation results obtained with the surge-spiral algorithm. The error bars indicate the 95 % confidence interval for the mean. (b) Close-up for $d_{gap} = 22$ cm. The thin arrows indicate the intrinsic performance gain by passing from a single-robot to a multi-robot system (Figure 3.26 (1)), while the thick arrows indicate the performance loss due to physical interaction between the robots (Figure 3.26 (2)).

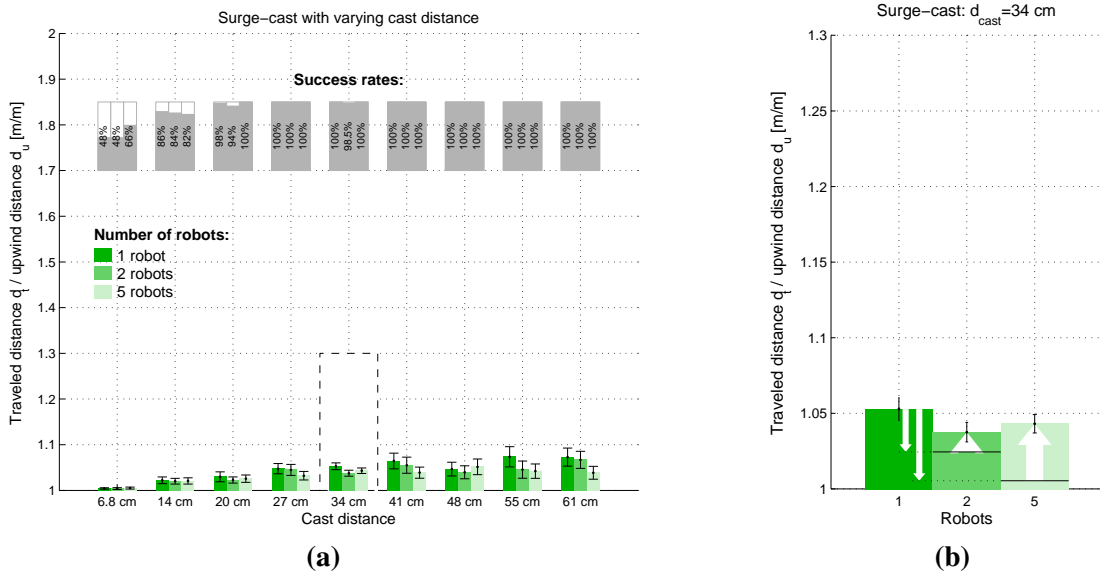


Figure 3.30: (a) Simulation results obtained with the surge-cast algorithm. The error bars indicate the 95 % confidence interval for the mean. (b) Close-up for $d_{cast} = 34$ cm. The thin arrows indicate the intrinsic performance gain by passing from a single-robot to a multi-robot system (Figure 3.26 (1)), while the thick arrows indicate the performance loss due to physical interaction between the robots (Figure 3.26 (2)).

3.5.5 Surge-Cast

The results for the *surge-cast* algorithm are similar, as Figure 3.30 reveals: the longer the cast distance, the more performance is gained by using multiple robots. This has to be taken with a grain of salt, though, since longer cast distances yield worse performance in the single-robot case and are therefore not desired anyway. Hence, in well-configured systems with near-optimal cast distances (here 27–34 cm), no performance gains are visible.

Contrary to the *casting* experiments, using multiple robots does not increase the robustness of the algorithm here. In some cases, the success rate even got worse. The *surge-cast* algorithm in its present form is clearly not robust with respect to physical collisions. Especially during plume reacquisition, a robot blocking the way at the plume boundary can cause another robot to lose the plume completely. This could certainly be improved by adaptively increasing the cast distance until the plume is found.

3.6 Thoughts on Collaboration

The multi-robot experiments without collaboration have shown that the use of multiple robots for plume tracking increases the performance slightly, but not as much as expected by theoretical considerations. The main reason for this is physical interference among collaborating robots. In particular, uncoordinated teams of robots have troubles overtaking each other when moving in the plume. Whenever two robots are coming close to each other, the obstacle avoidance mechanism overrides plume tracking and deviates the robot from its track. For upwind surge, for instance, this means that the robot will almost certainly lose the plume and switch to plume reacquisition.

The underlying problem here is that none of the three algorithms takes the effective robot movement into account. They generate their trajectories with a state machine that is entirely based on odor concentration and wind direction, and assume that the robot moves as dictated by the current state. Anything along this path — such as obstacles, other robots or simply uneven terrain — is a disturbance when following the plume up to its source. None of the algorithms can cope with such disturbances in an efficient manner, as the results of the non-collaborative multi-robot experiments showed.

Hence, for pure plume traversal (without initial plume search⁵), using multiple robots in a non-collaborative fashion is not worth the effort.

The question that naturally arises at this point is whether collaboration could help. But adding collaboration to these algorithms is a double-edged sword. From a theoretical perspective, collaboration should at least yield the same performance as the non-collaborative versions, as more information (position, concentration and wind direction measurements of other robots) is available to the algorithm. However, a reaction based on such information is again a disturbance to the algorithm, which the algorithms are inherently not good at coping with. Hence, without fundamentally changing the algorithms, collaboration is likely to not provide satisfying performance. The results of a student project (not reported in this thesis) underlined this hypothesis, and prevented us from digging any further into this direction.

Considering that these algorithms are inspired by the behavior of moths, this is actually not surprising. Moths, such as many other insects, are tracking plumes in a competitive manner, and not in a collaborative one. Plume tracking is heavily used for food scavenging and mating, both of which are highly competitive activities that are directly linked with the evolutionary success of an individual. Hence, individuals would certainly not share information that may favor other

⁵For search or coverage in general, a number of multi-robot algorithms available, and it is well known that the performance can be greatly enhanced by using multiple robots.

Table 3.4: High-level comparison of the three bio-inspired algorithms.

	Casting	Surge-spiral	Surge-cast
Speed	–	+	++
Robustness	+ / – (1)	++	+
With obstacles	–	+	+
In turbulent flow	– (2)	+ (3)	– (2)
Without wind	– (2)	+ (3)	– (2)
Kinematic constraints	– (4)	+	– (4)
Multi-robot capable	–	–	–
CPU / memory requirements	low	low	low
Self-localization requirements	low	low	low

- (1) Robust configurations are inherently slow.
(2) The algorithm does not work at all in these conditions.
(3) The algorithm (or a close variant) works, even if its performance suffers.
(4) The algorithm includes sharp turns.

individuals. This is not true for animals living in societies, who, for example, actively help each other for food scavenging. But as odors move quickly in the air, fast communication mechanisms, such as audible signals or visual cues, among individuals would be necessary to take advantage of the information acquired by other individuals before this information becomes irrelevant. We are not aware of any such mechanisms with insects. Hence, low-level plume tracking is still done individually, even if higher level tasks (such as food scavenging) are performed together.

This obviously does not mean that it is impossible to come up with a simple multi-robot plume tracking algorithm. We in fact show that this is possible later in Chapter 5. The collaborative plume tracking algorithm introduced there is not bio-inspired, but based on standard engineering principles.

3.7 Summary and Conclusion

Altogether, the experiments presented in this chapter provide a good overall picture of these three bio-inspired algorithms and demonstrate the interplay of the three underlying behaviors (casting, spiraling, and upwind surge) observed in nature.

The main conclusions over all experiments are summarized in Table 3.4.

Pure *casting* is inefficient for large upwind angles, and not very robust for small upwind angles. In addition, casting relies on fairly accurate wind direction measurements and would not work in environment without a main wind flow, for instance. The worst property of this algorithm however is that if anything goes wrong, the algorithm loses the plume and fails completely. Hence, combining the algorithm with other subsystems (e. g., obstacle avoidance) which take over control of the robot for short times is not possible.

Pure casting is an incomplete algorithm. Nevertheless, this was one of the most popular odor source localization strategies in the scientific literature prior to 2005. Many variants have been studied and sometimes referred to as moth-inspired [5] [20]. Moths, however, as other animals, do not rely on pure casting for airborne plume tracking, but on a combination of casting, upwind surge and spiraling [51].

Upwind surge strategies (*surge-spiral* and *surge-cast*) have a big speed advantage, especially if the wind direction can be determined accurately. However, they need to be combined with a

plume reacquisition strategy. Using a local search strategy (e. g., spiraling) to reacquire the plume yields very robust algorithms. Casting for plume reacquisition is faster if reliable wind direction information is available, but less robust. Additional robustness could be added by increasing the cast distance until the plume is found, instead of giving up after one back and forth sweep, or by switching to spiraling whenever casting fails.

As compared to the other two algorithms, the *surge-spiral* algorithm keeps a high success rate even under difficult conditions. Instead, the distance overhead increases. This allows the algorithm to coexist with other controllers in the same system. Whatever disturbance might occur, the algorithm will do its best to move towards the source.

Spiraling is also the only strategy (among spiraling, casting and upwind surge) that can be used in environments without a main wind flow, or on robots that do not have any wind direction sensor on board. The distance overhead would suffer significantly, of course, but the strategy still works.

Among the three algorithms tested in this chapter, the *surge-spiral* algorithm is therefore the preferred strategy.

4 Probabilistic Algorithms

In 2007, Vergassola et al. published a paper about *infotaxis* [30], an odor source localization algorithm based on probability and information theory. This was the first paper to apply probabilistic search to odor sources. Infotaxis is a rigorous application of Bayes inference, whereby a robot keeps a probability distribution (belief) for the location of the source. Based on a plume model, this probability distribution is updated each time the robot makes a new odor concentration measurement. Using this probability distribution, the robot decides upon its best next move based on a minimum entropy criterion. In other words, the expected entropy decrease of the probability distribution is calculated for each possible move, and the “most promising” move is chosen. The paper shows simulation results of a robot (modeled as a point) moving in a lattice-type environment.

The underlying concepts of infotaxis — probabilistic fusion of information, and probabilistic decision taking — was known for a long time in mobile robotics [82] [83], and applied to a number of tasks, including self-localization, SLAM and, more rarely, target localization (search) [84]. Hence, infotaxis can be seen as one particular algorithm within the framework of probabilistic search.

This chapter discusses odor source localization algorithms within this probabilistic search framework, and is structured as follows: we start by introducing a general mathematical description of probabilistic odor source localization. Then, we report on results carried out with a point simulator to compare the performance between single-robot and multi-robot systems. In Section 4.3, we demonstrate on the example of a simplified train station scenario how probabilistic odor source localization algorithms could be used in more complex environments, and applied to static nodes and mobile robots at the same time. In Section 4.4, we describe a lightweight distributed implementation of the same probabilistic model, and discuss real-robot as well as simulation results obtained with this algorithm.

4.1 General Model

4.1.1 The Basic Model

With probabilistic odor source localization, the position of the source is modeled as a probability distribution¹ S . In the case of a 3D environment with one point source of unknown intensity s_r , this distribution can be written as

$$P(S = (s_x, s_y, s_z, s_r)) \quad (4.1)$$

We call the tuple (s_x, s_y, s_z, s_r) the *state* of the source.

In addition, we need a plume propagation model. Such a model tells us the probability of observing a concentration c at a location (x, y, z) , given that the source is at (s_x, s_y, s_z) and has an

¹One could argue that this distribution is an estimate of the source location (a point in space) and should therefore be written \hat{S} . We stick to the notation S because we see it as a belief distribution for the source location. S is thereby the real belief (given the observations), and not an estimate of it.

intensity s_r . Formally, this can be written as

$$P(O = (x, y, z, c) | S = (s_x, s_y, s_z, s_r)) = f(s_x, s_y, s_z, s_r, x, y, z, c) \quad (4.2)$$

With this, we assume that the orientation of the robot does not have any influence on the measurement c . If it does, these parameters can of course be added to the model.

The algorithm then consists of an update rule which updates S whenever a new observation is made, and a decision rule which tells the robot where to go next.

4.1.1.1 Update

The update step consists of filtering a new observation $O_k = (x_k, y_k, z_k, c_k)$ into the current source state probability, S_{k-1} , using recursive Bayesian estimation. In the general case

$$P(S_k | O_k, \dots, O_0) = \alpha P(O_k | S_k) P(S_k | O_{k-1}, \dots, O_0) \quad (4.3)$$

$$= \alpha P(O_k | S_k) \int P(S_k | S_{k-1}) P(S_{k-1} | O_{k-1}, \dots, O_0) dS_{k-1} \quad (4.4)$$

Note that α is just a normalization factor. The expression $P(O_k | S_k)$ stands for the plume model, while $P(S_k | S_{k-1})$ is a model for the motion of the source, or its change in intensity. We will briefly discuss moving sources in Section 4.1.2.7, but all our experiments were carried out with a static source, for which the equation simplifies as follows:

$$P(S_k | O_k, \dots, O_0) = \alpha P(O_k | S_k) P(S_{k-1} | O_{k-1}, \dots, O_0) \quad (4.5)$$

Within the recursive Bayesian estimation terminology, a recursion always consists of a *prediction* and an *update* step. For static sources, the prediction step simply falls away resp. becomes trivial, and only the update step is left.

4.1.1.2 Decision

In the decision step, the robot calculates the expected information gain for each potential next move, or, more precisely, each potential next observation.

The amount of information in the probability distribution S can be quantified by its information entropy

$$e(S) = - \int P(S = (s_x, s_y, s_z, s_r)) \cdot \ln(P(S = (s_x, s_y, s_z, s_r))) d(s_x, s_y, s_z, s_r) \quad (4.6)$$

Initially, when we have little clue about the location of the source, $P(S)$ is almost uniform and this entropy therefore large. The more information we obtain through our observations, the more the probability mass concentrates on a specific region, and thereby reduces $e(S)$, as depicted in Figure 4.1. If we know the exact state of the source, the probability distribution reduces to a Dirac function, with an entropy of $e(S) = 0$.

Assume now that we are at time step k , and the robot can execute one action out of a discrete set of potential actions,

$$A_{k,1}, A_{k,2}, \dots \quad (4.7)$$

Such actions may have been proposed by a path planner, for example. If the robot chooses to perform action $A_{k,1}$, this yields a distribution of potential observations the robot will make during or after this action. Action $A_{k,1}$ might take the robot to some location (x_0, y_0, z_0) , for instance, where it will take a concentration measurement c_0 (e. g., a high odor concentration) and thereby

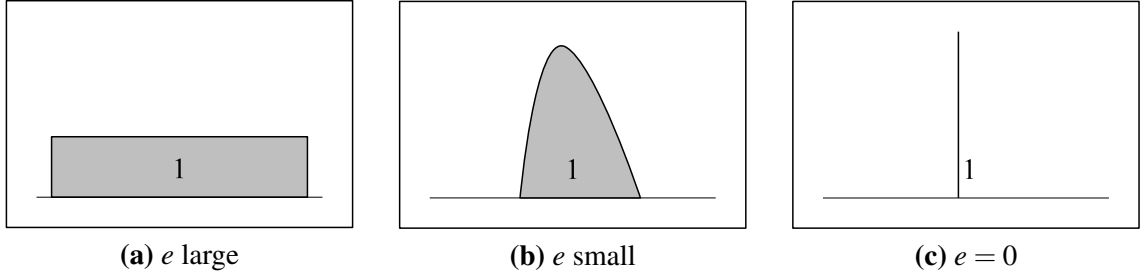


Figure 4.1: Graphical representation of the entropy of a probability distribution. (a) A uniform distribution (over a finite interval) has a high entropy. (b) The more the probability mass is concentrated in a small area, the lower the entropy. (c) If the distribution is a Dirac function, its entropy is 0.

produce the observation $O_{k+1,A_{k,1},0} = (x_0, y_0, z_0, c_0)$. At the same location, a robot may also measure a concentration c_1 (e. g., a low odor concentration), yielding a second potential observation $O_{k+1,A_{k,1},1} = (x_0, y_0, z_0, c_1)$. When considering uncertainty in the movement of the robot, action $A_{k,1}$ may even take the robot to different locations. For an illustration of this, check out Figure 4.2.

When taking action $A_{k,i}$, we will therefore make one of a whole set of observations,

$$O_{k+1,A_{k,i},0}, O_{k+1,A_{k,i},1}, \dots \quad (4.8)$$

Each of these observations occurs with some probability. Regarding the location of the robot, the motion model of the robot should be able to provide us with a probability value for each potential location. This is highly application- and platform-dependent, and in many cases using one deterministic target location per action might be a good approximation.

The probability of measuring a certain concentration c at a location (x, y, z) can be estimated using the plume model and the current belief of the source state:

$$P(C = c|x, y, z) = \int P(C = c|S, x, y, z)P(S)dS \quad (4.9)$$

We now have a discrete set of potential actions, and for each action a discrete set of potential observations, each of which is assigned a probability of actually happening if the robot decides to take the corresponding action ($P(O|A)$). To calculate the expected entropy² of an action $A_{k,i}$, the decision algorithm calculates the entropy for each potential observation of that action, and weights the results with the probability of these observations. Formally,

$$e_A(A_{k,i}) = \sum_n e_O(O_{k+1,A_{k,i},n})P(O_{k+1,A_{k,i},n}|A_{k,i}) \quad (4.10)$$

where $e_A(A)$ stands for the expected entropy if action A is chosen, and $e_O(O)$ the expected entropy if O is observed. Calculating the latter is pretty straightforward: we first apply an update step for this potential observation:

$$P(S_{k+1,A_{k,i},n}|O_{k+1,A_{k,i},n}, O_k, \dots, O_0) = \alpha P(O_{k+1,A_{k,i},n}|S_{k+1,A_{k,i},n})P(S_k|O_k, \dots, O_0) \quad (4.11)$$

which is essentially a prediction for the next step, under the condition that the corresponding observation is made. Then, we calculate the entropy of the resulting belief of the source state. Formally,

$$e_O(O_{k+1,A_{k,i},n}) = e(S_{k+1,A_{k,i},n}|O_{k+1,A_{k,i},n}, O_k, \dots, O_0) \quad (4.12)$$

²The expected entropy decrease is simply the difference between the current entropy and the expected new entropy. Maximizing the expected entropy decrease is therefore equivalent to minimizing the expected entropy.

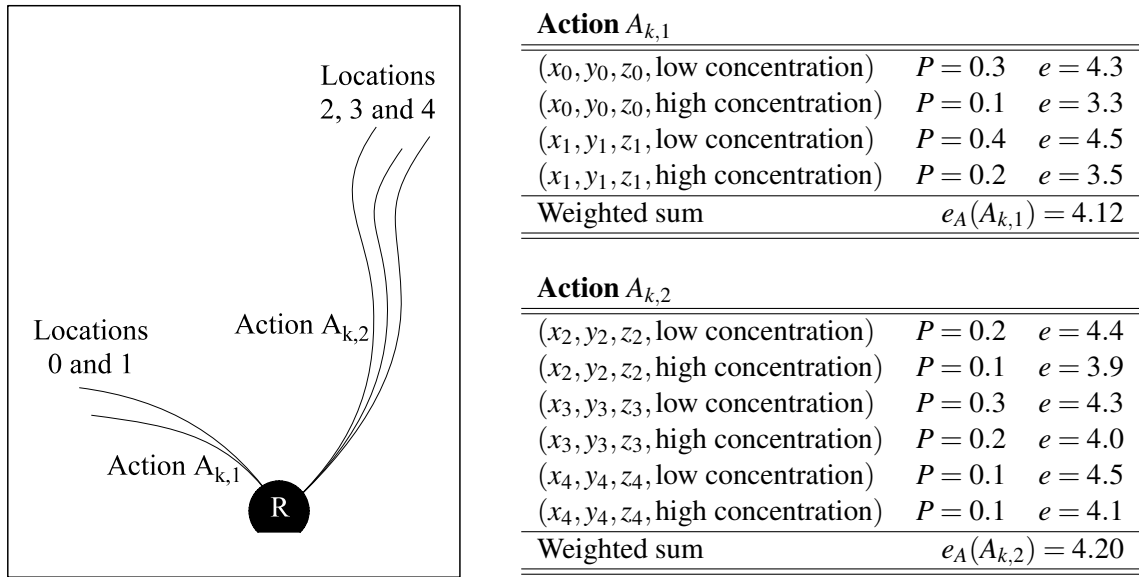


Figure 4.2: Decision step example with two potential actions with 4 resp. 6 potential observations. For the sake of simplicity, the sensor only distinguishes between low and high odor concentration. The probability values are used as weights for the weighted sum. In this example, action $A_{k,1}$ would be chosen, as it yields a lower expected entropy value.

An illustration of this procedure is depicted in Figure 4.2. All that remains to be done is to select the action yielding the minimum expected entropy, and execute it.

4.1.2 Extensions to the Model

4.1.2.1 Initialization and Human Input

When robots are deployed on a site to track down an odor source, they need to be given an initial guess of the source location, which is used as probabilistic prior the first time the update step is applied, i. e., S_0 . Usually, there is some information around already. For example, people may know in which building, room, or field the source is located. In this case, the prior can be initialized to be uniform over the search area.

In some cases, people have more knowledge. They may have a prior belief about the location or intensity of the source based on experience or other evidence, for instance. All such information can be integrated into the prior, and will help the robot to take better decisions especially at the beginning, when it has not got many observations on its own. When looking for a source in a building with several rooms, for example, the algorithm would automatically give preference to rooms with a high prior value, as the source is more likely to be located there.

Human input can even be integrated while the robot is searching. To do that, such input is integrated into the model as an observation. Unless this observation is a concentration measurement, its probabilistic description will be different from the observations the robot makes. Nevertheless, the update rule (equation (4.5)) is the same.

4.1.2.2 Path Planning and Obstacles

Combining this model with a path planner is straightforward. A path planner algorithm [83] will propose feasible trajectories that the robot can take from its current location, usually based on

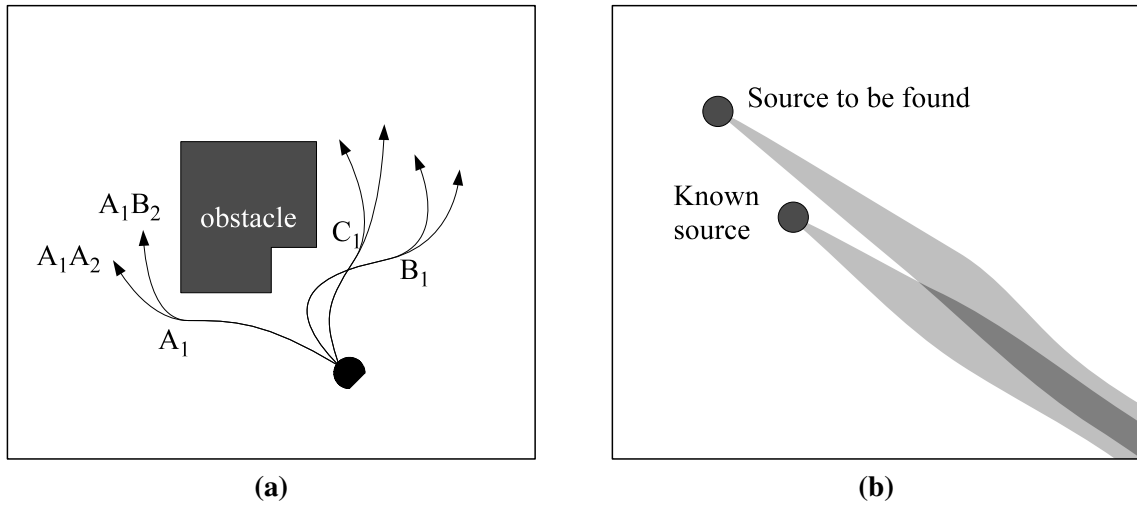


Figure 4.3: (a) Example of a path planner suggesting 6 different routes to navigate around an obstacle. The probabilistic odor source localization algorithm will analyze each path and choose the one which minimizes the expected entropy, or the best cost-to-entropy-reduction ratio. (b) Example of a scenario with one additional known source. This source is contaminating the area and must therefore be taken into account when localizing the unknown source.

a map, or information about the environment collected with other sensors. The decision step algorithm will simply evaluate all these trajectories and pick the best. If the path planner assigns costs to these paths, it can be compared with the entropy to select the best trade-off.

Dealing with obstacles, difficult terrain or strategic considerations are therefore nicely delegated to the path planner, or algorithms providing input to the path planner. Indeed, these are all issues related to standard robot navigation, and not odor source localization in particular. Hence, probabilistic odor source localization can be integrated very nicely into existing mobile robotic systems.

4.1.2.3 Multi-Step-Ahead Calculation

A good path planner provides a tree of feasible trajectories that extend to more than one step (e. g., iteration of the control loop) into the future. In the example in Figure 4.3, the path planner came up with 6 different trajectories, whereby two trajectories each share a common first part (A_1 , B_1 , and C_1).

Instead of analyzing the next step only, the decision step algorithm can evaluate the whole tree. At each step (depth of the tree), the algorithm must thereby iterate over all possible actions and observations. As shown in Figure 4.4, the evaluation tree has one observation layer for each action layer. The entropy is only calculated at tree leaves, and backtracked to the current position as follows:

- ▷ At tree leaves (which are always observations), the entropy is calculated using equation (4.6).
- ▷ Entropy values coming from different observations are averaged according to the probability with which observations are made. Indeed, the robot cannot influence the observation he will make at a particular position, and therefore calculates the expected entropy at this node.
- ▷ For entropy values coming from different actions, only the minimum entropy value is kept. As the robot can deliberately choose the action, it will select the one which minimizes the

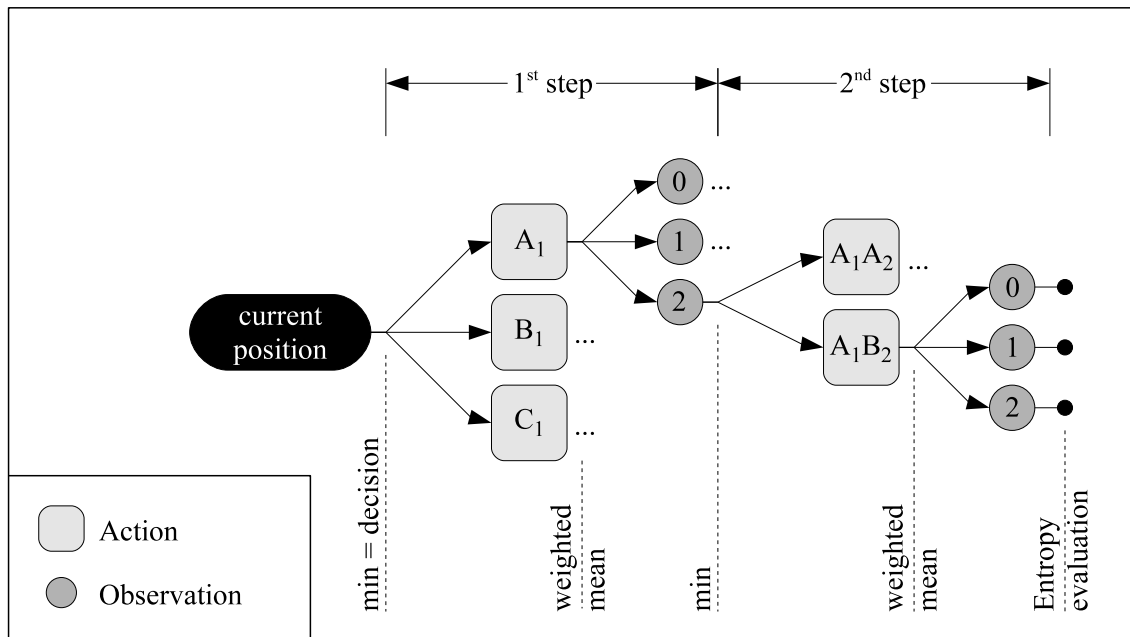


Figure 4.4: Evaluation tree when calculating multiple steps ahead of the current position. Note that branches marked with “...” are omitted for clarity — the algorithm obviously has to evaluate all branches.

entropy. The minimization at the root (current position) corresponds to the decision.

Note that there is a direct relationship with the well-known *min-max* algorithm for taking decisions in strategic games. The path planner tree defines the set of actions a player can take. The “opponent” is the observation, and since observations are not evil, the weighted average is calculated instead of the maximum. The entropy function at tree leaves corresponds to the evaluation function in a strategic game.

Just as with strategic games, the evaluation complexity grows exponentially with the number of steps that are calculated ahead of the current position. It is therefore impractical to calculate more than a few steps ahead.

4.1.2.4 Multiple Robots, or Multiple Sensors

To integrate the observations of multiple sensors (whether they are on a robot or not), it is enough to apply the update step once for each observation. Sensors can thereby have different sampling intervals (regular or irregular), different sensitivity ranges and even different noise distributions.

Things are a bit more complicated with the decision step, when multiple robots are supposed to take decisions in an optimal fashion. Since robots are moving in the same space, the observations they will make are *dependent*, and their actions therefore as well. To illustrate this, have a look at the example in Figure 4.5. Note that this example is constructed for the purpose of showing the dependence, and not very realistic. Both robots shared all their observations and are therefore taking their decisions based on the same source state probability distribution. For robot R_1 , the best decision clearly is to explore the big area. The same holds for robot R_2 , if it takes its decision independently. Since robot R_1 will explore this area already, the information gain of a second observation in this area will be minimal, however. If, on the other hand, both robots take their decisions together (exploiting the dependence of their actions), robot R_1 would explore the big

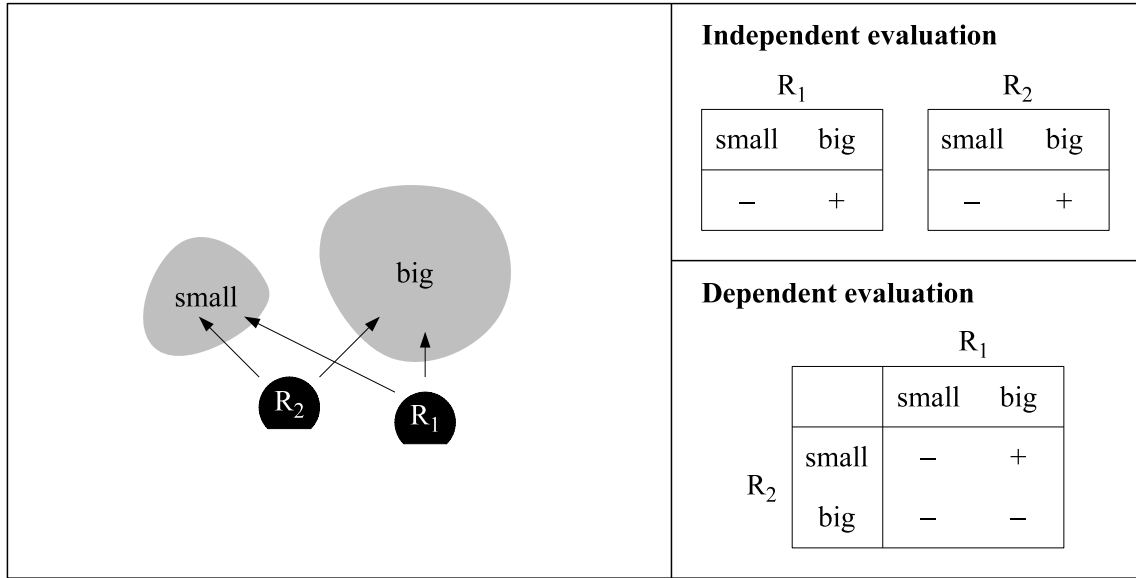


Figure 4.5: *Dependent vs. independent evaluation with multiple robots.* “+” stands for a favorable decision for the robot (independent evaluation) or the team (dependent evaluation), while “–” stands for an unfavorable decision. Note that the example shown here is constructed for the purpose of showing the difference, and not representative for the majority of the decisions robots take.

area, while robot R_2 would explore the small area, and thereby minimize the entropy after taking both observations into account.

Evaluating decisions in a completely dependent fashion grows exponentially with the number of robots (or sensors) and is therefore impractical to use with more than a few robots. Note that even if only one robot is moving, while all other sensors are static, the number of evaluations still grows exponentially as all possible combinations of observations by all sensors have to be considered.

Fortunately, there are few situations like the one shown in Figure 4.5, where dependence really makes a decisive difference (see Section 4.3.8 below). Reasons for this are two-fold: First, taking several measurements close by can still yield a substantial information gain if measurements are noisy. Second, since robots share their past observations, their decision is anyway “dependent” with respect to everything up to the present. Only the projected future observations are assumed to be independent. Since it is computationally not possible to calculate more than a few steps ahead into the future, the number of observations which are approximated in this way is relatively small.

Hence, for most practical purposes, taking independent decisions — in which case the complexity grows linearly with the number of robots (or sensors) — is a sufficiently good approximation.

4.1.2.5 Multiple Sources

Multiple sources (of the same type of odor) are straightforward to model within the above model. For N sources, the source state probability distribution simply becomes:

$$P(S = (s_{x,1}, s_{y,1}, s_{z,1}, s_{r,1}, s_{x,2}, s_{y,2}, s_{z,2}, s_{r,2}, \dots, s_{x,N}, s_{y,N}, s_{z,N}, s_{r,N})) \quad (4.13)$$

However, as each source adds 4 dimensions to the distribution, this very quickly becomes computationally intractable. Note that all dimensions are dependent, as the state of a source heavily influences the probable states of all other sources, given the observations made.

The limit case of this actually has a dual formulation. Imagine each physical location hosts a source, and each of these sources can either be switched on (at different intensity levels) or off. The number of possibilities grows exponentially with the number of physical locations, rendering the problem intractable.

This model is therefore only applicable if the source states space is very small. One may, for instance, be looking for multiple sources in a hospital. Each of the patient rooms on a floor potentially holds a source, but it is not of interest where in the room the source is located. Each source would therefore only be in one of two potential states (on, or off), making the problem tractable up to about 20–30 sources.

In many practical multi-source scenarios, however, single-source models should work reasonably. If two sources are very close to each other, they could as well be treated as one source. If the sources are far away from each other such that their plumes do not mix, a robot will track the plume of only one of the two sources. In addition, if the plume model is very general, the robot will likely find one of the sources and treat the plume of the other source as “noise”. We do not explore the multi-source problem any further in this thesis, and leave these assertions as hypotheses to be verified in future work.

4.1.2.6 Contaminating Sources

A special form of the multi-source problem occurs when the state of all but one of the sources is known. In this case the above formulation (see equation (4.13)) can be used, but

$$s_{x,2}, s_{y,2}, s_{z,2}, s_{r,2}, \dots, s_{x,N}, s_{y,N}, s_{z,N}, s_{r,N} \quad (4.14)$$

will have fixed and known values, i. e., their distribution collapses into a single deterministic value. The complexity of the model then reduces to that of a single-source model.

The contaminating source problem is actually very interesting from an applications point of view. Imagine, for instance, the problem of finding the source of some “bad smell” in the neighborhood. Compost bins or similar sources would obviously have to be considered as contaminating sources, but their locations and intensities are known (or can be measured).

Similarly, when working with non-selective odor sensors, sources of other types of smell can be inserted into the model as contaminating sources. As many odor sensors suffer from poor selectivity, this feature is of particular interest.

While the model is straightforward to apply, there are a few practical considerations to be made. Measuring the exact state of a source might be difficult or impractical. Furthermore, the source to be found might be “shadowed” in the plume of one of the known sources, rendering the problem particularly difficult. We do not explore this problem further in this thesis, and leave its exploration for future work.

4.1.2.7 Moving Sources

In Section 4.1.1.1, we briefly mentioned the extension of this model to moving sources. If a probabilistic description of the source movement is known, integrating it may seem fairly straightforward.

There is a small issue with this approach, however. Plume propagates at fairly low speed, and the plume concentration measured by the robot causally depends on the state of the source

somewhen in the past (and not of the current source state). The model presented here does not take this into account, and might therefore fail with fast moving sources.

As long as there is only one robot taking one measurement, this is not a big issue. As the robot gets closer to the source, the discrepancy between the estimated past state and the present state of the source shrinks, and by the time the robot reaches the source, there is no discrepancy any more.

For multiple robots, this approach can be problematic, as different robots will make observations corresponding to the state of the source at different times. To properly take this into account, the whole source state trajectory would have to be modeled as a random variable, which can add significant computational complexity.

4.1.2.8 Changing Wind

A similar problem could arise if the plume propagation model changes over time, for instance as a result of changing wind. While recalculating the plume propagation to account for the wind profile is feasible, one would have to apply this to the plume that has been released by the source in the past. This, again, can add significant computational complexity.

If a place is susceptible to different wind conditions, these different plume propagation models can of course be combined to one static model. Such a model is obviously less specific, as it ignores knowledge about the current wind conditions.

4.1.2.9 Robot Localization Uncertainty and SLAM

In many practical applications, the robot does not know its own position precisely, but has to estimate it from features in the surroundings. This is the case when using SLAM [83], for instance.

Localization uncertainty in the observations (i. e., odor concentration measurements) can be accounted for by extending the update step to distributions of observations, $O_{k,i} \in O_k$. All possible observations are thereby filtered into the source state probability with their corresponding probability weight. The update step then becomes

$$P(S_k | O_k, \dots, O_0) = \alpha \sum_i P(O_{k,i}) P(O_{k,i} | S_k) P(S_{k-1} | O_{k-1}, \dots, O_0) \quad (4.15)$$

When using probabilistic SLAM for mapping and localization, the probabilistic odor source localization model presented here could even be completely combined with the probabilistic model used for SLAM. The combined model (simultaneous localization, mapping and odor source localization) would allow the robot to build a map including the odor source, and localize itself on there. It is clear that such a model would be complex, but — at least from a theoretical perspective — certainly worth investigating further. A similar approach applied to gas distribution mapping was shown by Lilienthal et al. [85]. In this PhD thesis, we do not dig any deeper into this subject, and leave this research axis for future work.

4.1.3 Optimality

A natural question to ask ourselves at this point is whether this algorithm is optimal. Within information theory, this approach is indeed optimal: we are taking into account all information (i. e., all observations we make), and we base our decision upon the highest information gain (highest entropy decrease). This optimality is greedy, however, as the robot calculates the information gain on a step-by-step basis only. To make the algorithm globally optimal, we would have to calculate the information gain of each potential complete trajectory to the source, and not just the next step. This, obviously, is computationally not feasible. (In the framework of strategic games, this

corresponds to completely solving a game, which is usually possible for relatively short or simple games only.)

In addition, the decisions taken by the algorithm are greedy optimal with respect to the plume propagation model only. If this model is inaccurate, the algorithm will forcibly come up with a bad source state belief and take bad decisions.

Likewise, if the plume propagation model is very unspecific (even if accurate), the algorithm might be inferior to algorithms using (explicitly or implicitly) more specific models. To illustrate this, think of a plume propagation model yielding a uniform plume distribution over the whole arena. What this says is that a plume filament released by the source could be measured anywhere in the arena, and that there is no evidence to claim it is more probable to be measured in one place rather than another (not even close to the source). Such a model is accurate, but unspecific. In this particular case, it is even that unspecific that it is impossible to find the source based only on the observations.

Practical implementations usually add a few more limitations related to parametrization or discretization of the distributions. All observations made, as well as the source state belief have to be approximated by either parametrizing or discretizing them. This also holds for the target points (or paths) that are evaluated in the decision step.

4.2 Map-Centric Implementation

We first implemented the model described in Section 4.1 as a 2D point simulation with a simple plume propagation model. Robots are thereby modeled as points moving in a finite 2D arena with differential-drive kinematic constraints. The position of all robots and all obstacles is assumed to be precisely known in an absolute reference frame. The implementation is therefore map-centric, and inherently centralized³.

The simulation scenario, depicted in Figure 4.6, resembles the setup in the wind tunnel. The arena is 18 x 4 m large, and the distance between the source and the robot starting position is 14 m. The robot is not allowed to leave this arena, and its boundaries are modeled as walls. The source intensity is assumed to be known, and the state of the source can therefore be represented with the

³The program could of course be run directly on a robot, or ported to a distributed system using standard computer science techniques. But the underlying concept is that of a centralized algorithm where all pieces of information come together at one single point.

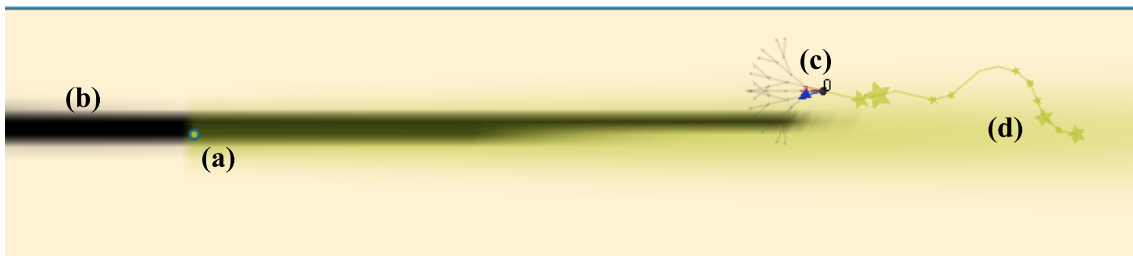


Figure 4.6: Scenario of the probabilistic odor source localization experiments with absolute localization. (a) Real location of the odor source, and the plume (average concentrations) generated by it. (b) Belief of the source location (darker = more probable). (c) Robot with the path planner tree and the selected path indicated by the big arrow. (d) Trajectory with observations. The bigger the stars, the higher the concentration was at that point.

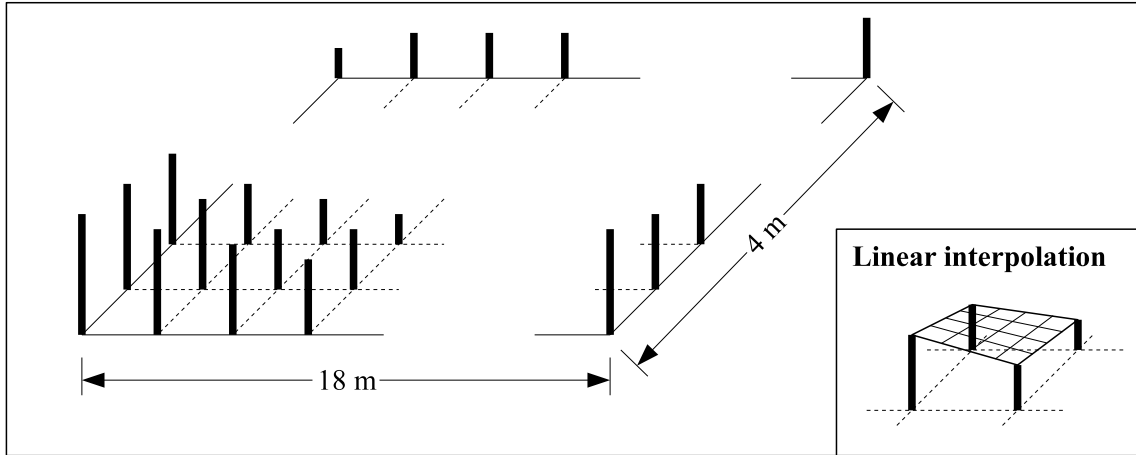


Figure 4.7: Implementation of the 2D probability distribution to model the believe of the odor source.

following 2D probability distribution:

$$P(S = (s_x, s_y)) \quad (4.16)$$

At the beginning of each experimental run, this probability distribution is uniform over the whole arena, i. e.,

$$P(S = (s_x, s_y)) = \frac{1}{18 \cdot 4} \quad (4.17)$$

There are several ways of representing probability distributions on a computer [82]. We chose a grid-based approach (static particles on a regular lattice) with particles spaced by 20 cm. In between particles, the probability value is linearly interpolated using the 4 neighboring particles, as shown in Figure 4.7. This was found to be a good trade-off between accuracy and computational load.

The plume propagation model for an observation at (x, y) and a source at position $(0, 0)$ is

$$r_y = \exp\left(-2\left(\frac{y}{A_w + A_d x}\right)^2\right) \quad (4.18)$$

$$r_x = \begin{cases} 1 - A_c x^2 & \text{if } 0 \leq x < A_c^{-\frac{1}{2}} \\ 0 & \text{otherwise} \end{cases} \quad (4.19)$$

$$C \sim \text{Poisson}(\lambda = A_s r_x r_y) \quad (4.20)$$

where $A_w = 0.5$ is the plume width parameter, $A_d = 0.03$ the diffusion parameter, $A_s = 3.0$ the scaling parameter, and $A_c = 0.002$ the decay parameter. Odor concentration is measured in *hits*, an approach which was also used in the original infotaxis paper [30]. Each measurement corresponds to a discrete number of hits, which is assumed to be a sample of the Poisson distribution C .

Note that this model does not have a direct physical interpretation, but produces a plume map similar to the one measured in the wind tunnel (see Section 2.1.7). The underlying wind flow in this model is a laminar flow from left to right. The wind speed, however, is not explicitly defined and irrelevant in this scenario.

Each robot is endowed with a basic multi-step-ahead path planner. It builds a trajectory tree from the robots' current position and checks for each trajectory whether it is feasible or not. The

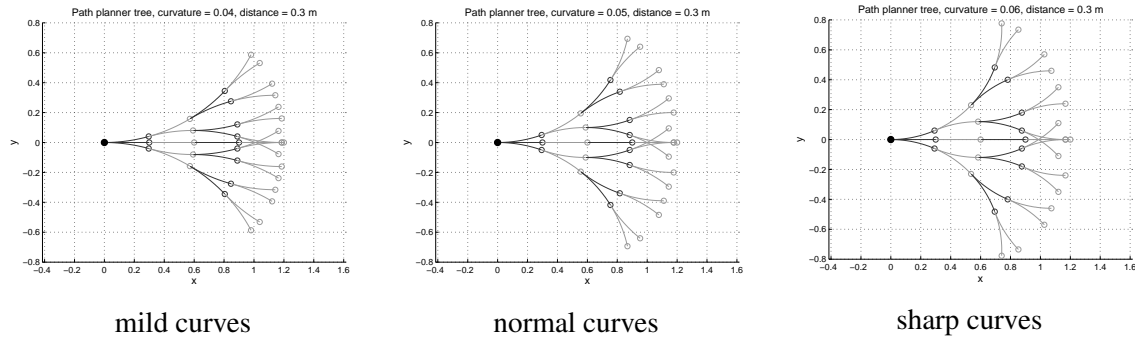


Figure 4.8: Path planner trees used for the experiments. The plots show trees with a target distance of 0.25 m. The gray shading of layers 2 and 4 only serves visualization purposes. Note that our path planner only considers forward trajectories, as backward trajectories are not relevant in these experiments and would only increase the computation time.

only reason for a trajectory to be erased is if the robot would bump into an obstacle otherwise. The path planner trees used in these experiments are depicted in Figure 4.8. Unless otherwise stated, the tree labeled “normal curves” was employed with different target distances. The target distance has a dual influence on the experiments: As a robot only makes a single observation per step, it will make more observations when shorter target distances are used. Hence, it has more information to fuse into the source probability distribution. On the other hand, shorter target distances will translate into slower robot speeds, or higher perception-to-action loop speeds, on a real system.

A substantial effort in this implementation was put in the decision step algorithm, which handles both multiple robots and multi-step-ahead evaluation at the same time.

For multi-robot evaluation, robots can be combined to evaluation groups. Within an evaluation group, all robots are evaluated in a dependent fashion. Evaluation groups themselves are evaluated independently from each other. (Robots in any case share all their observations, no matter whether they are in the same evaluation group or not.)

Within an evaluation group (set of robots), all combinations of trajectories are evaluated up to a certain depth (evaluation depth) of the path planner tree. Since path planning is computationally much cheaper than trajectory evaluation, the evaluation depth is limited to 1 – 3 steps in the experiments. After each evaluation, all robots move to the first-level target which was found to be best for themselves (independent evaluation) or the group (dependent evaluation).

Note that the simulation operates in synchronous steps and implicitly assumes that all robots reach their target positions at the same time (or wait there until all robots have reached their target positions). Low-level dynamics of the robots, its sensors and its actuators are not simulated. Hence, the simulation program does not keep track of any physical notion of time, but only of simulation steps which would ultimately translate into time if the algorithm was applied on a real platform. Nevertheless, calculating the success rate and distance overhead is straightforward, as they only depend on the trajectory.

4.2.1 Results

We run experiments for the 15 configurations listed in Table 4.1. For each configuration, we carried out 50 runs each for 11 different target distances, totaling to 8250 runs. A run was called successful as soon as the robot reached proximity (50 cm) of the source. The only failure possibility was when the path planner was unable to come up with any viable path, which could happen in corners. The success rate is therefore not indicative for the odor source localization performance.



Figure 4.9: Results obtained with the point simulation assuming absolute localization. **A, B:** Influence of multi-step-ahead evaluation with 1 robot resp. 3 robots. **C, D, E:** Dependent vs. independent evaluation with 2 robots with 1-step-ahead evaluation, 2 robots with 3-step-ahead evaluation, and 3 robots with 1-step-ahead evaluation. **F:** Effect of the curvature parameter.

Table 4.1: Configurations in which the map-based implementation of the probabilistic model was tested.

Algorithm	Robots	Steps ahead	Evaluation	Path planner	
A	Map-centric probabilistic OSL	1	–	normal curves	
		2			
		3			
B	Map-centric probabilistic OSL	1	independent	normal curves	
		2			
		3			
C	Map-centric probabilistic OSL	2	1	independent dependent	normal curves
D	Map-centric probabilistic OSL	2	2	independent dependent	normal curves
E	Map-centric probabilistic OSL	3	1	independent dependent	normal curves
F	Map-centric probabilistic OSL	1	2	–	mild curves
					normal curves
					sharp curves

The results are shown in Figure 4.9. Plots **A** and **B** show the impact of multi-step-ahead evaluation. Evaluating 2 steps ahead yields a significantly better performance than with just 1 step ahead. The third step ahead, however, does not increase the performance any more. Using multiple robots also increases the performance. While the distance overhead for 1 robot with one-step-ahead evaluation is about 1.2, it drops to about 1.05 for three robots with three-step-ahead evaluation.

Plots **C**, **D** and **E** compare dependent vs. independent evaluation for different configurations. All plots indicate that there is no significant advantage of dependent over independent evaluation for multiple robots. In theory, dependent evaluation should be at least as good as independent evaluation. This difference could be too small to be visible here, however, since most of the time, dependent and independent evaluation yield exactly the same result. With both dependent and independent evaluation, the robots share all their observations up to the current time step, and therefore have a common basis for their decision. With dependent evaluation, the robots also share the fictive observations assumed during the evaluation process, whereas with independent evaluation, these fictive observations are not shared. Since there are only few fictive observations as compared to real observations, they only have a minor impact.

The target curvature does not have a significant influence in the range tested here, and even different target distances only have a minor impact. The latter has to be attributed to the idealistic and static environment. Since the odor hits are sampled from the plume model itself, a few observations are enough to get a good belief of the source location. Whether the robot makes 25 observations (with a target distance of 70 cm) or 85 observations (with a target distance of 20 cm) does not make a big difference any more.

4.3 Train Station Scenario

In the previous section, we discussed a map-centric implementation of the probabilistic model. While this implementation is able to deal with walls and obstacles, the environment we used to

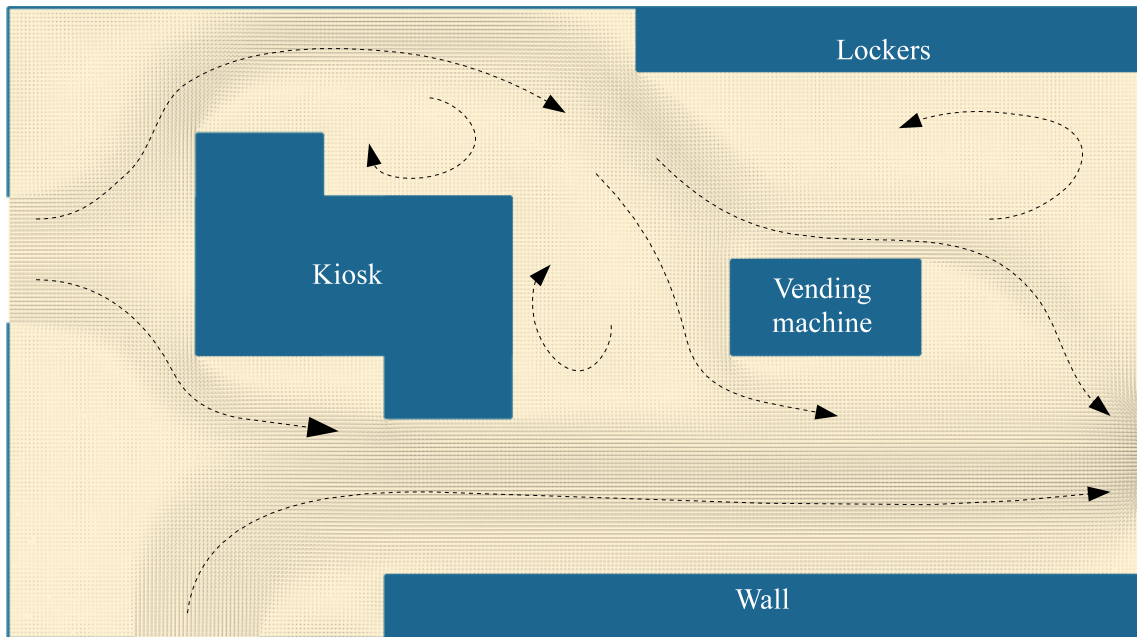


Figure 4.10: *The train station scenario used in Section 4.3. The room is 18x10m big, and represents a simplified version of room as it might exist in a train station. The gray shading stands for the wind speed, and the arrows indicate the main wind flows. Both doors on the left are wind inlets, and the door on the right is the only wind outlet. Wind speeds are in the order of 0–1 m/s.*

carry out simulation experiments was very simple, and the plume propagation model as well.

To demonstrate the potential of the probabilistic model for real applications, we therefore extended the previous implementation to use a data-driven plume propagation model. The scenario used in this section is the room depicted in Figure 4.10, a simplified 2D version of a room such as it might exist in a train station building. It has three doors: two through which wind flows into the room, and one (on the right), through which the wind flows out. A kiosk and a vending machine are in the middle of the room, and lockers can be found along one of the walls.

An odor source is expected somewhere in this room, but its location is unknown. Such an odor source could be a maliciously placed bomb (releasing minute amounts of an explosive substance), for instance, or a drug dealer (releasing minute amounts of the drug) waiting for a customer.

Note that this is a toy example neglecting many issues. A real application in a train station would have to deal with a 3D room, changing wind conditions, people walking around, material lying on the floor, uncertainties of the map, and so on. However, for the purpose of showing the concept and demonstrating the possibilities, the present level of details is fully sufficient.

4.3.1 Modeling the Plume Propagation

The main difference as compared to the simulation shown in Section 4.2 is the plume propagation model. While the previous simulation used a generic model which was independent of the location of the source, the present model consists of precomputed plume propagation data for each potential source location. Data were computed in two steps:

1. First, the scenario was simulated in a CFD (computations fluid dynamics) program. For this simple setup, we used EasyCFD [25]. After all walls, wind inlets and wind outlets were

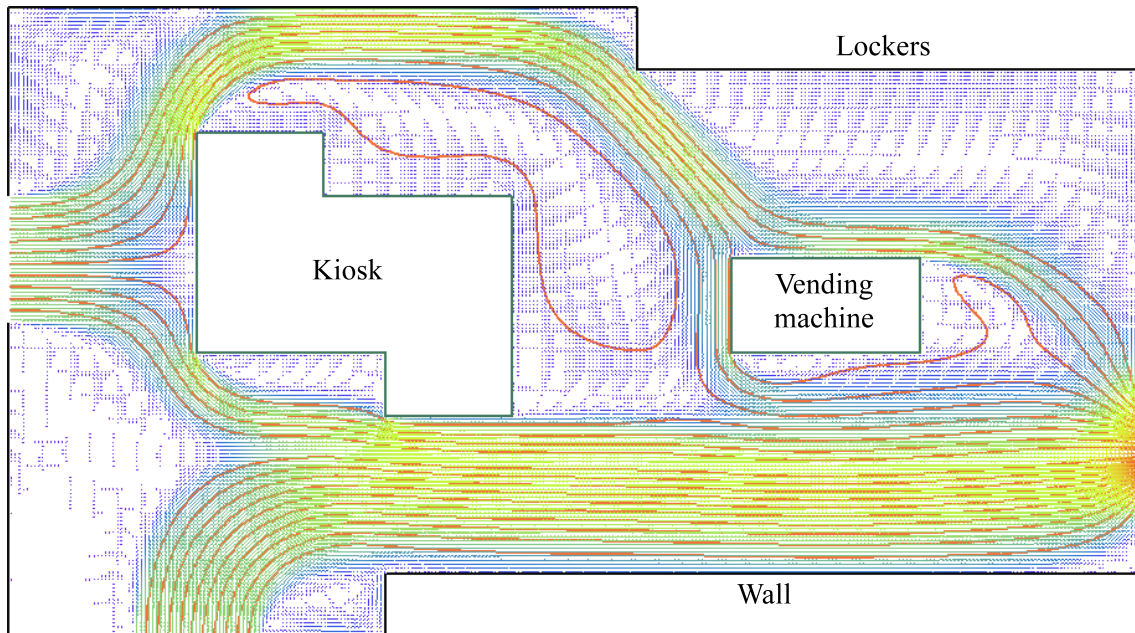


Figure 4.11: Result of the wind flow computation in EasyCFD [25].

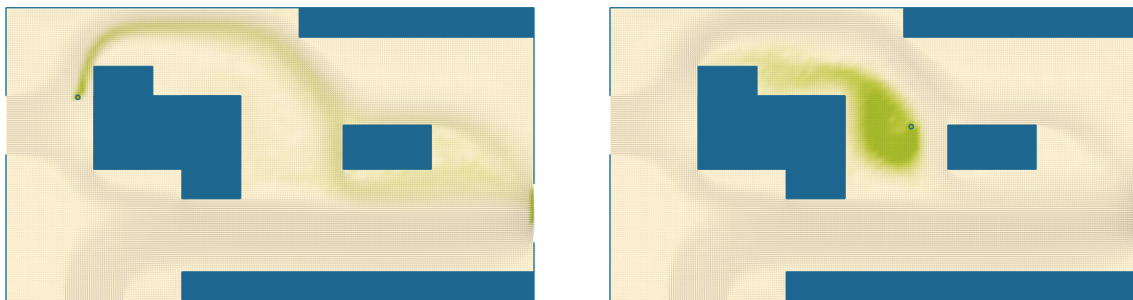


Figure 4.12: Plume propagation maps for two different source locations. The plume propagation model used for the train station scenario consists of 4500 such maps. The gray shading stands for the wind speed, while the olive shading is proportional to the odor concentration.

defined, the program calculated the wind flow through this room. A picture of the result is shown in Figure 4.11.

2. This wind data was then used in combination with Farrell's filament dispersion model [7] to create plume maps as those shown in Figure 4.12. Each such map was built by simulating 500 filaments. The map is a (scaled) 2D histogram of the filament trajectories with a 20 x 20 cm resolution. Maps were created for potential source locations on a regular 20 x 20 cm lattice over the whole area.

The so created data consist of 4500 individual plume maps stored as 176 MB of raw data (18 MB compressed). The plume model equation used in the previous simulation was replaced by a lookup in these plume maps with linear interpolation between data points. The rest of the implementation (update step algorithm, decision step algorithm, path planner, ...) was not modified.

4.3.2 Experiments

As this is an arbitrarily chosen scenario, we did not carry out any systematic experiments. Instead, we demonstrate possibilities and limitations of the algorithm through a set of representative runs. In all runs, robots are modeled as points, and inter-robot collision is neglected. Each robot is associated with a 5-step-ahead path planner which takes into account the kinematic constraints of the (differential-drive) robot, and prevents the robot from running into obstacles.

The odor sensor is sampling odor hits from the plume model data. Hence, the plume model is perfect with respect to the simulated odor propagation. By doing so, we neglect any plume propagation dynamics and implicitly assume that the source has been in the area for a duration that is sufficiently long for the odor to propagate through the whole room.

Note that the robots are not equipped with any wind direction sensor here, and wind is not simulated at all. Wind speed and direction actually only helps building a plume model. It is not the actual wind information, but the resulting plume model that odor source localization algorithms use. In this train station scenario, a complete and accurate plume model is provided to the robot, and therefore makes any information about the wind flow superfluous.

4.3.3 Plume Tracking with 1 and 2 Robots

Figure 4.13 shows 6 successful plume tracking runs with 1 or 2 robots, and up to 3-step-ahead target evaluation. In all runs, the robots are going upwind towards the source. With 1-step-ahead evaluation, the trajectory clearly exhibits counter-turning (zig-zagging), which is almost not present with 2- or 3-step-ahead evaluation.

The runs with 2 robots show very nicely how the robots explore the plume boundary, rather than staying in the center of the plume. Indeed, the plume boundary is where most information can be acquired. In the (expected) center of the plume, the robot expects to get a hit anyway, and a measurement is therefore superfluous there. In one of the runs, the robots explore both sides of the plume boundary, which is the optimal strategy. In one run, the robots mostly stay on the same side, which is a local optimum that is almost as good as the global optimum. Robots therefore like to stay on the same side once they have started that way.

4.3.4 Unreachable Source

In Figure 4.14, robots are tracking a source that turns out to be not reachable. For some external reason (e. g., debris lying around, delicate surface, dangerous area, . . .), robots are not allowed to enter the zone inside the circle, and it turns out that the source is inside this circle. If the source was placed maliciously, the attacker could actually have done that on purpose.

The path planner makes sure robots do not enter the forbidden area. But even though the robots are unable to physically reach the source, they come up with an accurate belief of the source position within the circle. Such information can be critical for human investigators when deciding upon further actions.

4.3.5 Non-Uniform Prior

Vice-versa, human investigators may have obtained information about the source location prior to launching the robots⁴. Figure 4.15 shows an example in which investigators believe that the source is located close to one of the doors. They therefore assign a 10 times higher probability

⁴Such information could also stem from other sensors in the environment or on the robots. For instance, camera images may have been processed to detect objects likely to be sources [26] [13].

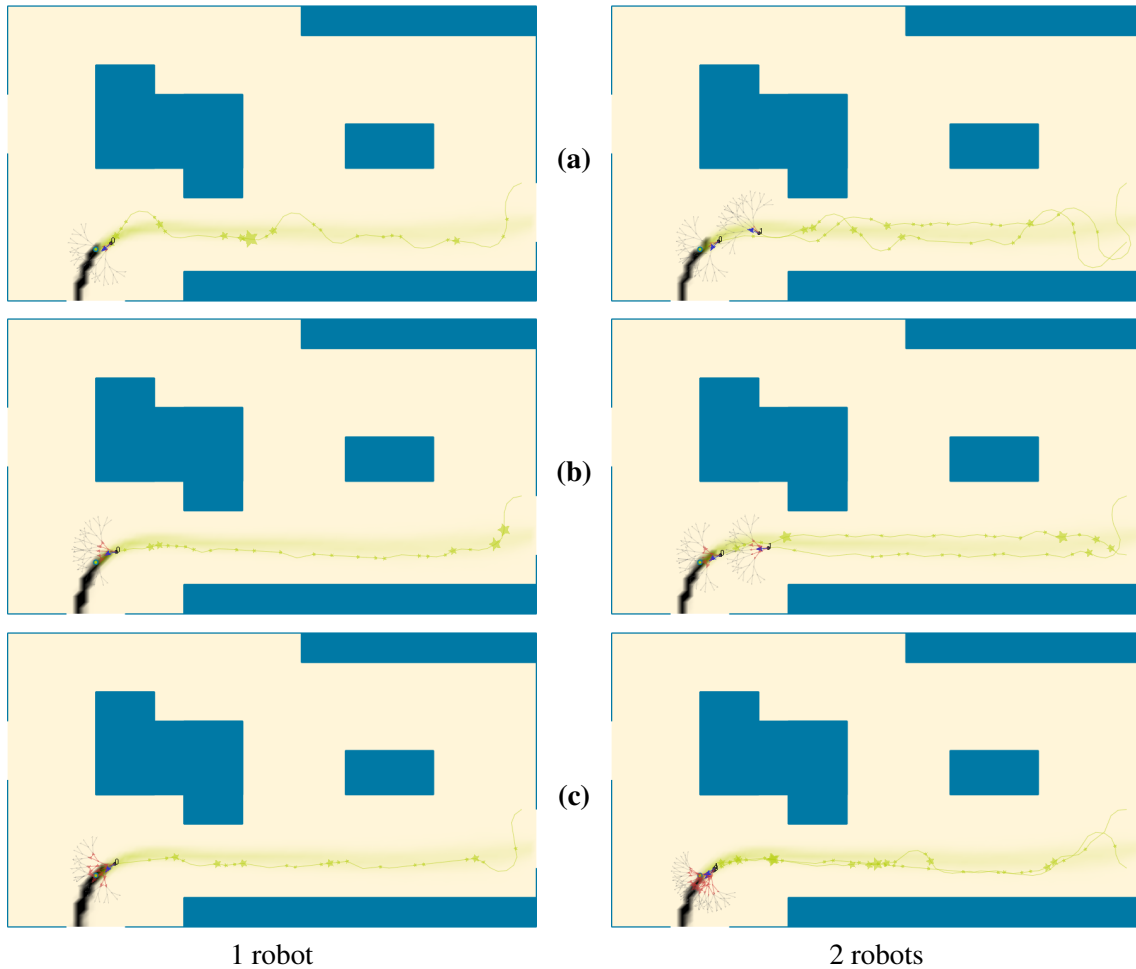


Figure 4.13: Plume tracking with 1 or 2 robots in the train station scenario. The pictures depict the situation when the robot first reached vicinity of the source (ideal source declaration). Note that the posterior usually still contains areas upwind of the actual source location, because the robot approached the source from the downwind area. (a) 1-step-ahead evaluation. (b) 2-step-ahead evaluation. (c) 3-step-ahead evaluation.

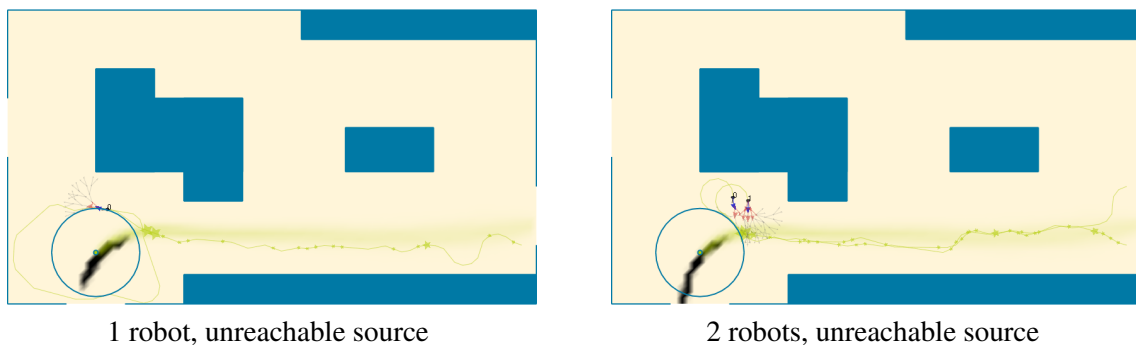


Figure 4.14: Tracking an unreachable source in the train station scenario. The robot is not allowed to enter the circle in which the source is located, but nevertheless comes up with a very good estimate of the source location.

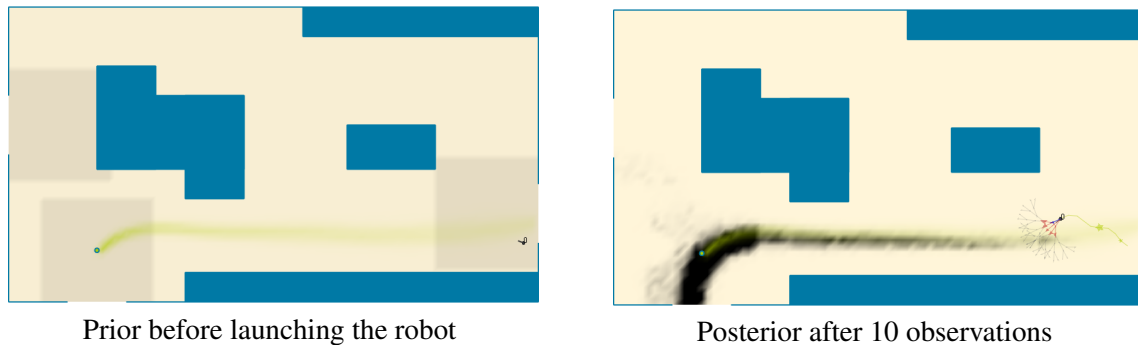


Figure 4.15: *Tracking an source using a non-uniform prior belief. The areas around the doors are assigned a 10 times higher probability of hosting the source. Such information helps the robot spotting the source.*

to these areas. After 10 observations (with just 2 odor hits), the robot has spotted the door, and narrowed down the area. Since locations outside the areas in front of the door are just less likely, but not impossible to host the source, the robot also considers the area downwind from that door somewhat likely.

This is important, as shown in Figure 4.16. In this example, the prior was wrong and the source elsewhere. At the beginning (after 8 observations), the robot assigns a very high probability to the area in front of the leftmost door. Through subsequent measurements, the corner where the source really is obtains more and more probability mass, and just before reaching the source (after 53 observations), the robot can exclude most of the area in front of the leftmost door as potential source location.

4.3.6 Surveillance and Area Coverage

In the previous examples, we mostly looked at plume tracking in different situations. The probabilistic model used here is also fairly good at doing area coverage to show that no source is present. This can be interesting for surveillance tasks, for example, where no source is available for most of the time, or for area reduction in humanitarian demining [86].

Figure 4.17 shows the trajectory of a robot doing surveillance. No source has been placed, and in each step, the robot assumes with a 1 % chance that a source has just been placed somewhere in the room. To model this, the prior source state distribution of the current step is not simply the posterior distribution of the previous step, but a weighted sum of this posterior distribution (99 %) with a uniform distribution (1 %). Hence, the robot needs to continuously sweep through the room to prove that there is still no source around. (If it stopped taking measurements, the probability distribution would converge towards a uniform distribution after some time.)

The robot first sweeps in a zig-zag fashion towards the right (downwind), and can thereby mark large parts of the room as unlikely to host a source. Even areas quite far away from the trajectory can be marked as “unlikely” using the plume propagation model. In the second part, the robot stays in the downwind area and keeps moving along similar trajectories there. Indeed, measuring in the downwind area is almost sufficient to detect an odor source should one appear anywhere in the room, as its plume would ultimately be carried to the downwind area.

The only two areas which are not well covered by this are the two corners on the left, and the robot will never go up there with the current configuration. The reason is simple: the robot’s coverage strategy is near-optimal locally, but not globally. It never evaluates whether exploring

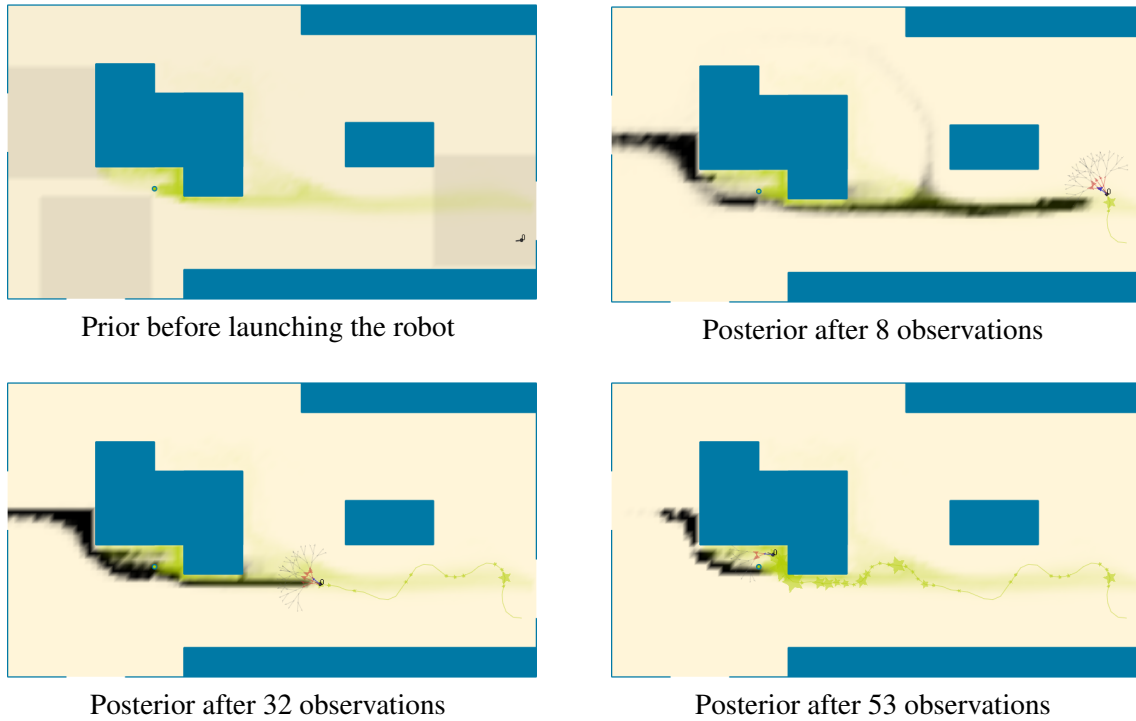


Figure 4.16: Tracking an source using a non-uniform prior belief that turns out to be wrong. The areas around the doors are assigned a 10 times higher probability of hosting the source, but the source is located outside of these areas.

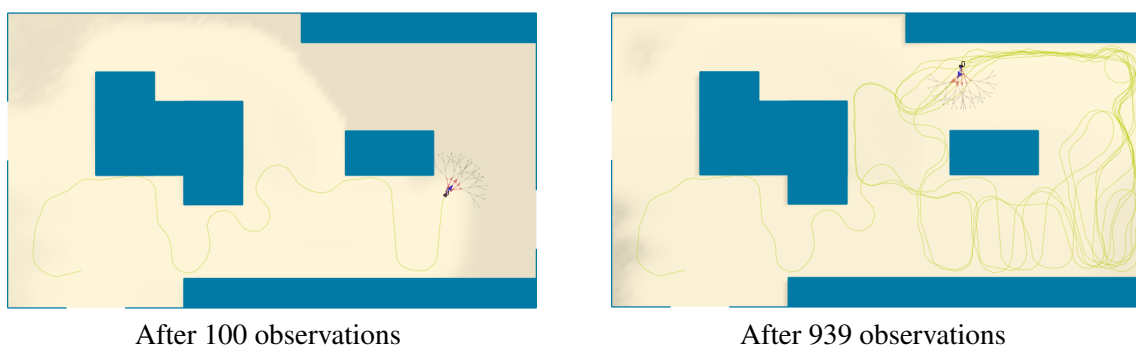


Figure 4.17: Coverage (or surveillance) task to monitor whether there is a source located in the room. As long as no source is available, the robot stays in the downwind area to maximize its chances of finding the source, should one suddenly appear.

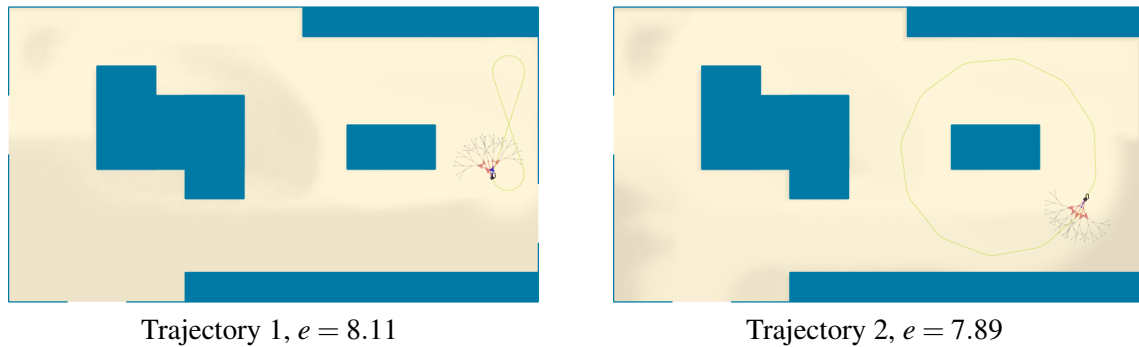


Figure 4.18: *Two preprogrammed trajectories for a patrolling mobile robot and their evaluation.*

these corners would be helpful or not, and therefore never gets to explore them.

To get around this issue, the robot could evaluate global targets on the map [17], instead of (or in addition to) the local targets around its current position. We do not dig further into probabilistic coverage for odor sources here and leave that research to future work.

4.3.7 Analyzing the Optimality of a Given Trajectory

There are a number of other algorithms for solving coverage problems with mobile robots. In addition, real-world scenarios may put a lot of other constraints. Hence, instead of letting the robot move autonomously, it may be necessary to preprogram a fixed trajectory along which the robot patrols.

In such a scenario, the probabilistic model can be used to analyze the trajectory with respect to optimality and zones that are not well covered. Two such trajectories are shown in Figure 4.18. The results show that by patrolling along trajectory 1, only a small area is covered. Trajectory 2, which is about twice as long, covers a much bigger part of the total area, and only leaves a few zones unsupervised. This is also reflected by the entropy, which is significantly lower for trajectory 2.

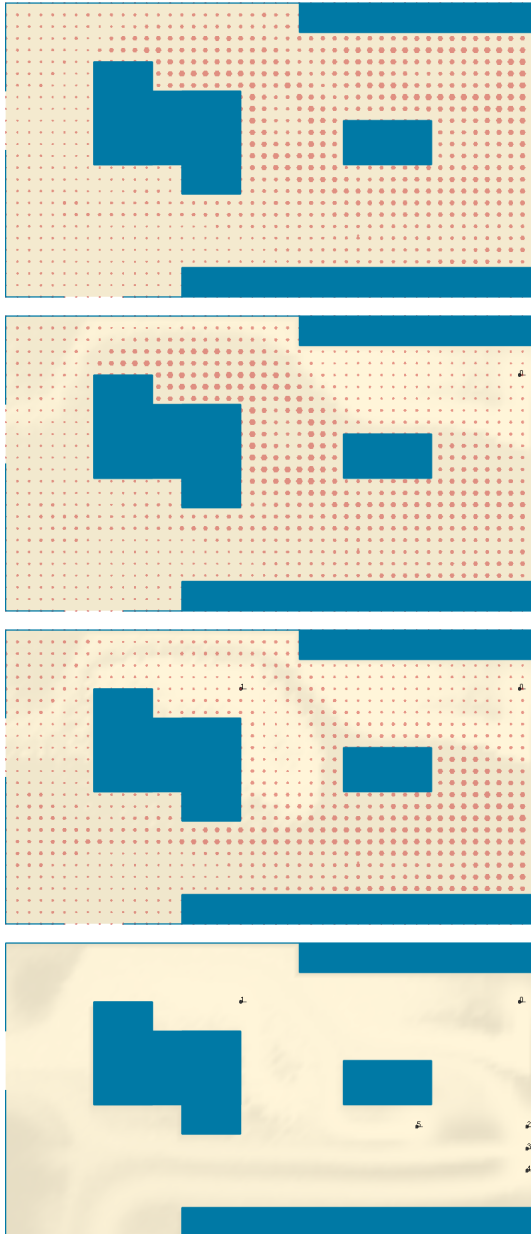
Note that a similar procedure could also be used to derive the optimality of a trajectory in presence of a source.

4.3.8 Placing Static Sensors

Instead of letting a robot sweep through the train station room all day, it would be easier (and probably cheaper!) to mount a number of static sensors in this room. From a modeling perspective, a static sensor is mostly to a robot that does not move. Within our framework, a static sensor is a robot with a path planner which offers only one possible trajectory: standing still. As for any robot, we assume the exact location of the sensor is known, and its measurements (observations) as filtered into the probability distribution just as any other observation is.

In addition, the probabilistic model can calculate optimal locations for these sensors. To do that, let us assume a sensor can jump to any location in the area of interest. This can easily be implemented as a path planner which suggests plenty of viable target locations. After evaluation of all these targets (decision step), the target with the lowest expected entropy designates the best place for a sensor node. The decision step algorithm is thereby exactly the same.

Obviously, one sensor is not enough to cover the whole room. To deploy multiple sensors, we first need to define the purpose of our network of sensors. In contrast to a robot, which attempts



Locations (lattice with 40 cm spacing) for the first sensor node are evaluated. Guided by this data, but having constructional constraints in mind, the user places a base station in the upper right corner.

Locations for the second sensor node are evaluated under the assumption that sensor node 1 did not measure any odor. The second sensor node is placed in proximity of the kiosk in the middle of the room.

Using the same procedure, three additional sensors are placed in proximity of the rightmost door, and one next to the vending machine.

The resulting sensor network (6 sensors) covers the majority of the area. Only the areas shaded in gray are not yet well covered by the sensors. To further optimize the location of these 6 sensors, each sensor could now be removed and replaced while holding the other 5 sensors at fixed locations.

Figure 4.19: Finding good sensor node locations in the train station scenario. Sensor locations are evaluated on a regular 40x40 cm lattice (red hexagons). Larger hexagons denote lower expected entropy, and therefore better location quality. The algorithm applied here is a heuristic which finds good locations within a reasonable time.

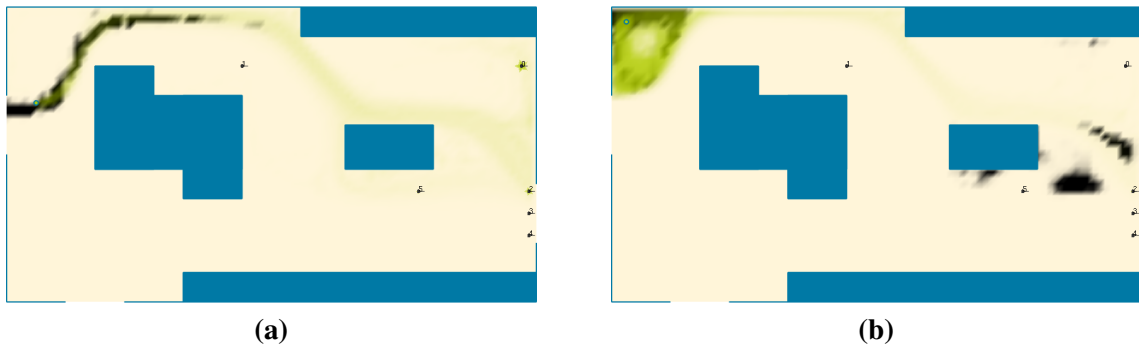


Figure 4.20: Coverage (or surveillance) task with 6 sensors to monitor whether there is a source located in the room. The sensor network not only reports that there is a source in the room, but also provides hints about its location.

to localize an odor source, the purpose here is to raise an alarm whenever a source appears. The problem is not that of source localization, but that of room coverage. Hence, any sensor node should be placed at the location yielding the lowest expected entropy, given that none of the other sensors has detected any odor (for some time).

Sensor node locations are therefore highly dependent (probabilistically), and calculating optimal locations for all nodes grows exponentially with the number of nodes. Since each node has plenty of potential target locations, this approach is computationally not viable, even for a small number of nodes.

Fortunately, good results can be achieved by heuristically placing (or moving) one sensor after the other. When looking for the optimal location of a sensor, the position of all other sensors is thereby assumed to be fixed. The following procedure can be applied iteratively to place (or move) sensors to good locations:

1. Initialize the source probability distribution (e. g., uniformly).
2. Place all sensors in the environment for which locations have been found already (if any).
3. For each sensor, filter one (or more) no-odor observations into the source probability distribution.
4. Evaluate the entropy decrease of all potential locations for the sensor node to add, and chose the best.

Figure 4.19 shows a sensor placement example based on this procedure. For all practical purposes, the sensor locations found by applying these steps are probably good enough. Mounting such sensors in a building will anyway heavily depend on construction issues, and a mathematically optimal solution will hardly be implementable. In such circumstances, it is better to find a new sensor location given the actual (and not the ideal) locations of all other sensors.

4.3.9 Surveillance with Static Sensors

Figure 4.20 shows two examples of source detection by the 6 static sensors placed before. In example (a), the source was placed at a location that is well covered by the sensor network and the sensors come up with a fairly good estimate of the source location. Indeed, since sensors 3 and 4 do not measure anything, the source is almost certainly not in the lower part of the room. From

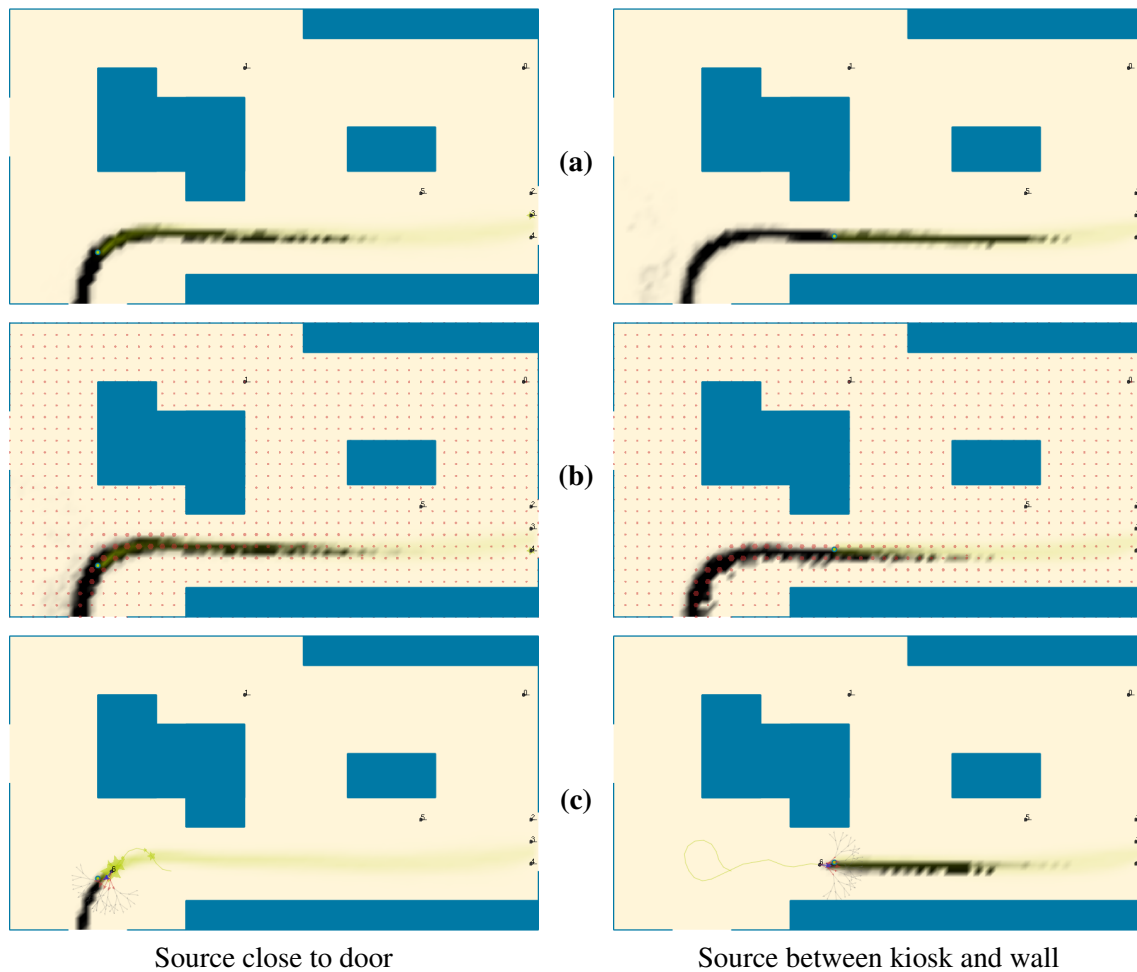


Figure 4.21: *Odor source localization with 6 static nodes and 1 mobile robot in the train station scenario. (a) The sensor network raises an alarm, but is unable to tell where exactly the source is. In particular, it is unable to discriminate between the two sources. (b) The probabilistic model suggests deploying a robot between the lowermost door and the kiosk (area with the biggest red hexagons). (c) In both cases, the robot finds the source within a very short time.*

the concentrations measured mainly by sensors 0 and 2, the probabilistic model can deduce that the source is likely to be close to the leftmost door, or downwind from there.

In example (b), the source was placed at a location not well covered by the sensor network. It is still detected, but the probabilistic model has some doubt about its location. Several badly covered areas are candidates for hosting the source, and obtaining a more accurate estimate with the current sensor network is not possible.

4.3.10 Source Localization with Static Sensors and Mobile Robots

In order to track down an odor source once the sensor network has raised an alarm, mobile robots could be deployed. In Figure 4.21, two scenarios are shown where the sensor network comes up with roughly the same belief about the source position, even though the sources are at different places. The probabilistic model then suggests taking measurements in the area where it suspects the source to be. In both cases, a robot is deployed in this area, which finds the source within a short time.

4.3.11 Conclusion of the Train Station Scenario

The train station examples shown above demonstrate the usefulness of the probabilistic model for a variety of tasks. The core algorithm, consisting of an update step and a decision step, is thereby exactly the same all the time, and only the inputs are modified:

- ▷ To guide a robot to a source (plume tracking, source localization, search), the trajectories proposed by the path planner are evaluated by the decision step algorithm, and observations are filtered by the update step algorithm.
- ▷ To cover an area with a mobile robot (coverage), the update step algorithm is fed with observations that no plume is measured.
- ▷ To place static sensor nodes, the decision step algorithm evaluates all potential locations for that node.
- ▷ Similarly, to deploy a robot, the decision step algorithm evaluates all potential release locations.

Hence, this framework is extremely flexible and general.

In addition, the algorithm can easily be integrated into (or combined with) other systems and processes. We have seen how prior evidence can be integrated, and how information collected during plume tracking can help human investigators even if the robot does not reach the source.

It is clear that this toy example neglects many real-world issues that would have to be addressed when applying this model to a real scenario. In particular, it may not be trivial to obtain a faithful and yet detailed plume propagation model, especially for dynamic environments. Nevertheless, we believe that such issues can be solved, although some will offer interesting research opportunities.

4.4 Lightweight Robot-Centric Implementation

When trying to use the map-centric implementation used in the previous sections on the real robots, it quickly turned out that the computational power of the Korebot was not sufficient to carry out all the calculations within a reasonable time. In brief experiments (that we do not further elaborate here), calculating one complete step (i. e., one update step plus one decision step with 4 targets) took about 50 seconds.

In addition, the implementation of the previous section assumes perfect global localization of the robot. While we could emulate this in the wind tunnel using the camera system, global localization is difficult to achieve in some potentially interesting environments, such as disaster recovery areas.

We therefore passed to a second implementation of the probabilistic model, but this time with a reference frame relative to the robot's pose. This new algorithm is robot-centric and inherently designed to run in a fully distributed fashion (if multiple robots are used) on the robot(s). Furthermore, the calculation overhead was reduced with a number of tricks:

- ▷ Since the processor on the robot does not have native floating point support (see Table 2.1), the algorithm was implemented with **fixed-point integers**. Additions and multiplications, which represent the majority of the operations, can thereby be sped up by a factor of 2 to 5. If implemented carefully, the precision does not suffer, but the time needed to implement and test an algorithm is multiplied by about 5 to 10.

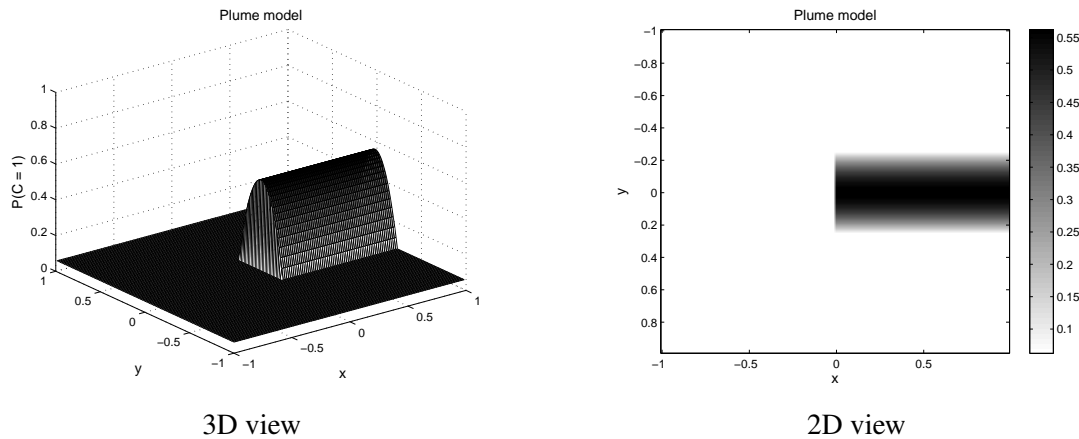


Figure 4.22: Two different views of the plume model used for the robot-centric implementation of the probabilistic model. The model is general and can be evaluated at any point with a few CPU instructions only. In both pictures, the source is at $(0,0)$, and the wind blowing in positive X direction. The plume extends to infinity, but only a 2×2 m area around the source is plotted here.

- ▷ **Parameters were approximated** to the nearest power of 2 whenever this made sense. This way, many divisions could be replaced by bit shift and mask operations which execute as fast as integer additions. (Yet, the algorithm parameters are still kept in SI units!)
- ▷ Integer versions of the functions *sin* and *cos* were implemented with a lookup table and linear interpolation. Furthermore, $\sin(\alpha)$ and $\cos(\alpha)$ are calculated in one step, as they are always used together (for rotations).
- ▷ To generate **random numbers**, we use the Mersenne Twister algorithm [87] which is faster (and better) than the built-in *rand()* function.
- ▷ The source belief distribution is approximated with up to 1024 particles, and observations are integrated using **particle filtering**. The number of particles is a trade-off between accuracy and computational overhead, whereby the latter grows linearly with the amount of particles. This allows for fine-grained adjustment of this trade-off with respect to the available computational power.
- ▷ Particles cover an **area of only 4 m by 4 m** in which the robot is in the center. If the source is further away, this is enough to make the robot go in the right direction.
- ▷ We are using **lazy updates** in several places. Particle weights, for instance, are normalized only when needed. The robot is always around the center, but not exactly in the center of the reference system. Furthermore, we never rotate the reference system, as rotating all particles is more expensive than rotating the target points.
- ▷ At each step, we move the particles to a random location near their current location. This effectively flattens out the source probability distribution, and is a simple and fast way to account for odometry errors, wind sensor noise, or other inaccuracies.
- ▷ To move more particles to interesting areas, big particles are split into two, and an equal amount of small particles are removed. This way, the number of particles always remains constant.

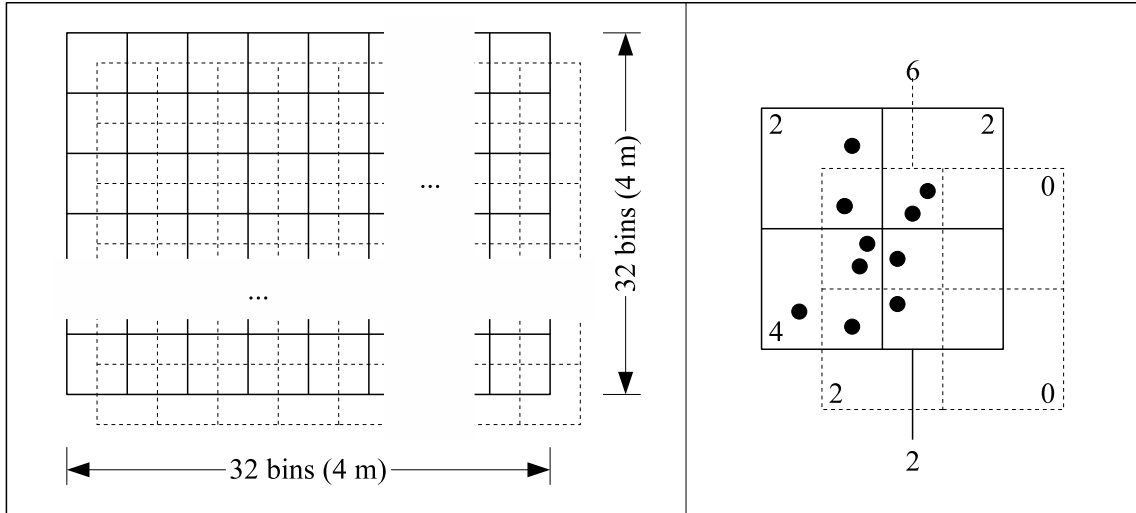


Figure 4.23: Histograms used to evaluate the quality (as replacement of the entropy) of the particles. Each histogram consists of 32×32 square bins arranged regularly over the 4×4 m area around the robot. The histograms are shifted by half a bin in both directions to minimize effects of the bin borders.

- ▷ To forget history, and to avoid particles being caught in one area (known as the kidnapped robot problem in probabilistic self-localization), 8 particles are moved to random locations (in the 4 m by 4 m area around the robot) at the end of each step. Particles are chosen in a round-robin fashion, which guarantees that a particle is moved to a random location every 128 observations when using with 1024 particles, and even earlier when using fewer particles. Put differently, the robot completely forgets observations that have happened more than 128 steps ago.
- ▷ The **plume model**, depicted in Figure 4.22, is kept simple and general. It is generated by the function

$$P(c = 1|x, y) = \frac{1}{16} + \frac{1}{2} \cdot \begin{cases} 1 - 16y^2 & \text{if } x > 0 \text{ m and } -0.25 \text{ m} < y < 0.25 \text{ m} \\ 0 & \text{otherwise} \end{cases} \quad (4.21)$$

and rotated according to the current wind direction measured by the robot.

- ▷ **Observations are binary**, which greatly reduces the number of potential observations to evaluate in the decision step.
- ▷ As **calculating the information entropy** with fixed-point integers in the decision step is tricky and lengthy, it is replaced by a function that is easier to calculate. Particles are thereby collected in two histograms with 1024 bins, in the way depicted in Figure 4.23. Let us denote the bins of one histogram as a_i , and those of the other histogram as b_i . The *quality* q is then calculated as

$$q_a = \frac{1}{\sum_{i=1..1024} a_i} \sum_{i=1..1024} a_i^2 \quad (4.22)$$

$$q_b = \frac{1}{\sum_{i=1..1024} b_i} \sum_{i=1..1024} b_i^2 \quad (4.23)$$

$$q = \frac{1}{2}(q_a + q_b) \quad (4.24)$$

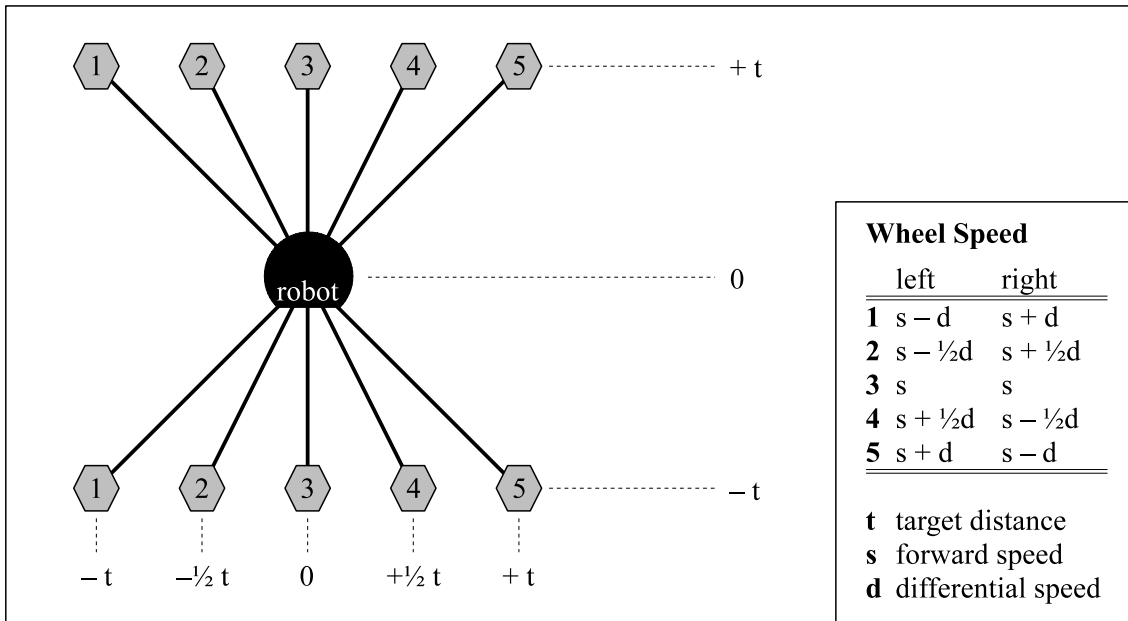


Figure 4.24: Target positions (hexagons) that are evaluated at each step. In each iteration of the algorithm, all 10 target positions are evaluated, and the best target dictates the wheel speed.

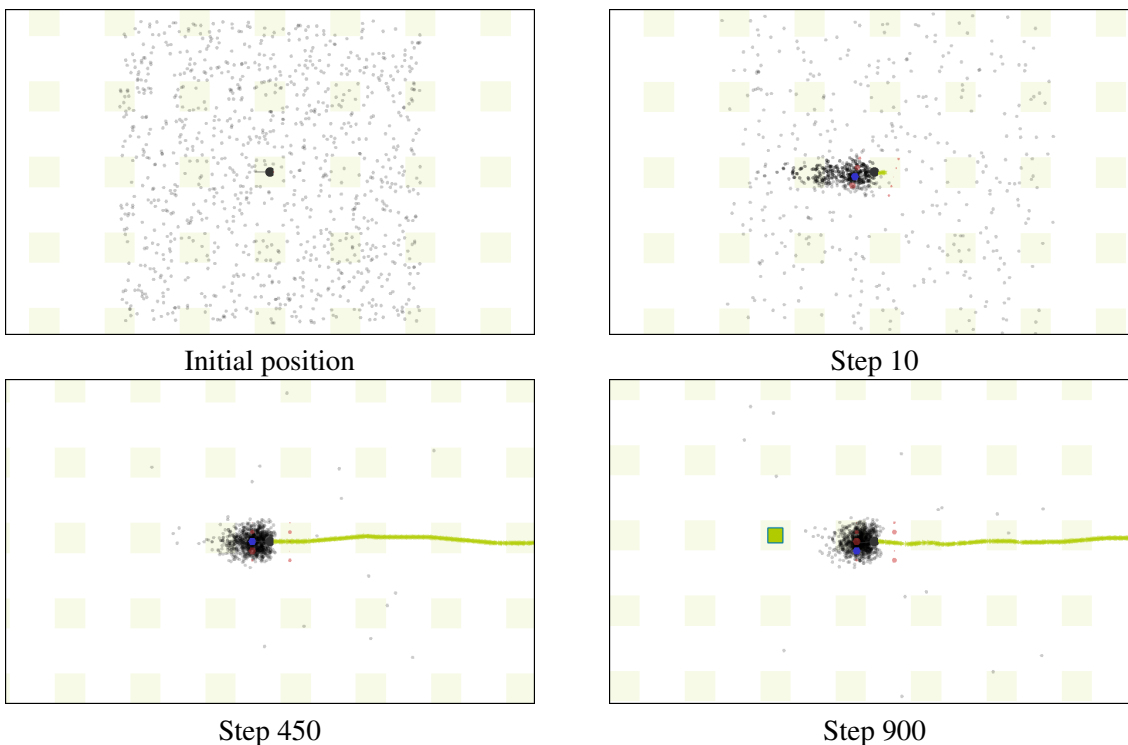


Figure 4.25: Screenshots of the simulation program. The black circle (to scale) is the robot, and the yellow tail its trajectory. Particles are drawn as small gray hexagons, and darker particles have more weight. The yellow tiles in the background are spaced by 1 m and only serve visualization purposes. The odor source, represented with a green square, is only visible in the screenshot of step 900.

Using two histograms in this way reduces bin border effects. With a single histogram, similar particle distributions could result in significantly different q values. Note that creating a histogram and squaring its values is a matter of only about two dozen operations per particle, plus half a dozen per histogram bin. As this function is called twice for each target position to evaluate, this is an important optimization. (Note that this is much cheaper as compared to approximating the distribution with a normal distribution, for which the entropy is known.) q exhibits an inverse behavior with respect to the entropy: the higher q , the more we know about the position of the source.

- ▷ Only the 10 target positions depicted in Figure 4.24 are evaluated in each decision step. No path planner tree is used and no multi-step-ahead calculation performed. The goal is just to figure out whether going left, right, or straight is better in the current situation. The target positions therefore do not represent actual points that the robot will drive to — the robot only drives a few millimeters from one step to the next — but are located much further away from the robot. This can be compared to multi-step-ahead calculation, except that intermediate observations are not considered in the evaluation.

Despite all the simplifications and approximations, the algorithm still implements the probabilistic model presented in Section 4.1. A few screen shots of the simulation program are depicted in Figure 4.25.

4.4.1 Complexity and Execution Time

The complexity (per robot) of this implementation is only

$$O(\#\text{particles} \cdot \#\text{targets}) \quad (4.25)$$

which is the complexity of the decision step. This is mostly the result of avoiding multi-step-ahead calculation, and representing the distribution with particles.

In addition, the constant factor in this complexity is very low. On the Korebot processor board on the Khepera III robot, steps can be executed at up to 11.5 Hz, including the overhead to communicate with the sensors (approx. 5 ms per step) and actuators (approx. 1 ms per step).

4.4.2 Abstract Simulation

In order to test the implementation and to check the influence of the parameters, we first carried out a number of simulation experiments at a high abstraction level. The wind direction, w_a , was thereby set to

$$w_a \sim \text{Uniform}(-w_e, w_e) \text{ with } w_e = \frac{360^\circ}{32} = 11.25^\circ \quad (4.26)$$

and the odor hits were randomly sampled with probability

$$P(c = 1|x, y) = \begin{cases} 1 - 16y^2 & \text{if } x > 0m \text{ and } -0.25m < y < 0.25m \\ 0 & \text{otherwise} \end{cases} \quad (4.27)$$

where x and y denote the robot position relative to the source, and the wind is blowing in positive X direction. This is the same as the plume model (see equation (4.21)), but scaled differently. The robot was simulated as a differential drive robot with the same wheel and axis parameters as the Khepera III robot.

The following sets of experiments were launched:

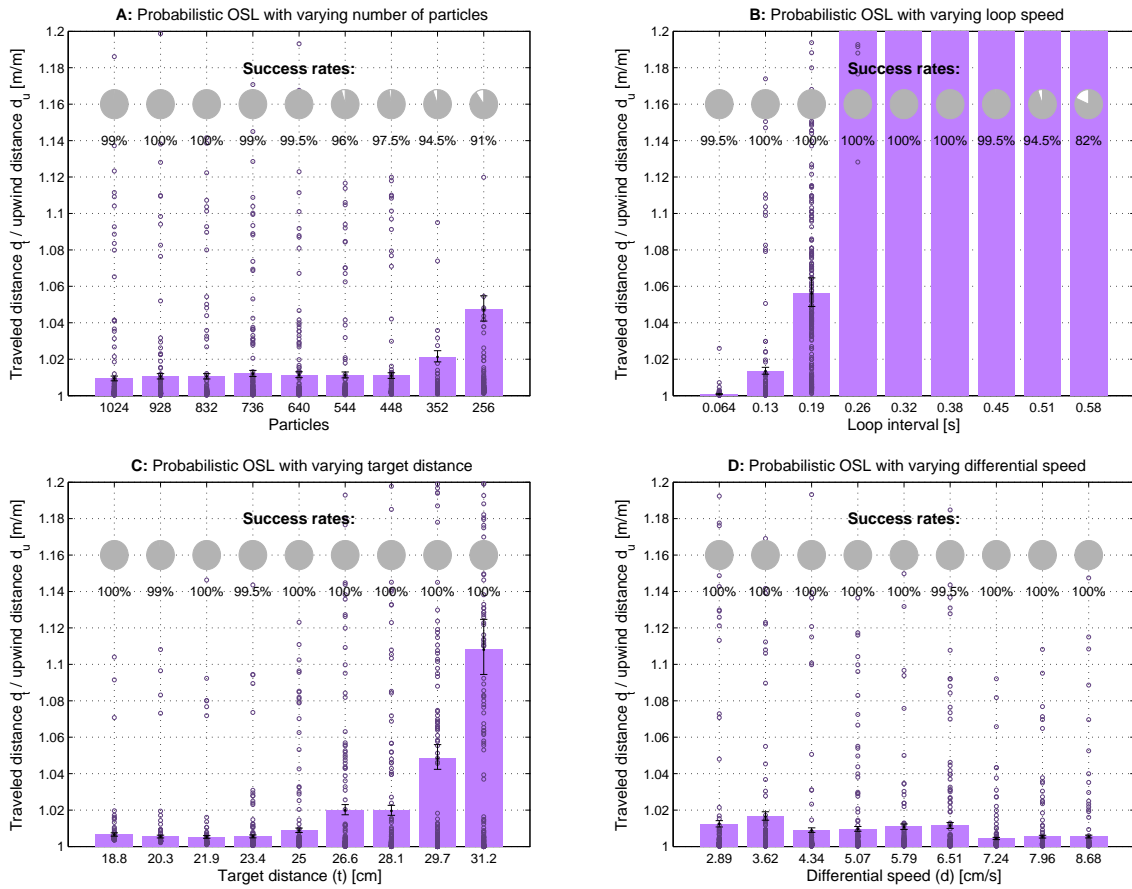


Figure 4.26: Distance overhead and success rate obtained with the point simulator for different configurations. The error bars indicate the 95% confidence interval for the mean (assuming exponentially distributed data), while the dots stand for results of individual runs. **A:** With varying number of particles. **B:** With different loop intervals. **C:** With different target distances. **D:** With different differential speeds.

Algorithm	Particles	Loop interval	t	s	d
A Probabilistic OSL	variable	128 ms	25 cm	10.6 cm/s	5.8 cm/s
B Probabilistic OSL	1024	variable	25 cm	10.6 cm/s	5.8 cm/s
C Probabilistic OSL	1024	128 ms	variable	10.6 cm/s	5.8 cm/s
D Probabilistic OSL	1024	128 ms	25 cm	10.6 cm/s	variable

t , s , and d are the target distance, the forward speed and the differential speed, respectively, defined as in Figure 4.24. Each set consists of 9 choices for the variable parameter with 200 independent runs each. In each run, the robot was released in the odor at a position 14 m downwind from the odor source. If the robot reached the odor source within 256 s ($d_t \approx 27$ m), the run was considered successful. Distance and upwind distance were derived from the trajectory, recorded during the run. The forward speed of the robot (on straight lines) was 10.6 cm/s and therefore same as for the experiments with the bio-inspired algorithms in Section 3.2.

The results are plotted in Figure 4.26. Most noticeable is the dependence on the loop speed. For high loop intervals (low loop speeds), the performance drops significantly. Three reasons can be attributed to that: First, as only one odor measurement is taken (and filtered into the source

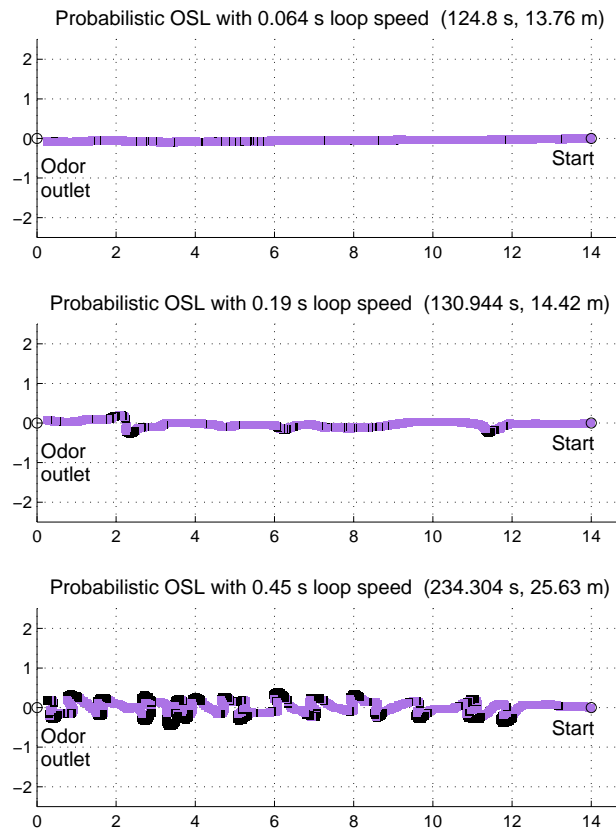


Figure 4.27: Successful point simulation runs with different loop speeds. All runs are with 1024 particles and one odor concentration measurement is taken per loop iteration. With lower loop speeds, the algorithm produces more casting behavior, yielding very long trajectories.

probability distribution) per iteration of the loop, far less information is available to the algorithm. Second, each decision results in a trajectory of several centimeters until the next decision is taken. This is a substantial distance as compared to the plume width. Third, the robot moves too fast for particles to stay in the 4 x 4 m area around it, and the heuristic algorithm moving the particles to places of high probability is not able to keep up with that speed. Hence, within a dozen steps, most particles are lagging behind the robot and only very few particles are modeling the source distribution in front of the robot. Note that this last point is only an implementation issue — the way particles are redistributed could be changed to fix this issue. Figure 4.27 displays trajectories of a few runs with different loop speeds, where the impact of the loop speed on the trajectory can be observed.

Interestingly, the number of particles (over the range from 256 to 1024 particles) does not have a big influence on the performance in this point simulation. Even the result for 256 particles (distance overhead of 1.05, 91 % success rate) is still good as compared to the best bio-inspired algorithms. Since we are using a perfect plume model (with respect to the simulated plume), the probability distributions are very “nice”, and a low number of particles are enough to approximate it.

Another very positive result is that the performance is over a wide range almost independent from the differential speed. The algorithm will simply keep turning in one direction until it believes the robot is heading the odor source again.

Overall, the results are extremely good. Over a large range of parameters, the success rates are above 99 %, and the average distance overheads, d_o , roughly 1.01. Hence, in ideal conditions, this probabilistic algorithm achieves near-optimal performance.

4.4.3 Robotic Simulation (Webots)

After the simulations at a high abstraction level, we carried out experiments with Webots [72] using the odor simulation plugin presented in Section 2.2. Exactly the same controller with the same configuration was used, except that sensors and actuators were simulated by Webots. The measured odor concentration, $c(x, y)$, was thereby converted into random hits using

$$P(c = 1|x, y) = \begin{cases} \frac{\log_2 c(x, y)}{p_t} & \text{for } \log_2 c(x, y) < p_t \\ 1 & \text{otherwise} \end{cases} \quad (4.28)$$

In this equation, p_t is the concentration ceiling, a free parameter of the algorithm. The concentration readings were passed through the \log_2 function to account for the large dynamic range of the odor concentration measurements. The simulation setup with all parameters was the same as for the bio-inspired algorithms (see Section 3.2.2), and the results are therefore comparable.

The following sets of experiments were launched:

	Algorithm	Particles	p_t	σ_a	Loop int.	t	s	d
E	Prob. OSL	variable	9	10 cm	128 ms	25 cm	10.6 cm/s	5.8 cm/s
F	Prob. OSL	1024	variable	10 cm	128 ms	25 cm	10.6 cm/s	5.8 cm/s
G	Prob. OSL	1024	9	variable	128 ms	25 cm	10.6 cm/s	5.8 cm/s

Each set consists of 9 choices for the variable parameter with 200 independent runs each. In each run, the robot was released in the odor at a position about 14.5 m downwind from the odor source. If the robot reached the odor source, the run was considered successful. If the robot touched an arena wall, the run was aborted and declared unsuccessful. Distance and upwind distance were derived from the trajectory, recorded during the run. The forward speed of the robot (on straight lines) was 10.6 cm/s and therefore same as for the experiments with the bio-inspired algorithms in Section 3.2.

The results are plotted in Figure 4.28. As observed with the point simulator already, the number of particles (over the tested range) only has a minor influence on the performance. Runs with lower numbers of particles tend to be slightly less successful.

The wind sensor noise has a major impact on both the distance overhead and the success rate. We observe an almost binary behavior: up to $\sigma_a = 16$ cm, the performance is very good, while for $\sigma_a \geq 32$ cm, the performance drops significantly and abruptly. Indeed, our probabilistic model does not model any noise for the wind direction sensor, i. e., any measured wind direction is considered accurate. This is a good approximation for accurate wind directions sensors, but entails devastating side effects for noisy wind direction sensors. Taking this noise into account is straightforward from a theoretical point of view, but computationally rather expensive for our highly optimized implementation.

The concentration ceiling, p_t , has a small impact on the distance overhead. There seems to be an optimal point at 8 in our simulation, but even with $p_t = 11$, the performance is still very good as compared to the bio-inspired algorithms. This implicitly means that the algorithm can deal with sources of unknown intensity (at least up to some extent).

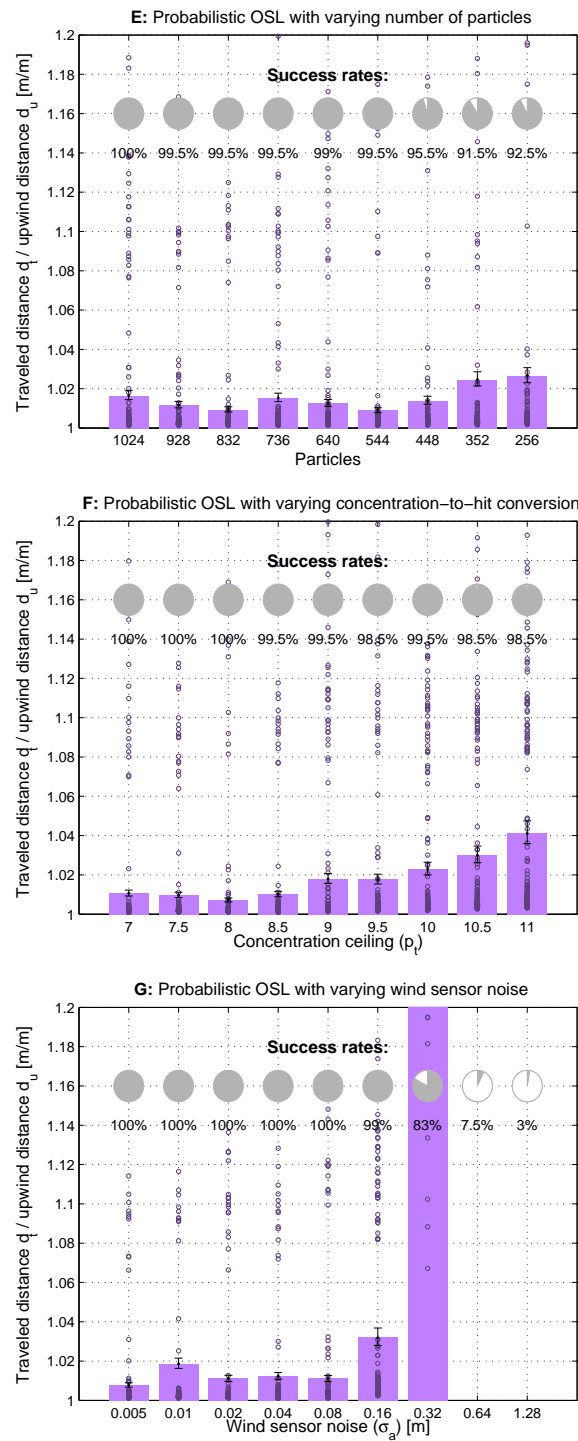


Figure 4.28: Results obtained in Webots. The error bars indicate the 95 % confidence interval for the mean (assuming exponentially distributed data), while the dots stand for results of individual runs. **E:** With varying number of particles. **F:** With varying concentration ceiling. **G:** With varying wind sensor noise. Note that the bars for 0.64 m and 1.28 m are omitted because of the low success rate indicating that the algorithm was clearly not well configured for these environments.

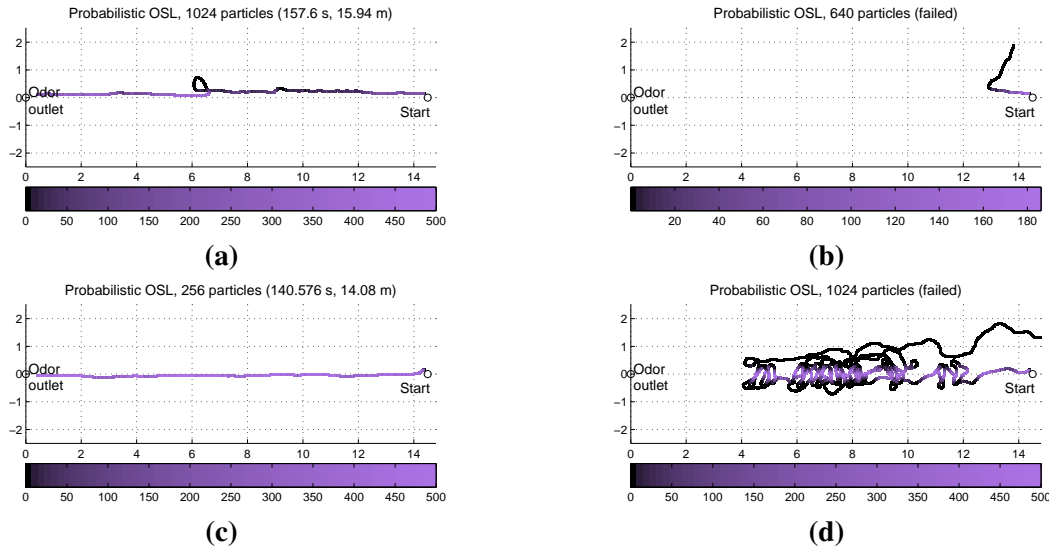


Figure 4.29: *Webots simulation runs of the probabilistic odor source localization algorithm. (a) The algorithm produces casting, spiraling, and upwind surge, even though none of these behaviors are explicitly coded. (b) In rare cases, the robot does not manage to turn back towards the plume when it loses it. (c) In contrast to the bio-inspired casting algorithm, probabilistic odor source localization takes into account different concentration levels and corrects its heading before losing the plume completely. The trajectories of many runs are therefore almost straight. (d) For very high wind sensor noise, the current implementation of the algorithm produces a lot of irregular counter-turning and loses the plume after a while.*

4.4.4 Real Robots

We finally carried out real-robot experiments with the probabilistic odor source localization controller. Again, the same controller was used, except that the real sensors and actuators served as input to the algorithm. The response of the odor sensor, $c(x,y)$, was thereby converted into hits using

$$P(c = 1|x,y) = \begin{cases} 0 & \text{if } c(x,y) - b < 0 \\ \frac{c(x,y)-b}{p_t} & \text{if } 0 \leq c(x,y) - b \leq p_t \\ 1 & \text{otherwise} \end{cases} \quad (4.29)$$

In this equation, p_t is the concentration ceiling, a free parameter of the algorithm, and b is the odor concentration baseline. The setup with all parameters was the same as for the bio-inspired algorithms (see Section 3.2.1), and the results are therefore comparable.

The following experiments were carried out:

	Algorithm	Particles	p_t	Loop interval	t	s	d
H	Probabilistic OSL	1024	4096	131.4 ms	25 cm	10.6 cm/s	5.8 cm/s
H	Probabilistic OSL	768	4096	132.5 ms	25 cm	10.6 cm/s	5.8 cm/s
H	Probabilistic OSL	512	4096	131.1 ms	25 cm	10.6 cm/s	5.8 cm/s

Each experiment consists of 20 independent runs. In each run, the robot was released in the odor at a position about 14.5 m downwind from the odor source. If the robot reached the target area around the odor outlet (determined with the floor sensors), the run was considered successful. During the run, the trajectory (using odometry) and the odor concentration were recorded.

Table 4.2: Comparison of the settings for the real-robots experiments, the Webots simulations, and the simulations under the abstract model.

	Real robots	Webots sim.	Abstract sim.
Environment	wind tunnel	Webots [72]	–
Wind	\approx laminar	laminar	laminar
Plume	real, ethanol	filaments [7]	model
Plume width (w)	\approx 35 cm	35.4 cm	50 cm (end-to-end)
Robot	Khepera III	Khepera III	point
Locomotion	diff.-drive	diff.-drive	diff.-drive
Mean speed (s)	10.6 cm/s	10.6 cm/s	10.6 cm/s
Odometry	good	perfect	perfect
Particles	1024	1024	1024
Loop interval	\approx 132 ms	128 ms	128 ms
Target distance (t)	25 cm	25 cm	25 cm
Target differential (s)	5.8 cm/s	5.8 cm/s	5.8 cm/s
Wind sensor error	non-Gaussian	$N(0, (10^\circ)^2)$	$U(-11.25^\circ, 11.25^\circ)$
Odor sensor error	negligible	Gaussian, small	0
Odor sensor delay	$t_{90\%} \approx 0.1$ s	none	none

Distance and upwind distance were derived from the trajectory, and the duration of each run was measured on a host computer. The forward speed of the robot (on straight lines) was 10.6 cm/s and therefore same as for the experiments with the bio-inspired algorithms in Section 3.2. The loop interval was adjusted to approximately 132 ms by inserting waiting times. With 1024 particles, this corresponds to roughly 80 % CPU utilization of the ARM processor on the KoreBot, while with 512 particles, the CPU utilization is about 50 %. A good value for p_t was determined experimentally with a few initial runs, while b was measured in clean air (inside the wind tunnel) before each run.

The results are plotted in Figure 4.30, and Figure 4.31 depicts the trajectories of 2 selected runs. As compared to the simulation results, the distance overhead and the success rate are slightly worse. But with only 8 % distance overhead when using 1024 particles, the algorithm still beats all bio-inspired algorithms. In addition, some of the runs were almost optimal.

Fewer particles lead to significantly worse performance, however. In particular, chances are much higher that the robot is engaged in an expensive plume reacquisition behavior during then run, and only few quasi-optimal runs (with distance overheads below 1.05) are observed.

Reasons for the performance drop as compared to the simulation results are two-fold: First, the error distribution of the wind sensor board is not nicely Gaussian, as this was the case in the simulation. (The error distribution over all wind angles is approximately Gaussian, but for a specific angle, it tends to be biased towards specific values.) Second, the odometry in both the point simulation and the Webots simulation was perfect, while with the real-robot results, there was an angular error of a few degrees over the whole trajectory, mostly due to a not perfectly flat floor. Since the algorithm automatically forgets old measurements, this does not matter too much, but still has some influence. In particular, the robot tends to go towards the plume borders more often than with perfect odometry.

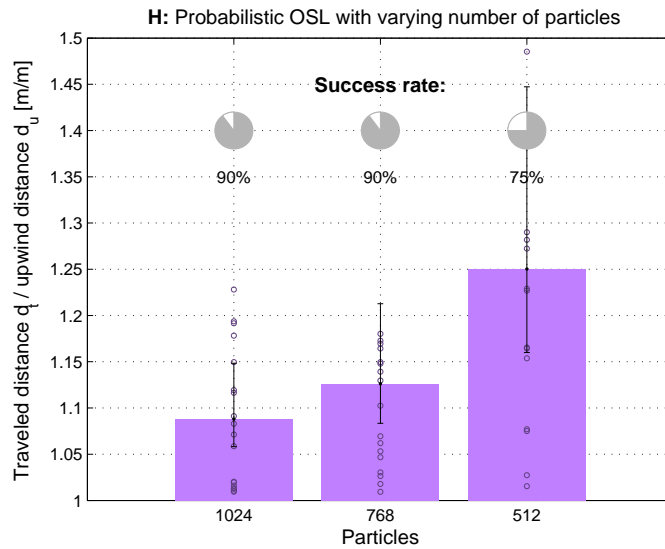


Figure 4.30: Results obtained with the real robots for varying number of particles. The error bars indicate the 95 % confidence interval for the mean (assuming exponentially distributed data), while the dots stand for results of individual runs.

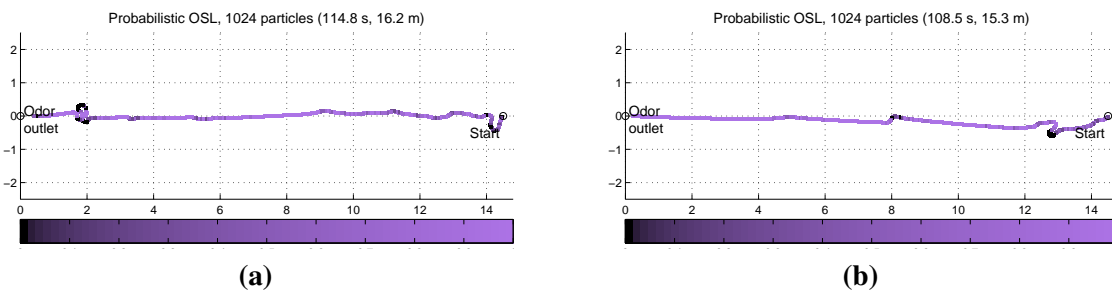


Figure 4.31: Two successful real robot runs of the probabilistic odor source localization algorithm. The trajectories look similar as those obtained in simulation.

4.4.5 Similarity to Insect Trajectories

Interestingly, the trajectories produced by this probabilistic algorithm look astonishingly similar to plume tracking trajectories of insects [48] [47] [46]. Upwind-surge, spiraling and casting can be observed, even though none of these behaviors has been explicitly coded into the algorithm. A similar correspondence has previously been found with the infotaxis algorithm [30] where agents were moving on a rectangular lattice.

What is striking here, however, is that the shape and irregularity of the trajectories is very similar to their biological counterparts. Not only pure and nice spiraling is observed, but a variety of irregularities that can be either interpreted as casting or as spiraling.

Naturally, the question arises of whether insects are using a probabilistic strategy as well. The strategy of insects has evolved over millions of years with a significant evolutionary pressure. As plume tracking is used for food scavenging and mating, individuals with better strategies had a tremendous advantage and therefore a higher survival and reproduction rate. Hence, insect plume tracking strategies observed can be assumed to be nearly optimal (for their environments and intents). The probabilistic model is greedy-optimal, and the experiments above show that it yields almost optimal performance in our environment. This match does not prove anything, but at least provide another correspondence between probabilistic plume tracking and trajectories observed in nature.

Regarding the complexity of an algorithm implementing the probabilistic model, we have shown that there exist efficient implementations with relatively low computational requirements. Our implementation here is able to track a plume with only a few 10'000 operations — mainly additions and multiplications — per loop iteration, or a few 100'000 operations per second. A honeybee with its 950'000 neurons [88], for instance, would have enough neurons to implement such an algorithm. As plume tracking is one of the most crucial behaviors for survival, it would not be surprising if a substantial fraction of these neurons were dedicated to it. In addition, an implementation based on particles can be implemented in a parallel fashion, and is robust with respect to small differences in the implementation of each particle. In other words, if the neuronal circuitry of one particle consistently fires higher than another, this would not make a substantial difference to the final behavior.

In addition, the sensory input of many insects is similar to the one of our robot. Besides odor concentration and wind direction, insects such as honeybees rely on optical flow [89] for motion estimation, a technique which provides information similar to wheel odometry.

Despite the striking similarity of the trajectories, none of these arguments serves as a proof that insects are using a probabilistic approach for odor source localization. They only underline that this may be possible regarding sensory input information and algorithmic complexity. But even if this was the case, an implementation with neurons would obviously be vastly different from a CPU-based implementation. In addition, insects are flying in a 3D space, whereas our robotic experiments were carried out in a 2D arena.

4.5 Summary and Conclusion

In this chapter, we have discussed two different implementations of the same probabilistic model. The map-centric implementation is based on a centralized architecture and relies on perfect absolute localization and global communication among all robots and nodes. The approach is completely different from the bio-inspired algorithms presented in Chapter 3. While bio-inspired algorithms are lightweight and based on minimal information about the environment, this implementation of the probabilistic model is CPU and memory intensive, and suitable for combination

with other subsystems on a mobile robot (e. g., path planner, map of the environment). The algorithm runs at reasonable speed on nowadays desktop or laptop computers ($\approx 40'000$ MIPS), but is too CPU intensive for current embedded systems (≈ 400 MIPS).

The second implementation is a lightweight robot-centric implementation using only short-term motion estimation with wheel odometry. With a number of simplifications and approximations, the execution time was drastically reduced, allowing it to be run on small embedded platforms. The plume model is kept very simple, for example, and the path planner is very basic, too. In contrast to the bio-inspired algorithms, however, this algorithm is able to cope with perturbations because it keeps track of the actual motion of the robot over short distances. Runs with this algorithm have shown stunning similarity with trajectories of real insects, even though no bio-inspired behavior is explicitly coded into the algorithm.

The probabilistic model introduced at the beginning of the chapter is a very general and flexible model from which odor source localization algorithms can be derived. It is mathematically concise and complete and specifies all that is needed to design an odor source localization algorithm. The model itself is not an algorithm, however.

How to derive a suitable algorithm (and implementation) from it heavily depends on the exact problem, the environment and the available hardware resources (CPU, memory, sensors, and actuators). Such considerations will determine how the source state probability distribution should be approximated, what an observation looks like, and which target positions are evaluated. Alongside with this, the system designer will have to think about communication among robots (in the case of multi-robot systems), available information about the environment (e. g., maps), and possibly even interfaces with human investigators.

This may sound like a complex task leaving a huge amount of choices — and it certainly is. But a major advantage of probabilistic models is that they actually make it possible to do all that, and even provide a mathematical description for it. In addition, probabilistic models for different tasks of a mobile robot can be combined at model level, even if the individual parts share very little in common. Algorithms can then be designed to implement the complete model, which is often easier than designing individual algorithms and make them work together.

This probabilistic approach to odor source localization is therefore a very promising direction which might be a suitable choice for many potential odor source localization applications.

5 Algorithms Based on Robot Formations

Experiments with the multi-robot versions of the bio-inspired algorithms (see Section 3.5) revealed that robots are competing for space even when they are communicating and collaborating. When robots avoid each other (to prevent collisions), they often lose the plume and switch to plume reacquisition, which obviously results in bad performance. Reasons for this are the state-machine nature of the bio-inspired algorithms on one hand, but also the fact that robots do not plan their path with respect to the positions of the other robots. The low-level controller (obstacle avoidance) interferes with the high-level controller (bio-inspired odor source localization algorithm).

With probabilistic algorithms, on the other hand, one can easily integrate collision avoidance constraints into the high-level controller. In addition, experiments revealed that robots would optimally explore the plume boundary to infer the location of the source, and not try to locally maximize the odor concentration. However, the computational cost of these algorithms is much higher as compared to the bio-inspired algorithms.

This brought us to the idea of designing a simple multi-robot algorithm to do boundary exploration. The robots should thereby be aware of each other's location, such that low-level obstacle avoidance does not interfere with the odor source localization algorithm.

We came up with such an algorithm based on a crosswind line formation and tested it in the same wind tunnel environment in which we previously tested the bio-inspired and probabilistic algorithms. In the remainder of this chapter, we describe the algorithm and the experiments we carried out with real robots. Note that this whole chapter is only concerned with the odor source localization aspects of these algorithms. We do not attempt to optimize formation aspects, or reduce the communication overhead.

5.1 The Crosswind Formation Algorithm

The underlying idea of the algorithm is to keep some robots on the left side of the plume, and some robots on the right side of the plume. While they are going upwind, they try to stay centered in the plume, i. e., keep the (average) concentrations on the left and on the right approximately equal.

With the present algorithm, the robots are kept on a line in crosswind direction and share their measured concentration as well as their position with all other robots in the formation. As the robots do not use any global reference system, they only know their locations relative to each other.

In a reference system defined by the wind direction, each robot calculates a (virtual) crosswind and a (virtual) upwind force, as depicted in Figure 5.1. The upwind force, f_u is

$$f_u = u + \frac{1}{N} \sum_i y_i \quad (5.1)$$

where N denotes the number of robots and u the constant upwind drag, a parameter of the algorithm. f_u keeps the robot aligned with the other robots, such that they all have approximately the

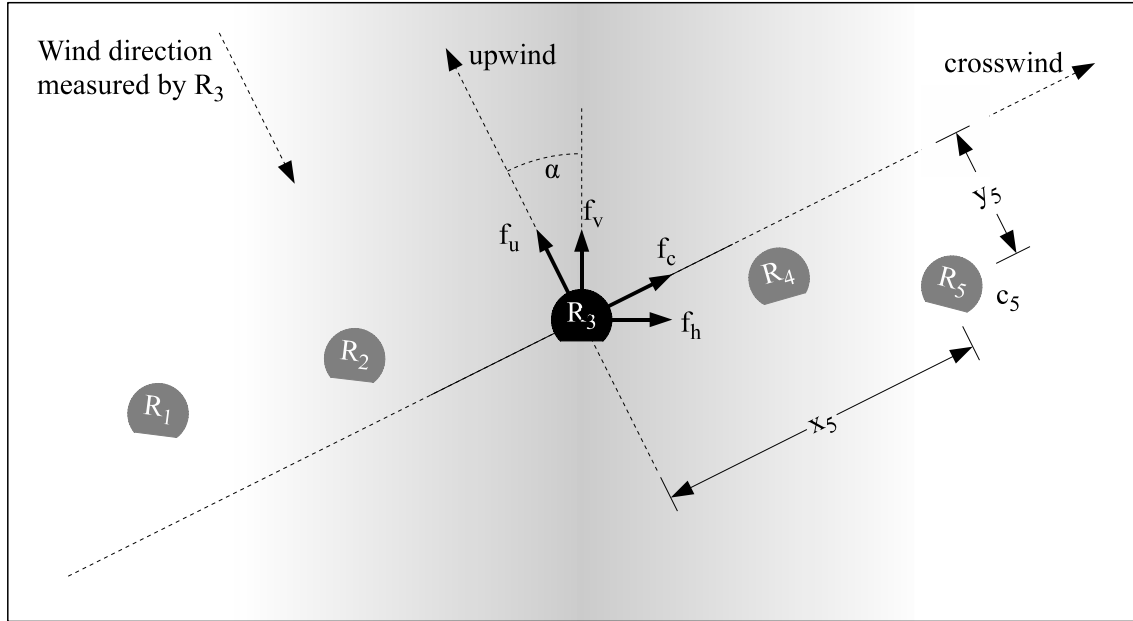


Figure 5.1: Sketch of the formation algorithm from the perspective of robot 3. Calculation of the forces f_u and f_c is carried out in a reference system defined by the wind direction. The resulting force vector is then rotated into the robot's reference system. All robots carry out the exact same calculation, but from their own perspective.

same downwind distance from the odor source. If, for instance, one robot is behind, the y_i tend to be more positive in the coordinate system of this robot and the resulting force is stronger.

The crosswind force, f_c , is a weighted difference (with weights a and r) between an attractive and a repulsive force. Formally,

$$f_c = af_a - rf_r \quad (5.2)$$

$$f_a = \frac{\sum_i x_i c_i}{\sum_i c_i} \quad (5.3)$$

$$f_r = \frac{1}{N} \sum_{i, i \neq me} \frac{1}{x_i} \quad (5.4)$$

The attractive force, f_a , takes into account the odor concentrations, c_i , measured by all other robots and is responsible for keeping the formation centered in the plume. Robots measuring a high concentration contribute more weight, and therefore pull the other robots towards them. The system is in equilibrium state if the robots on the left and on the right measure the same concentrations. The repulsive force, f_r , keeps the robot at a certain distance from all other robots.

The vector (f_u, f_c) is then rotated into the reference system defined by the robots heading,

$$f_v = f_u \cos(-\alpha) - f_c \sin(-\alpha) \quad (5.5)$$

$$f_h = f_u \sin(-\alpha) + f_c \cos(-\alpha) \quad (5.6)$$

where α denotes the wind angle relative to the robots heading. The resulting vector (f_v, f_h) is finally transformed into differential drive wheel speeds as follows:

$$s_l = s(k_v f_v + k_h f_h) \quad (5.7)$$

$$s_r = s(k_v f_v - k_h f_h) \quad (5.8)$$

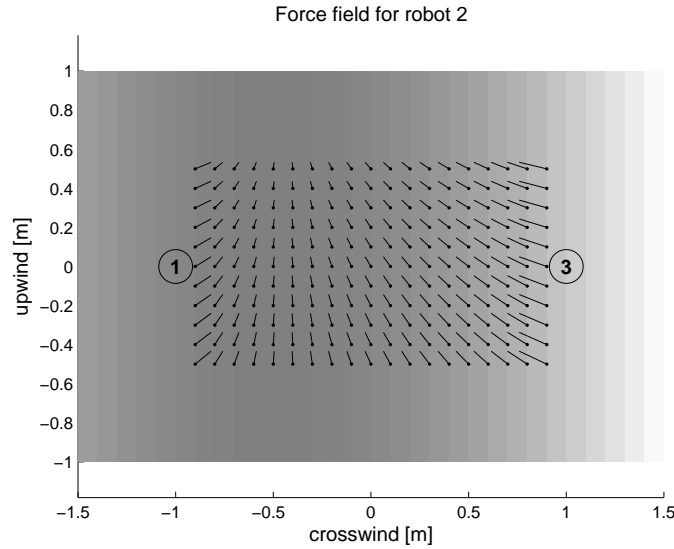


Figure 5.2: Crosswind and upwind forces (scaled) for different positions of robot 2 in a formation with 3 robots. Robot 1 is currently measuring a 4 times higher concentration as compared to robot 3. The gray shading stands for the odor concentration, which attains its maximum at $x = -0.5$ m.

k_v and k_h are thereby factors to scale the forward and differential speed appropriately, and s denotes the mean forward speed.

All robots keep executing these steps continuously in a loop. In each iteration of the loop, a robot takes one measurement with the wind direction sensor and one with the odor sensor, and broadcasts the latter to all other robots. To calculate the forces, it uses the last received odor concentration and relative position values of each robot.

5.2 Experiments with 3 and 5 Robots

We tested the algorithm with real robots in the wind tunnel in the following settings:

Algorithm	Robots	Start position	Runs
A Formation OSL	3	left	10
B Formation OSL	3	middle	10
C Formation OSL	3	right	10
D Formation OSL	5	middle	5

The experimental setup in the wind tunnel was thereby the same as with the bio-inspired and probabilistic algorithms. Relative positions were emulated using the camera system and sent to the robots via wireless LAN at a 10 Hz update rate. With the start position *left* (resp. *right*), the robots started slightly at the left (resp. right) of the plume, and only the rightmost (resp. leftmost) robot was measuring an above-baseline odor concentration. With the start position *middle*, one robot was placed in the plume center at the beginning of the experiment, while an equal number of robots started on the left and on the right of the plume. The mean forward speed of the robots

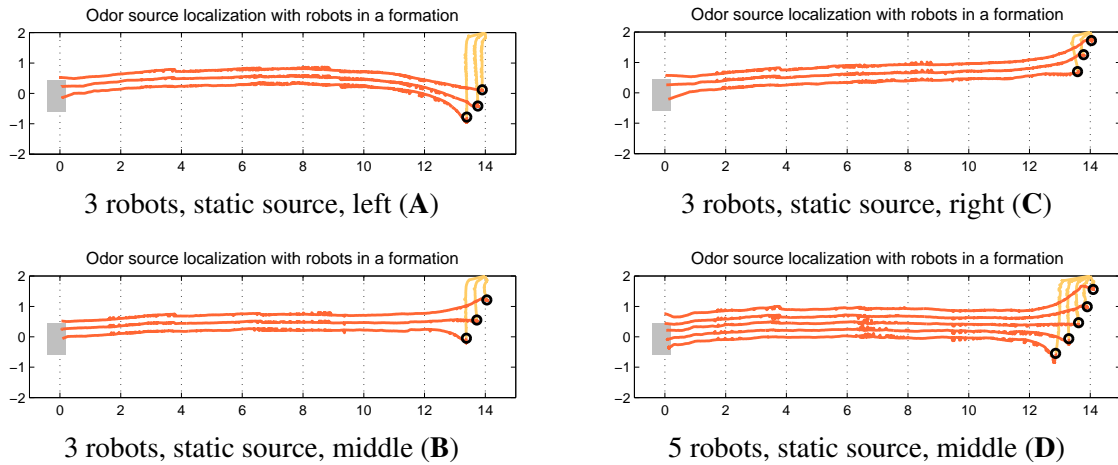


Figure 5.3: Real-robot trajectories produced by crosswind formation algorithm. The gray rectangle represents the target area with the odor source while the black circles denote the starting positions of the robots. The robots go almost straight upwind towards the source, yielding very low distance overheads and high success rates.

was $s = 7.1$ cm/s, and the parameters of the force model were set as follows:

$$\begin{aligned}
 u &= 1 \\
 a &= 1 \\
 r &= \begin{cases} 0.1225 & \text{for the experiments with 3 robots} \\ 0.4225 & \text{for the experiments with 5 robots} \end{cases} \\
 k_v &= 1 \\
 k_h &= 1
 \end{aligned}$$

Note that no attempt was made to systematically optimize these parameters, as the main objective is to demonstrate odor source localization using formations, and not formation control itself.

Before each run, the odor concentration baseline was determined individually for each robot by taking a few measurement samples in fresh air. The sensitivities of the odor sensors were not systematically calibrated, but believed to be approximately the same. Slight sensitivity differences would result in a little drift of the formation in crosswind direction, but hardly affect the performance.

Figure 5.3 shows one run of each setting. No matter where the robots started, the robots found the center of the plume within the first 2 m upwind distance and then continued going straight upwind. The distance overheads are therefore extremely low, as shown in Figure 5.4. The success rate was 100 % in all settings.

In our setup, this algorithm clearly outperforms all bio-inspired and probabilistic algorithms discussed in Chapters 3 and 4 in terms of distance overhead and success rate. The distance overhead for most runs was below 2 %, and for some runs even below 1 %. Included in this overhead is the initial phase in which robots get into the predefined formation shape. Without this, the results would be even closer to the optimal performance.

This is not surprising: with sensors on the left and on the right of the plume, the formation obtains direct feedback about its position with respect to the plume, and can correct for that long before leaving the plume completely. A plume reacquisition phase, as it is used with the bio-inspired algorithms (Chapter 3) and also observed with the robot-centric implementation of the

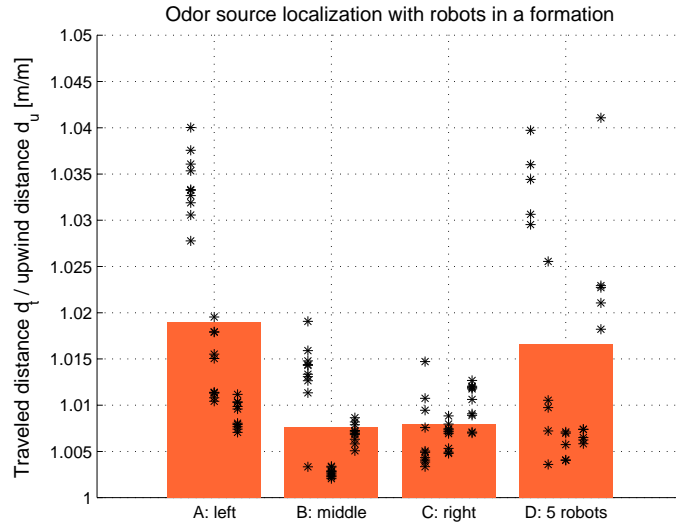


Figure 5.4: Distance overheads (by starting position) of the experiments with three robots. The dots stand for the distance overhead of the robots in individual runs, and are classified by the robot's position in the formation. The bars indicate the mean distance overhead over all runs and all robots.

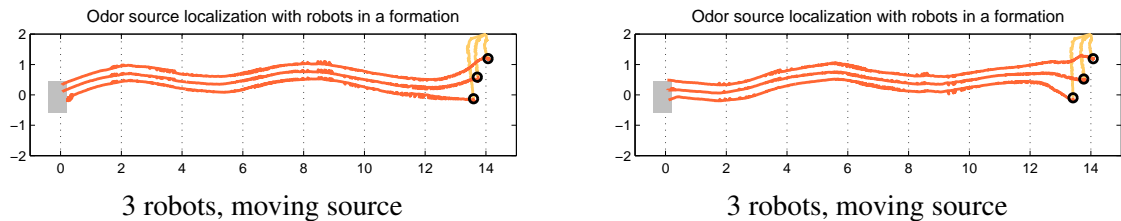


Figure 5.5: Real-robot trajectories produced by the crosswind formation algorithm tracking a source moving in crosswind direction. The gray rectangle represents the target area with the odor source while the black circles denote the starting positions of the robots. The robots nicely follow the movement of the plume.

probabilistic model (Section 4.4), is not necessary any more. The robots keep going upwind without ever losing the plume.

Using 5 instead of 3 robots did not improve the performance in our experiments. There is no a priori reason for which 5 robots should yield worse results. After all, 5 robots collect even more information about the plume than 3 robots, and should therefore do at least an equally good job. However, the information gain when switching from 3 to 5 robots might be tiny, and therefore irrelevant in our setup. Main reason for the performance drop here are presumably the outermost robots, which started at a suboptimal position quite far away from the plume center, and first had to move closer to the center. We believe, however, that increasing the number of robots would be advantageous in settings with a sparser plume.

5.3 Experiments with a Moving Source

Since this algorithm measures the odor concentration at several points at the same time, it is particularly well suited for scenarios with moving sources. With a single sensor, an algorithm is unable to tell with a single measurement in which direction the source moved. Such information can only be deduced from multiple (sequential or parallel) measurements at different locations.

Multi-robot algorithms provide just this: taking several measurements at the same time. This allows the formation to know immediately whether the source moved towards the left or towards the right. The force model takes advantage of that information in that it tries to keep the robots centered around the plume.

We carried out 5 runs with the same algorithm tracking a moving source. The source was thereby moved back and forth by 92 cm in crosswind direction at constant speed. All other parameters of the setup and the algorithm were kept the same. Two of these runs are drawn in Figure 5.5. While all runs were successful, it is not possible to calculate the distance overhead (in the way it is defined in Section 1.11) in this case. The trajectories however reveal that the algorithm works fine.

5.4 Summary and Conclusion

In this short chapter, we presented the *crosswind formation* algorithm, an odor source localization algorithm which is inherently designed for multi-robot systems. In its present form, the algorithm requires at least 2 robots to collaborate. Experiments were carried out with 3 and 5 real robots, and showed that this algorithm achieves close-to-optimal performance in terms of distance overhead and success rate. We also demonstrated the algorithm's robustness in a scenario with a moving source.

The *crosswind formation* algorithm is barely more complex than the bio-inspired algorithms, but in no way inspired by them. Instead, it is based on standard engineering principles, with some inspiration from the boundary exploration behavior observed with the probabilistic algorithms.

Whether robotic teams moving in formations are suitable for the all applications of odor source localization (see Chapter 1) is questionable. In applications that do not require the full flexibility of a multi-robot system, using multiple sensors on the same robot may however yield an interesting performance advantage for a comparatively little price (design, hardware, deployment). If — like the robots here — these sensors are spaced by about half the plume width, a similar algorithm could be implemented and used on a single robot.

6 Comparison

In the last three chapters, we discussed three different classes of algorithms for odor source localization. In Chapter 3, we presented three bio-inspired algorithms, namely *casting*, *surge-spiral* and *surge-cast*. Chapter 4 discussed two vastly different implementations of a probabilistic model for odor source localization, and in Chapter 5, we finally introduced an algorithm based on formations, which is inherently designed for multi-robot systems.

Table 6.1 shows a side-by-side comparison of different properties of the 6 algorithms studied in this thesis, and summarizes their results. Let us discuss this table from top to bottom.

Apart from the *casting* algorithm, which we consider an incomplete odor source localization strategy, all algorithms yield a decent performance in laminar flow. With its 2 % distance overhead and 100 % success rate, the formation algorithm is clearly the winner. The probabilistic algorithms are just slightly better than the bio-inspired algorithms — a difference which won't be important in most applications — and the success rate of all algorithms was above 90 % given that they were properly configured.

Note that these performance values provide a good guideline for the performance to expect in a laminar flow scenario without meandering or obstacles. Since this scenario is very basic, one may however argue that these values are too optimistic for any realistic application. Indeed, the raw values should be taken with a grain of salt and understood as the achievable performance in a simple environment, or a “soft upper bound” on the performance.

For the bio-inspired algorithms, we carried out a number of real-robot experiments with obstacles and turbulent flow, and concluded that such challenges affect the algorithms very differently. While the *surge-spiral* algorithm was able to locate the source with a higher distance overhead, the *casting* algorithm failed completely. Hence, there is no general rule for transforming the raw performance values obtained in laminar flow to more sophisticated scenarios, and certainly a substantial amount of research left to be done.

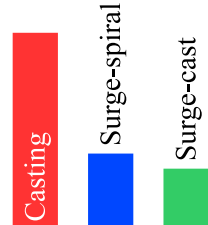
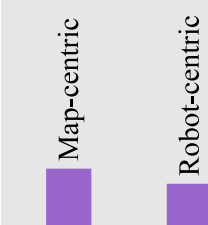




The raw performance values however helped us to assess a number of other properties of the algorithms. In Section 3.5, for instance, we discovered through experiments that the bio-inspired algorithms do not perform well when using multiple robots. As robots are physically competing for space, only a small part of the statistical advantage can be recuperated, and collaboration was found to be a double-edged sword. The use of the moth-inspired algorithms tested in this thesis for plume tracking in a multi-robot system is therefore not recommended.

The opposite is true for the map-centric implementation of the probabilistic model for odor source localization. The use of multiple robots does increase the performance significantly here, and the additional CPU and memory requirements for that are moderate. The robot-centric implementation of the same probabilistic model is again a single-robot algorithm by design, even if the use of multiple sensors (spaced by not more than about 20 cm) would easily be possible.

The crosswind formation odor source localization algorithm finally is inherently for multi-robot systems, and would not work at all when using a single robot only.

One of the most pertinent differences between the algorithms lies in their computational re-

Table 6.1: Comparison of the algorithms presented in the previous chapters. The performance values (distance overhead and success rate) are approximate values based on the real-robot experiments in laminar flow and a holistic assessment.

	bio-inspired	probabilistic		formations
Distance overhead				
Success rate				
Single-robot	•	•	•	○
Multi-robot	○	•	○	•
Structure	state machine	discrete decisions (Bayes inference)		smooth adaptation (control theory)
CPU requirements (MIPS)	0.001	10'000	300	0.01
Memory requirements	few bytes	~ GB	< 50 kB	< 1 kB
Implementation	straightforward	hard	medium	straightforward
Information				
Odor concentration	•	•	•	•
Wind direction	•	•/○	•	•
Plume map	○	•/○	○	○
Map	○	•	○	○
Wheel odometry	○ (•)	○	•	○
Relative position	○	○	○	•
Absolute position	○	•	○	○
Communication	○	•/○	○	•
Plume model	implicit	explicit		implicit
Environments				
Laminar flow	•	•	•	•
Turbulent flow	○	•	•	○ [?]
With obstacles	○	•	•	○ [?]
Integration with				
Path planner	○	•	•	○
Static nodes	○	•	○	○
SLAM	○	•	•	○
Human input	○	•	○	○

quirements¹. In particular, the probabilistic algorithms have a several orders of magnitude higher demand for computational power and memory. The reasons for that are to a large extent intrinsic to the approach:

- ▷ The bio-inspired algorithms are implemented as state machines with two states, whereby state transitions are based on concentration measurements and timeouts. The source code for such a state machine is only about 10 lines long, and straightforward to implement. Apart from conditioning the input signals (odor concentration and wind direction), only counters and thresholds are necessary to run the state machine and decide upon state transitions, and the state information consists of a few numbers only (counters and current state).
- ▷ The crosswind formation algorithm is similarly straightforward to implement. This algorithm is inclined towards control theory: in each loop iteration, it calculates the wheel speed values based on the input values using a predefined formula. The system is thereby smoothly guided towards its optimal state, but perturbed by sensor and actuator noise. The force model employed here requires basic math operations (addition, subtraction, multiplication, division, sin and cos) only, and a small amount of memory to store the measurements communicated by the other team members.
- ▷ Implementing the probabilistic algorithms is far more challenging. The core part consists of several hundred lines of source code and requires careful data structure design, especially for multi-step-ahead evaluation. Each high-level operation on the source probability distribution results in thousands of multiplications and additions, and the evaluation of a single target consists of several such operations. In addition, the state is a probability distribution amounting to a few kilobytes of data. In the map-based implementation, a big chunk of memory is required for the plume model used for the train station scenario. Such entirely data-driven models can easily amount to gigabytes of data.

Despite the big differences, none of the algorithms is outside of what is achievable on current off-the-shelf computers. Prices for computational power in this order of magnitude are low, but a question that remains is that of energy. As mobile robots are most often running on batteries (or other energy sources with a fairly limited capacity), energy is a precious resource. The main question thereby is how the energy required for computation compares to the total energy budget.

As an example, let us compare the *surge-spiral* algorithm with the robot-centric implementation of the probabilistic model. On the Khepera III platform with the Korebot, computation at maximum CPU load accounts for about 5 % of the total energy budget of the robot. Hence, the *surge-spiral* algorithm burns about 5 % less energy on that. The difference in distance overhead between the two algorithms is also about 5 %, but in favor of the probabilistic algorithm. Hence, the two algorithms are energy-wise approximately equal for plume tracking. If the success rate is considered as well, the *surge-spiral* algorithm is about 10 % more efficient. One may further argue, however, that the probabilistic algorithm will be able to tell the location of the source before reaching it, and thereby save energy.

Another significant difference between the algorithms is the information they need (or consider) throughout the plume tracking process. The bio-inspired algorithms are based on odor and wind information only. No positioning information is taken into account by the state machine. (Depending on the implementation, wheel encoder feedback may be used to turn by a given angle, or move forwards for a given distance.) The formation algorithm is similar in that respect, but each robot needs to know the current positions of the other robots relative to its own pose.

¹The numbers provided in Table 6.1 are orders of magnitude for running the algorithm at reasonable speed in a moderately complex scenario.

The probabilistic algorithms offer substantial flexibility with respect to the information they process. Our robot-centric implementation is comparable to the bio-inspired algorithms, except that it uses wheel odometry to infer its movement. This is very precise over short distances, and as the algorithm anyway forgets old information, error accumulation (which is unavoidable with odometry) does not harm the algorithm in any way.

The map-centric implementation works with a much larger set of information. In the train station scenario, it combines the actual measurements with pre-calculated plume propagation information to come up with a belief of the odor source location. The decision step is based on a map of the environment, and assumes that all robots can localize themselves accurately on this map.

Any odor source localization algorithm makes some assumptions about the plume. The weakest assumption is made by the bio-inspired and formation algorithms, which are only based on the fact that odor filaments are mainly transported by advection. This assumption does not explicitly appear anywhere in the algorithm, but is implicitly assumed to hold. The probabilistic model, on the other hand, requires the plume model to be explicitly defined. Even if this may sound like an additional hassle at first sight, it allows for much more flexibility.

The probabilistic algorithms are therefore applicable to almost any type of environment, given that the right plume model is used. In the train station scenario, for example, there are zones with a strong main wind flow and zones with mostly turbulence or whirlwind. All this is captured by the plume model, which allows the robot to take the right decision in all zones.

The bio-inspired algorithms are generally not capable of doing this. The *surge-spiral* algorithm is an exception because the spiraling part can recover from almost any situation. The upwind surge part however is useless in scenarios without a clear main wind flow, and the performance of the algorithm suffers substantially (see Section 3.4). Similarly, the crosswind formation algorithm was not designed with turbulence or obstacles in mind, and its performance in such conditions is expected to be bad.

This is not to say that there are no computationally cheap algorithms for environments with a high degree of turbulence (i. e., without a main wind flow), and/or with obstacles. But we did not study any such algorithm throughout this thesis.

Finally, probabilistic algorithms offer a lot of flexibility for integration with other subsystems typically employed on a mobile robot, as we discussed in Section 4.3.

7 Conclusion

In this thesis, we evaluated and compared 6 odor source localization algorithms for mobile robots. Three algorithms are bio-inspired and imitating the behavior of moths. We tested these algorithms with real robots in a wind tunnel as well as in simulation, and derived their expected performance through a theoretical model. We also studied multi-robot versions of these algorithms. Two algorithms are implementations of a general probabilistic model for odor source localization, which we introduced in this thesis. Both algorithms combine observations (i. e., concentration measurements) through probabilistic filtering using a plume model, and thereby calculate a belief for the location of the odor source. In spite of the algorithms both implementing the same probabilistic model, their implementation is vastly different. While the map-centric implementation features path planning around obstacles, multi-step-ahead evaluation and a completely data-driven plume model, the robot-centric implementation is lightweight with respect to CPU and memory usage. Both implementations were evaluated through simulations, and the latter was also deployed on the real robots in the wind tunnel. The last of the 6 algorithms we studied is based on a crosswind formation, and inherently designed for multi-robot systems.

All experiments with the real robots were carried out with Khepera III robots in an 18 m long wind tunnel, and hence in a systematic environment under repeatable and measurable conditions. The main performance indicators we studied were the distance overhead (which stands in direct relationship with the speed of an algorithm) and the success rate, both of which are applicable to all algorithms. The results are therefore directly comparable. A number of experiments were also carried out in various simulation environments at different abstraction levels. The bio-inspired algorithms as well as the robot-centric implementation of the probabilistic model were tested in Webots (robotic simulator) with simulated Khepera III robots moving through a plume simulated using filaments. The setup was thereby closely matched to the setup in the wind tunnel, allowing us to compare the results to those obtained with the real robots. Other experiments were carried out using dedicated body-less simulations at higher abstraction levels, and under simple plume propagation models.

The algorithms were in general evaluated in an environment with a laminar flow. For the bio-inspired *surge-spiral* algorithm, we additionally carried out real-robot experiments in turbulent flow and with obstacles, and the map-centric implementation of the probabilistic odor source localization model was demonstrated to work in complicated (but known) flow conditions by means of simulation experiments.

The analysis of the results obtained through these experiments allowed us to characterize the performance of the algorithms, as well as their strengths and weaknesses. A (technical) side-by-side comparison of all six algorithms was shown and discussed in Chapter 6.

For the odor source localization research field, a number of conclusions can be drawn from our results.

One of our first observations was that pure (bio-inspired) casting, which was very prominent in the scientific literature before 2006, does not perform well. The algorithm is comparatively slow in simple laminar flow, and — since it heavily relies on accurate wind direction measurements — unable to find the source in turbulent flow. We believe that this strategy is incomplete when

used alone, and findings in behavioral biology confirm that insects use casting in combination with other strategies. This may look like a relatively simple observation, but is actually an important one, given the number of scientific publications that promote pure casting or variations of it.

Second, the class of probabilistic algorithms, which only appeared in the last few years and to which we greatly contribute with this thesis, looks very promising. With the train station scenario in Section 4.3, we demonstrated the flexibility of such a probabilistic model for odor source localization. To our knowledge, no previous work on odor source localization algorithms addressed scenarios this complex, and despite our simulations were carried out at a relatively high abstraction level, we are convinced that this is the right approach to find odor sources in environments for which a lot of information (maps, wind flow, plume propagation, . . .) is available.

Third, we discovered an interesting similarity between insect trajectories and trajectories produced by the robot-centric implementation of the probabilistic model. It is certainly too early to make bold statements, but should our hypothesis discussed in Section 4.4.5 turn out to be true, this would not only be a significant results for behavioral biologists, but also raise questions about what we currently call “bio-inspired”. Indeed, of the six algorithms tested in this thesis, the robot-centric implementation of the probabilistic model best imitates moths — much better than the algorithms dubbed “inspired by the behavior of moths”. Just by looking at the number of neurons of even fruit flies (approximately 50’000), which are definitely not among the most sophisticated insects, it is perhaps even a bit pretentious towards nature to put the label “bio-inspired” onto algorithms consisting of two deterministic states and simple state transitions. (Nevertheless, we did it in this thesis as well.)

Finally, our experiments clearly indicate that multi-robot algorithms can be extremely fast and robust when properly designed. Such algorithms thereby do not need to be complicated, as the crosswind formation algorithm shows. This algorithm was designed from scratch for a multi-robot system, and is based on well-known engineering concepts (mainly from control theory) rather than inspired by observations in nature¹. The probabilistic model is straightforward to extend to multiple robots as well (see Section 4.1.2.4), and has been demonstrated to improve the performance of algorithms with respect to comparable single-robot configurations.

In our opinion, and based on the results and observations made throughout this thesis, trying to forcefully extend the bio-inspired algorithms to multiple robots is a dead end, however. These algorithms are not able to cope with disturbances (see Section 3.6) and therefore inherently suitable for single-robot systems only. Even if a marginal gain can be obtained by using multiple robots, we demonstrated that there are much better approaches to design multi-robot algorithms for odor source localization. One may of course argue that sophisticated collaboration schemes may be able overcome all shortcomings, and one may even classify the crosswind formation algorithm as a bio-inspired algorithm, as it is clearly based on upwind surge. This, however, is a question of semantics and classification.

7.1 The Robots vs. The Dogs

In the introduction, we stated that the ultimate goal of research on odor source localization is to replace animals — most notably dogs — by mobile robots for such tasks. We mentioned a number of potential applications in the introduction (Chapter 1), and stated advantages of using autonomous mobile robots instead of animals. So how does the current state of the art in odor source localization (including the results presented in this thesis) compare to dogs?

¹This is not to say that similar algorithms do not exist in nature, of course.

More as a recreational activity, we once took two trained dogs into the wind tunnel and let them search for an odor source in a similar setup as the robots did. The wind tunnel was configured with the same laminar wind flow at a speed of 1 m/s, but instead of using ethanol, we hid a piece of meat close to the wind inlet². In each run, one of the two dogs was let loose about 15 m downwind from that place.

The result was amazing and, to be honest, quite discouraging with regard to our real robot experiments: it rarely took a dog more than 5 seconds to reach the source. They were basically running straight towards it, despite the fact that they could not see it. From time to time, one could observe them moving their head a bit left or right in order to keep track of the plume, but they hardly slowed down for that. It is needless to say that their success rate was 100 % over about 20 runs. Moreover, the dogs got bored after a while — the task was simply too easy for them, their holder told us smilingly.

With the best odor source localization algorithms, our robot found the source within about 140 seconds. One may of course argue that our Khepera III robot with its maximum speed of about 30 cm/s is simply too slow to be competitive with dogs, and it is true that the robot and the chemical sensor we used for our experiments are not among the top products available on the market. Nevertheless, even with the latest sensor technology and the most agile mobile platform available, we could most probably not have come close to the dogs' performance.

A more relevant question with respect to this thesis, however, is whether our algorithms could compete with the processing that is going on in a dog's brain (at least in our wind tunnel scenario). Hence, let us assume we have a robotic platform and a chemical sensor comparable to a dog and its nose.

When looking at the distance overhead obtained with the robot-centric implementation of the probabilistic model, one would say that we are competitive. We obtained a mean distance overhead of 8 % with the real robots, and less than 2 % in simulation. The distance overhead of a dog's nose when taking into account the occasional left-and-right sweeps was presumably in the 1 – 2 % range as well. But this alone is not an entirely fair comparison. At the forward speed of a dog, our algorithm would have to run about 50 times faster and thereby take 50 times more samples as well. Getting enough computational power for that is certainly not a problem, but whether the algorithm works at such speeds remains yet to be demonstrated.

Since dogs are using vision and their memory as well, they certainly use a much more complex representation of the environment in the brain. Hence, the processing going on in a dog's brain could be closer to the map-centric implementation of the probabilistic model (which integrates room geometry and a more complex plume propagation model), but is likely to be orders of magnitude more sophisticated with regard to the integration of supplementary information.

Fortunately, many industrial applications do not need odor source localization algorithms that are as sophisticated as those used by animals, and a mobile platform to track down a gas leak in a simple environment, for example, could even be built nowadays.

7.2 Outlook and Future Work

As the comparison with the dogs shows, odor source localization is still in its infancy, and a lot of research remains to be done to enable more sophisticated applications.

In our opinion, the focus of odor source localization should move away from bio-inspiration in the way "bio-inspiration" is currently understood by the research community (i. e., simple algo-

²Since dogs remember the location of the meat from one run to the next, we moved it to a different place in each run.

rithms based on state machines). The research community has learned a lot from trying to imitate insect trajectories, but these algorithms have limitations that will hinder them from being successfully used applications later on. This obviously does not mean that these algorithms are completely useless, and algorithms such as the surge-spiral algorithm may still jump in whenever a simple and robust plume tracking algorithm is needed in a simple setting. But judging by our experience, they will only play an anecdotal underpart in the future.

Nevertheless, the design of simple single-robot and multi-robot odor source localization algorithms should still be pushed forwards. Such algorithms could be inspired by more complicated approaches, such as the probabilistic model, or simply be based on standard engineering principles. An interesting direction here would be to design algorithms which focus on exploring the plume boundary, as most information about the source is located there.

A topic that definitely deserves more attention is the probabilistic approach to odor source localization. On a number of issues, we only scratched the surface and left a closer investigation to future work:

- ▷ A fundamental part of any probabilistic odor source localization algorithm is the plume propagation model. In the train station scenario in Section 4.3, we used a completely data driven model for plume propagation which we created using CFD and filament-based plume simulations with arbitrary constraints and parameters. Setting up an appropriate model for a given environment is not that simple, however, and systematic methods for that have yet to be presented, perhaps based on existing work on plume mapping, plume propagation and wind flow modeling.
- ▷ Similarly, the impact of the accuracy and specificity of a plume propagation model on the performance of a probabilistic algorithm should be studied. (How much does an algorithm suffer from inaccuracies in the plume propagation model?) Along with that, malicious behavior may be studied. (Can an attacker deliberately modify the wind flow to prevent his bomb from being detected?)
- ▷ The problem of multiple sources has been stated in Section 4.1.2.5, but is hardly tractable in its current formulation. Future work could suggest solutions to overcome this problem, or study the possibility of tracking down one source after the other.
- ▷ The problem of contaminating sources, mentioned in Section 4.1.2.6, needs experimental validation.
- ▷ The idea of combining the probabilistic odor source localization model with probabilistic simultaneous (self-)localization and mapping (SLAM) has only been crudely outlined in Section 4.1.2.9. We believe that such a model — if computationally tractable — could be key for many future applications of odor source localization. Although the methodology is clear from the perspective of probabilistic mobile robotics, a significant amount of work remains to be done until such systems are operable and well understood.
- ▷ Another problem that has only been discussed superficially (see Section 4.3.6) is that of probabilistic area coverage using plume propagation models. While area coverage per se is a well-studied problem, the inclusion of plume propagation models has not been presented to date.
- ▷ The probabilistic model presented in this thesis (or variations of it) could be implemented in many more ways than the two presented here. It would not come as a surprise to us to see a number of optimizations of our implementations, or completely different implementations

in the future. Different applications (with static or mobile nodes) will call for different implementations.

- ▷ Finally, the issue of integrating human input into the probabilistic model may be tackled.

A very interesting observation we made in our experiments is the similarity of insect trajectories to the trajectories produced by the robot-centric implementation of the probabilistic model, but our hypothesis formulated in Section 4.4.5 remains to be verified by biologists.

Almost all experiments in this thesis were carried out in laminar wind flow with a static odor source. Future work could address meandering, higher degrees of turbulence, moving sources, low intensity sources, or sources that only intermittently emit odors. Such experiments could be carried out in a systematic manner in the wind tunnel, or validated with simulations.

Besides research on the algorithmic part of odor source localization, a lot of research is also needed to develop better chemical sensors, and build more agile robotic platforms.

Finally, researchers should not be shy to work towards real applications of odor source localization systems. Although many challenges remain, a number of solutions are available. As a positive side effect, this would allow the research community to assess the relative importance of open problems, and perhaps even point out new issues.

Bibliography

- [1] G. S. Settles, “Sniffers: Fluid-dynamic sampling for olfactory trace detection in nature and homeland security—the 2004 Freeman scholar lecture,” in *Journal of Fluids Engineering*, ser. Transactions of the ASME, vol. 127, 2005, pp. 189–218.
- [2] A. Rose, Z. Zhu, C. F. Madigan, T. M. Swager, and V. Bulovic, “Sensitivity gains in chemosensing by lasing action in organic polymers,” *Nature*, vol. 434, pp. 876–879, April 2005.
- [3] J. W. Gardner, *Electronic Noses and Sensors for the Detection of Explosives*. Springer, October 2004.
- [4] R. Beccherelli, E. Zampetti, S. Pantalei, M. Bernabei, and K. C. Persaud, “Very large chemical sensor array for mimicking biological olfaction,” in *Proceedings of the 13th International Symposium on Olfaction and Electronic Nose*, M. Pardo and G. Sberveglieri, Eds., vol. 1137, no. 1. AIP, 2009, pp. 155–158. [Online]. Available: <http://link.aip.org/link/?APC/1137/155/1>
- [5] A. J. Lilienthal, D. Reiman, and A. Zell, “Gas source tracing with a mobile robot using an adapted moth strategy,” in *Autonome Mobile Systeme (AMS), 18. Fachgespräch*. GDI, December 2003, pp. 150–160.
- [6] A. J. Lilienthal and T. Duckett, “Gas source localisation by constructing concentration gridmaps with a mobile robot,” in *Proceedings of the European Conference on Mobile Robots (ECMR 2003)*, Warsawa, Poland, 2003, pp. 159–164.
- [7] J. A. Farrell, J. Murlis, X. Long, W. Li, and R. T. Cardé, “Filament-based atmospheric dispersion model to achieve short time-scale structure of odor plumes,” *Environmental Fluid Mechanics*, vol. 2, pp. 143–169, 2002.
- [8] Q. Liao and E. A. Cowen, “The information content of a scalar plume – a plume tracing perspective,” *Environmental Fluid Mechanics*, vol. 2, no. 1-2, pp. 9–34, November 2004.
- [9] A. T. Hayes, A. Martinoli, and R. M. Goodman, “Distributed odor source localization,” *IEEE Sensors Journal*, vol. 2, no. 3, pp. 260–271, June 2002.
- [10] ———, “Swarm robotic odor localization: Off-line optimization and validation with real robots,” *Robotica*, vol. 21, pp. 427–441, 2003.
- [11] W. Li, “Abstraction of odor source declaration algorithm from moth-inspired plume tracing strategies,” in *Proceedings of the 2006 IEEE International Conference on Robotics and Biomimetics (ROBIO 2006)*. IEEE, December 2006, pp. 1024–1029.

- [12] W. Li, M. M. Elgassier, C. Bloomquist, and K. Srivastava, "Multisensor integration for declaring the odor source of a plume in turbulent fluid-advected environments," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*. IEEE/RSJ, October 2006, pp. 5534–5539.
- [13] G. Kowadlo, D. Rawlinson, R. A. Russell, and R. Jarvis, "Bi-modal search using complementary sensing (olfaction/vision) for odour source localization." in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*. IEEE, May 2006, pp. 2041–2046.
- [14] A. J. Lilienthal, H. Ulmer, H. Fröhlich, A. Stützle, F. Werner, and A. Zell, "Gas source declaration with a mobile robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2004)*, 2004, pp. 1430–1435.
- [15] H. Choset, "Coverage for robotics — a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 113–126, 2001.
- [16] N. Correll and A. Martinoli, "Robust Distributed Coverage using a Swarm of Miniature Robots," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 379–384.
- [17] E. Acar, H. Choset, Y. Zhang, and M. Schervish, "Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods," *International Journal of Robotics Research*, vol. 22, no. 7–8, pp. 441–466, 2003.
- [18] J. A. Farrell, S. Pang, and W. Li, "Chemical plume tracing via an autonomous underwater vehicle," *IEEE Journal of Oceanic Engineering*, vol. 30, no. 2, pp. 428–442, April 2005.
- [19] W. Li, "Moth plume-tracing derived algorithm for identifying chemical source in near-shore ocean environments," in *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, November 2007, pp. 2162–2167.
- [20] W. Li, J. A. Farrell, S. Pang, and R. M. Arrieta, "Moth-inspired chemical plume tracing on an autonomous underwater vehicle," *IEEE Transactions on Robotics*, vol. 22, no. 2, pp. 292–307, April 2006.
- [21] R. A. Russell, "Robotic location of underground chemical sources," *Robotica*, vol. 22, no. 1, pp. 109–115, 2004.
- [22] —, "Chemical source location and the robomole project," in *Proceedings of the 2003 Australasian Conference on Robotics and Automation (ACRA 2003)*, December 2003.
- [23] P. Sousa, L. Marques, and A. T. de Almeida, "Toward chemical-trail following robots," *Machine Learning and Applications, Fourth International Conference on*, vol. 0, pp. 489–494, 2008.
- [24] H. Ishida, K. Suetsugu, T. Nakamoto, and T. Moriizumi, "Study of autonomous mobile sensing system for localization of odor source using gas sensors and anemometric sensors," *Sensors and Actuators A: Physical*, vol. 45, no. 2, pp. 153 – 157, 1994.
- [25] A. M. G. Lopes. (2009) EasyCFD - numerical calculation of fluid flow and heat transfer. [Online]. Available: <http://www.easycfd.net>

- [26] H. Ishida, H. Tanaka, H. Taniguchi, and T. Moriizumi, "Mobile robot navigation using vision and olfaction to search for a gas/odor source," *Autonomous Robots*, vol. 20, no. 3, pp. 231–238, June 2006.
- [27] G. Kowadlo and R. A. Russell, "Improving the robustness of naïve physics airflow mapping, using bayesian reasoning on a multiple hypothesis tree," *Robotics and Autonomous Systems*, vol. 57, pp. 723–737, 2009.
- [28] —, "Naive physics for effective odour localisation," in *Proceedings of the 2003 Australasian Conference on Robotics and Automation (ACRA 2003)*, December 2003.
- [29] —, "Robot odor localization: A taxonomy and survey," *The International Journal of Robotics Research*, vol. 27, no. 8, pp. 869–894, 2008.
- [30] M. Vergassola, E. Villermaux, and B. I. Shraiman, "'Infotaxis' as a strategy for searching without gradients," *Nature*, vol. 445, pp. 406–409, January 2007.
- [31] A. J. Rutkowski, R. D. Quinn, and M. A. Willis, "A sensor fusion approach to odor source localization inspired by the pheromone tracking behavior of moths," in *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA 2007)*. IEEE, April 2007, pp. 4873–4878.
- [32] A. J. Rutkowski, M. A. Willis, and R. D. Quinn, "Simulated odor tracking in a plane normal to the wind direction," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*. IEEE, May 2006, pp. 2047–2052.
- [33] H. Ishida, M. Zhu, K. Johansson, and T. Moriizumi, "Three-dimensional gas/odor plume tracking with blimp," in *Conference Proceedings, International Conference on Electrical Engineering*, 2004, pp. 117–120.
- [34] A. Dhariwal, G. S. Sukhatme, and A. A. G. Requicha, "Bacterium-inspired robots for environmental monitoring," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA 2004)*. IEEE, April 2004, pp. 1436–1443.
- [35] C. Lytridis, E. E. Kadar, and G. S. Virk, "A systematic approach to the problem of odour source localisation," *Autonomous Robots*, vol. 20, no. 3, pp. 261–276, June 2006.
- [36] R. A. Russell, A. Bab-Hadiashar, R. L. Shepherd, and G. G. Wallace, "A comparison of reactive robot chemotaxis algorithms," *Robotics and Autonomous Systems*, vol. 45, no. 2, pp. 83–97, 2003.
- [37] L. Marques, U. Nunes, and A. T. de Almeida, "Particle swarm-based olfactory guided search," *Autonomous Robots*, vol. 20, no. 3, pp. 277–287, June 2006.
- [38] F. W. Grasso, T. R. Consi, D. C. Mountain, and J. Atema, "Biomimetic robot lobster performs chemo-orientation in turbulence using a pair of spatially separated sensors: Progress and challenges," *Robotics and Autonomous Systems*, vol. 30, no. 1, pp. 115–131, January 2000.
- [39] A. J. Lilienthal and T. Duckett, "Experimental analysis of gas-sensitive braitenberg vehicles," *Advanced Robotics*, vol. 18, no. 8, pp. 817–834, December 2004.
- [40] —, "Experimental analysis of smelling braitenberg vehicles," in *Proceedings of the IEEE International Conference on Advanced Robotics (ICAR 2003)*. IEEE, 2003, pp. 375–380.

- [41] ———, “Approaches to gas source localisation and declaration by pure chemo-tropotaxis,” in *Autonome Mobile Systeme (AMS), 18. Fachgespräch, Karlsruhe*. Stuttgart, Germany: GDI, December 2003, pp. 161–171.
- [42] D. Martinez, O. Rochel, and E. Hugues, “A biomimetic robot for tracking specific odors in turbulent plumes,” *Autonomous Robots*, vol. 20, no. 3, pp. 185–195, June 2006.
- [43] W. Jatmiko, Y. Ikemoto, T. Matsuno, and T. Fukuda, “Distributed odor source localization in dynamic environment,” in *Proceedings of the 4th IEEE Conference on Sensors (Sensors 2005)*. IEEE, 2005, pp. 254–257.
- [44] W. Jatmiko, K. Sekiyama, and T. Fukuda, “A pso-based mobile robot for odor source localization in dynamic advection-diffusion with obstacles environment,” *IEEE Computational Intelligence Magazine*, pp. 37–51, May 2007.
- [45] ———, “A pso-based mobile sensor network for odor source localization in dynamic environment: Theory, simulation and measurement,” in *IEEE Congress on Evolutionary Computation, 2006. CEC 2006*, 2006, pp. 1036–1043.
- [46] L. P. S. Kuenen and H. C. Rowe, “Cowpea weevil flights to a point source of female sex pheromone: analyses of flight tracks at three wind speeds,” *Physiological Entomology*, vol. 31, no. 2, p. 103, June 2006.
- [47] R. T. Cardé and B. G. J. Knols, “Effects of light levels and plume structure on the orientation manoeuvres of male gypsy moths flying along pheromone plumes,” *Physiological Entomology*, vol. 25, no. 2, pp. 141–150, June 2000.
- [48] K. A. Justus, S. W. Schofield, J. Murlis, and R. T. Cardé, “Flight behaviour of *cadra cautella* males in rapidly pulsed pheromone plumes,” *Physiological Entomology*, vol. 27, no. 1, p. 58, March 2002.
- [49] K. A. Justus and R. T. Cardé, “Flight behaviour of males of two moths, *cadra cautella* and *pectinophora gossypiella*, in homogeneous clouds of pheromone,” *Physiological Entomology*, vol. 27, no. 1, pp. 67–75, March 2002.
- [50] B. Webb, R. R. Harrison, and M. A. Willis, “Sensorimotor control of navigation in arthropod and artificial systems,” *Arthropod Structure and Development*, vol. 33, pp. 301–329, May 2004.
- [51] R. A. Russell, *Odour Detection by Mobile Robots*, ser. World Scientific Series in Robotics and Intelligent Systems. World Scientific Publishing Company, 1999, vol. 22.
- [52] E. Balkovsky and B. I. Shraiman, “Olfactory search at high reynolds number,” *PNAS*, vol. 99, no. 20, pp. 12 589–12 593, October 2002.
- [53] W. Li, J. A. Farrell, and R. T. Cardé, “Tracking of fluid-advected odor plumes: Strategies inspired by insect orientation to pheromone,” *Adaptive Behavior*, vol. 9, no. 3-4, pp. 143–170, 2001.
- [54] P. Pyk, S. B. i Badia, U. Bernardet, P. Knüsel, M. Carlsson, J. Gu, E. Chanie, B. S. Hansson, T. C. Pearce, and P. F. M. J. Verschure, “An artificial moth: Chemical source localization using a robot based neuronal model of moth optomotor anemotactic search,” *Autonomous Robots*, vol. 20, no. 3, pp. 197–213, June 2006.

- [55] J. H. Berlinger and M. A. Willis, "Biologically-inspired search algorithms for locating unseen odor sources," in *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference*. Gaithersburg, MD: IEEE, September 1998, pp. 265–270.
- [56] G. Ferri, E. Caselli, V. Mattoli, A. Mondini, B. Mazzolai, and P. Dario, "A biologically-inspired algorithm implemented on a new highly flexible multi-agent platform for gas source localization," in *Proceedings of the First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechanics (BIOROB 2006)*, February 2006.
- [57] M. Long, A. Gage, R. Murphy, and K. Valavanis, "Application of the distributed field robot architecture to a simulated demining task," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA 2005)*, April 2005, pp. 3193–3200.
- [58] J.-B. Masson, M. B. Bechet, and M. Vergassola, "Chasing information to search in random environments," *Journal of Physics A: Mathematical and Theoretical*, vol. 42, 2009.
- [59] S. Vijayakumaran, Y. Levinbook, and T. F. Wong, "Maximum likelihood localization of a diffusive point source using binary observations," *IEEE Transactions on Signal Processing*, vol. 55, no. 2, pp. 665–676, February 2007.
- [60] T. Zhao and A. Nehorai, "Distributed sequential bayesian estimation of a diffusive source in wireless sensor networks," *IEEE Transactions on Signal Processing*, vol. 55, no. 4, pp. 1511–1524, April 2007.
- [61] M. Wieneke, K. Safenreiter, and W. Koch, "Hazardous material localization and person tracking," *Signal and Data Processing of Small Targets 2008*, vol. 6969, no. 1, p. 696919, 2008.
- [62] M. Wieneke and W. Koch, "Combined person tracking and classification in a network of chemical sensors," *International Journal of Critical Infrastructure Protection*, vol. 2, no. 1-2, pp. 51 – 67, 2009.
- [63] J. A. Farrell, S. Pang, and W. Li, "Plume mapping via hidden markov methods," *IEEE Transaction on Systems, Man and Cybernetics - Part B: Cybernetics*, vol. 33, no. 6, pp. 850–863, December 2003.
- [64] A. J. Lilienthal, M. Reggente, M. Trincavelli, J. L. Blanco, and J. Gonzalez, "A statistical approach to gas distribution modelling with mobile robots – the kernel DM+V algorithm," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.
- [65] C. Stachniss, C. Plagemann, and A. J. Lilienthal, "Gas distribution modeling using sparse gaussian process mixtures," *Autonomous Robots*, vol. 26, no. 2-3, pp. 187–202, April 2009.
- [66] M. Trincavelli, M. Reggente, S. Coradeschi, H. Ishida, A. Loutfi, and A. J. Lilienthal, "Towards environmental monitoring with mobile robots," in *Proc. IEEE Int. Conf. On Intelligent Robots and Systems*, 2008, pp. 2210–2215.
- [67] S. Larionova, N. Almeida, L. Marques, and A. T. de Almeida, "Olfactory coordinated area coverage," *Autonomous Robots*, vol. 20, no. 3, pp. 251–260, June 2006.
- [68] M. Broxvall, S. Coradeschi, A. Loutfi, and A. Saffiotti, "An ecological approach to odor recognition in intelligent environments," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*. IEEE, May 2006, pp. 2066–2071.

- [69] M. Trincavelli, S. Coradeschi, and A. Loutfi, “Classification of odours with mobile robots based on transient response.” in *In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008)*, 2008, pp. 4110–4115.
- [70] T. H. Chung and J. W. Burdick, “A decision-making framework for control strategies in probabilistic search,” in *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA 2007)*. ICRA, April 2007, pp. 4386–4393.
- [71] —, “Multi-agent probabilistic search in a sequential decision-theoretic framework,” in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation (ICRA 2008)*. ICRA, May 2008.
- [72] O. Michel, “Webots: Professional mobile robot simulation,” *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.
- [73] T. Lochmatter, P. Roudit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli, “SwisTrack: A flexible open source tracking software for multi-agent systems,” in *Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008)*. IEEE/RSJ, 2008, pp. 4004–4010.
- [74] T. Lochmatter, P. Roudit, and N. Correll. (2008) SwisTrack (Wikibook). [Online]. Available: <http://en.wikibooks.org/wiki/SwisTrack>
- [75] T. Lochmatter. (2008) Khepera III Toolbox (Wikibook). [Online]. Available: http://en.wikibooks.org/wiki/Khepera_III_Toolbox
- [76] N. Correll, G. Sempo, Y. L. de Meneses, J. Halloy, J.-L. Deneubourg, and A. Martinoli, “SwisTrack: A tracking tool for multi-unit robotic and biological systems,” in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*. IEEE/RSJ, 2006, pp. 2185–2191.
- [77] R. Y. Tsai, “A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses,” *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 4, pp. 323–344, August 1987.
- [78] T. Lochmatter and A. Martinoli, “Tracking odor plumes in a laminar wind field with bio-inspired algorithms,” in *Proceedings of the 11th International Symposium on Experimental Robotics 2008 (ISER 2008)*, ser. Springer Tracts in Advanced Robotics (2009), vol. 54. Athens, Greece: Springer, 2008, pp. 473–482.
- [79] T. Lochmatter, X. Raemy, L. Matthey, S. Indra, and A. Martinoli, “A comparison of casting and spiraling algorithms for odor source localization in laminar flow,” in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation (ICRA 2008)*, May 2008, pp. 1138–1143.
- [80] T. Lochmatter and A. Martinoli, “Understanding the potential impact of multiple robots in odor source localization,” in *Proceedings of the 9th Symposium on Distributed Autonomous Robotic Systems (DARS 2008)*, 2008, to appear.
- [81] V. Braitenberg, *Vehicles: Experiments in synthetic psychology*. Cambridge, MA: MIT Press, 1984.
- [82] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, September 2005.

-
- [83] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, ser. Intelligent Robotics and Autonomous Agents series. The MIT Press, April 2004.
- [84] T. H. Chung, “On probabilistic search decisions under searcher motion constraints,” in *Proceedings of the 2008 International Workshop on the Algorithmic Foundations of Robotics*, 2008.
- [85] A. J. Lilienthal, A. Loutfi, J. L. Blanco, C. Galindo, and J. Gonzalez, “A rao-blackwellisation approach to GDM-SLAM – integrating SLAM and gas distribution mapping,” in *Proceedings of the European Conference on Mobile Robots (ECMR)*, September 19–21 2007, pp. 126–131.
- [86] C. Bruschini, *Guidebook on Detection Technologies and Systems for Humanitarian Demining*. Geneva International Centre for Humanitarian Demining, 2006.
- [87] M. Matsumoto and T. Nishimura, “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator,” *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.
- [88] R. Menzel and M. Giurfa, “Cognitive architecture of a mini-brain: the honeybee,” *Trends in Cognitive Sciences*, vol. 5, no. 2, pp. 62–71, February 2001.
- [89] H. Ikeno, “Flight control of honeybee in the y-maze,” *Neurocomputing*, vol. 58-60, pp. 663–668, 2004.